Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ?

Simon Modeste

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques-Informatique**

Arrêté ministérial : 12.1

Présentée par

MODESTE Simon

Thèse dirigée par **GRAVIER Sylvain**
et codirigée par **OUVRIER-BUFFET Cécile**

préparée au sein **Institut Fourier**
et de **MSTII**

Enseigner l'algorithme pour quoi ?

Quelles nouvelles questions pour les mathématiques ?
Quels apports pour l'apprentissage de la preuve ?

Thèse soutenue publiquement le **5 décembre 2012**,
devant le jury composé de :

Mme BLOCH Isabelle
Professeure des Universités émérite, Université Bordeaux 4, Rapporteur
M. SOPENA Éric
Professeur, Université Bordeaux 1, Rapporteur
Mme CASTELA Corine
Maître de conférence HDR, Université de Rouen, Examinatrice
M. VUILLON, Laurent
Professeur des Universités, Université de Savoie, Examinateur
M. GRAVIER Sylvain
Directeur de Recherche, CNRS et Université Grenoble 1, Directeur de thèse
Mme OUVRIER-BUFFET Cécile
Maître de conférence, Université Paris-Est Créteil, Co-Directrice de thèse

Remerciements

Le problème des remerciements exhaustifs est d'une grande complexité : aucun algorithme exact n'est beaucoup plus efficace que l'énumération de toutes les personnes à la surface du globe. Je vais donc me contenter ici d'une heuristique de remerciement, plus rapide mais sans garantie de n'oublier personne...

Je tiens tout d'abord à remercier Cécile Ouvrier-Buffet et Sylvain Gravier, qui, après avoir accompagné mon travail de Master, ont accepté de m'encadrer pour trois années supplémentaires. J'ai beaucoup appris à leurs côtés. Leurs questions et leurs conseils m'ont guidé tout au long de ce travail. Ils ont su me laisser suivre mes propres pistes et être présents quand je me perdais. J'ai sincèrement apprécié de travailler avec eux, de discuter de nos recherches et de tout le reste. Certains doctorants, surprotégés par leurs encadrants, sont parfois coupés de la réalité (pas si idyllique) de la recherche et de l'université. À l'opposé, Cécile et Sylvain m'en ont fait découvrir les rouages, m'ont impliqué dans des projets et poussé à m'investir dans d'autres. Je suis ravi d'être tombé sur de tels encadrants, qui ne limitent pas leur rôle aux aspects scientifiques.
Merci encore à Sylvain pour tout le reste : les maths, les sms "je fume en bas", les bières, les discussions sur tout, nos désaccords constructifs, son enthousiasme et ses 12 000 projets/seconde.
Merci aussi à Cécile qui m'a fait découvrir les branches et autres chocolats suisses, les coulisses de la DDM et la vodka à la cerise en Pologne ; merci aussi pour les restos partagés en conf et les appels téléphoniques rassurants de ces derniers mois.

Isabelle Bloch et Éric Sopena ont accepté d'être rapporteurs de cette thèse. Je les remercie pour cela, pour leur efficacité, pour leur lecture attentive de mon manuscrit et leurs remarques qui m'ont aidé à clarifier mes idées. Mes remerciements vont aussi à Corine Castela et Laurent Vuillon qui ont accepté de participer au jury de cette thèse.

Une partie de mes recherches s'est appuyée sur des interviews de chercheurs. Je remercie ici, anonymement, tous ceux et toutes celles, de *chercheur 1* à *chercheur 22*, qui ont porté un intérêt à mon travail en acceptant de me donner un peu de leur temps pour répondre à mes questions.

Je remercie les membres de l'Institut Fourier, quelle que soit leur fonction, que j'ai côtoyés durant ces trois années. Je remercie plus particulièrement les thésards avec qui j'ai partagé les tracas et les euphories de la recherche, en particulier Bashar, Aline, Ximena, Laurent, Camille, Nicolas, Thomas, Ariadna, Gunnar, Élise, Marianne, Maxime, Alix et Mathieu (j'en oublie surement). Je remercie aussi les membres de l'IREM de Grenoble pour leur accueil, les enseignants rencontrés dans un cadre de recherche pour leurs retours, les collègues de l'IUT Informatique de Grenoble (en particulier l'équipe des matheux) qui m'ont

guidé dans mes premiers pas dans l'enseignement supérieur, toute l'équipe du département de mathématiques de l'Université de Liège et Michel Rigo en particulier, et mes nouveaux collègues à l'IUFM de Chambéry qui me font découvrir un monde infini de sigles et de réformes...

Quand j'ai découvert l'équipe *Maths à Modeler* et ses actions, j'ai aussi fait la connaissance de nombreuses personnes passionnées, sympathiques et bienveillantes. Je les salue toutes ici.

À chaque fois que j'ai ressenti la solitude du didacticien au milieu des matheux Grenoblois, l'ARDM et le groupe des jeunes chercheurs en particulier m'ont permis de me sentir membre d'une communauté de recherche vivante et de parler en détail de certaines de mes préoccupations. Je remercie aussi les *young researchers* rencontrés à Rzeszów et Faro, mais aussi les *moins young researchers* de ERME, qui m'ont montré que la didactique française n'est pas tout.

Merci encore à tous les amis proches qui m'ont soutenu, questionné et maintenu en contact avec le monde réel. Merci à Claire C, Claire B et Baptiste, merci à Mélaine, Fred M, Fred S et à tous ceux cités plus haut. Une mention particulière doit être faite à Jean Q, qui a été tout à la fois β-testeur de mes idées, soutien moral, support technique, référence scientifique en informatique théorique, co-créateur de projets jamais réalisés et même fournisseur de cloud. Je remercie aussi les acteurs de notre projet Palestinien, avec qui j'ai partagé cette expérience incomparable.

Je remercie enfin toute ma famille (dans un sens très large), à qui je dois beaucoup, et plus spécialement mes parents, pour le goût d'apprendre et l'esprit critique qu'ils m'ont transmis et pour leur soutien moral et logistique sans faille, ainsi qu'Alex et Lucie pour leur humour et peut-être pour m'avoir, malgré eux, donné le goût d'enseigner les maths.

Pour terminer, je remercie Camille, la fille la plus super du monde, de m'avoir supporté (dans presque tous les sens du terme). Sans sa présence à mes côtés, je n'aurai peut-être pas tenu le coup ces trois années. Pour tout le bonheur que tu m'apportes, Camille, merci.

Table des matières

Introduction **11**
1 Introduction . . . 11
1.1 Algorithme, mathématiques et enseignement . . . 11
1.2 Contexte en France . . . 12
2 Problématique et questions de recherche . . . 13
2.1 Une approche épistémologique . . . 14
2.2 L'enseignement de l'algorithmique en France . . . 16
2.3 Développement de situations pour la classe . . . 17

I Une analyse épistémologique du concept algorithme **19**

1 Un premier modèle épistémologique **21**
Introduction . . . 21
1 Autour du concept d'algorithme . . . 22
1.1 Qu'est-ce qu'un algorithme ? . . . 22
1.2 Une formalisation du concept mathématique . . . 25
1.3 Qu'est-ce-que l'algorithmique ? . . . 26
2 Exemples caractéristiques . . . 29
2.1 Plus grand diviseur commun . . . 30
2.2 Cycle eulérien dans un graphe . . . 33
2.3 Tris et permutations . . . 37
3 Aspects de l'algorithme . . . 40
3.1 Cinq aspects fondamentaux . . . 40
3.2 Dualité outil-objet . . . 42
Conclusion et nouvelles questions . . . 43

2 Réflexion sur la pensée algorithmique **45**
1 La « Pensée » ? . . . 46
2 La pensée algorithmique en tant que pensée mathématique . . . 46
2.1 Considérations épistémologiques . . . 46
2.2 Perspectives pour l'enseignement . . . 48
3 Pensée algorithmique versus pensée mathématique . . . 48
3.1 Considérations épistémologiques . . . 48
3.2 Perspectives pour l'enseignement . . . 51
Conclusion et nouvelles questions . . . 53

3 Un modèle de conceptions pour l'algorithme **55**
1 La notion de conception et le modèle cK¢ 55
2 Problèmes et problèmes d'algorithmique 57
2.1 Une notion de problème adaptée 57
2.2 Différentes familles de problèmes 58
2.3 Dialectique outil-objet 59
3 Trois fois deux conceptions pour l'algorithme 60
3.1 Trois paradigmes pour l'algorithmique 60
3.2 Six conceptions pour l'algorithme 62
3.3 Relation entre ces conceptions 64
3.4 Relation aux ASPECTS 66
Conclusion et nouvelles questions 66

Conclusion et utilisation les modèles **69**

II Entretiens avec des chercheurs **71**

Questions et problématiques **73**

4 Entretien avec des chercheurs - Confrontation des modèles **75**
1 Construction et gestion des entretiens 75
1.1 Choix de l'étude par des entretiens 75
1.2 Construction des entretiens 76
1.3 Réalisation et traitement des entretiens 79
2 Analyse des entretiens 79
2.1 Méthodologie pour la validation des modèles 79
2.2 Aspects fondamentaux dans les entretiens 80
2.3 μ-conceptions dans les entretiens 85
Conclusion 89

5 Conceptions des chercheurs - Construction d'un outil **91**
1 Motivations 91
2 Méthodologie 92
2.1 Éléments pour l'analyse des conceptions 92
2.2 ASPECTS et paradigmes en jeu 92
3 Les conceptions des chercheurs 93
3.1 Tableau récapitulatif 93
3.2 Résultats 94
Conclusion - Vers une grille d'analyse 95

Conclusion sur les entretiens **99**

III L'algorithmique au lycée en France **101**

Questions et problématiques **103**

6 Programmes et documents-ressources **105**
1 Organisation des programmes . . . 105
1.1 Le lycée en France . . . 105
1.2 En mathématiques : des objectifs communs pour le lycée . . . 106
1.3 Document ressource . . . 107
2 Grille d'analyse . . . 107
3 Analyse des programmes . . . 108
3.1 Algorithmique dans l'en-tête des programmes . . . 108
3.2 Objectifs pour le lycée . . . 109
3.3 Les algorithmes présents dans les programmes . . . 112
3.4 Conclusion pour les programmes de mathématiques . . . 118
4 Algorithmique de la spécialité ISN . . . 120
4.1 Discours . . . 121
4.2 Contenus . . . 122
5 Documents-ressources pour la seconde . . . 124
5.1 Discussion sur l'algorithmique . . . 125
♢ programme-papier . . . 129
5.2 Activités proposées pour la classe . . . 135
♢ algorithme-instancié . . . 139
♢ programme de modélisation-simulation . . . 142
5.3 Bilan . . . 145
Conclusion . . . 145

7 Ressources du site des IREM en algorithmique **147**
1 Quelles ressources ? Quels outils ? . . . 147
1.1 Les ressources en ligne des IREM . . . 148
1.2 Outils d'analyse . . . 149
2 Analyse . . . 150
2.1 Illustration : différents traitements de la dichotomie . . . 150
2.2 Quels Aspects dans les ressources ? . . . 164
2.3 Quelles conceptions dans les ressources . . . 168
2.4 Quels algorithmes dans les ressources ? . . . 173
2.5 Classification des documents . . . 174
Conclusions . . . 174
2.6 Résumé des résultats . . . 174
2.7 Interprétations . . . 175

8 Manuels du lycée **177**
Introduction . . . 177
1 Méthodologie . . . 178
1.1 Collections de manuels étudiées . . . 178
1.2 Hypothèses . . . 178
1.3 Grille d'analyse . . . 179
2 Résultats . . . 181
2.1 Collection Indice . . . 181
2.2 Collection Transmath . . . 187
2.3 Enseignements de spécialité . . . 192
Conclusion . . . 196

Conclusions **199**

IV Problèmes et situations pour l'algorithme **203**

Introduction : un travail en cours **205**

9 Problèmes fondamentaux - Vers des situations pour l'algorithme **207**
Introduction . . . 207
1 Problèmes fondamentaux . . . 208
1.1 Problèmes et dialectique outil-objet . . . 208
1.2 La notion de problème fondamental . . . 209
2 Quelques propositions de problèmes . . . 211
3 Quels types de situations pour l'algorithme ? . . . 215
3.1 Objectifs . . . 215
3.2 SiRC . . . 216
3.3 Problèmes d'optimisation . . . 216
4 Une proposition de situation . . . 217
4.1 Le problème des pesées . . . 217
4.2 Analyse mathématique du problème . . . 218
4.3 Un problème fondamental . . . 227
4.4 Analyse a priori de la situation . . . 228
Conclusion et perspectives . . . 232

Conclusions et perspectives **233**
1 Résultats . . . 233
1.1 Épistémologie de l'algorithme . . . 233
1.2 Conceptions de chercheurs . . . 234
1.3 Transposition au lycée . . . 235
1.4 Situations pour la classe . . . 236
1.5 Résultats théoriques . . . 237
2 Perspectives . . . 238
2.1 Situations pour l'algorithme . . . 238
2.2 Contraintes et conditions . . . 238
2.3 Conceptions sur l'algorithme . . . 239
2.4 Sémiotique . . . 239
2.5 Modèles et épistémologie . . . 240

Références **241**

Glossaire **243**

Acronymes et symboles **249**

Introduction

Sommaire

1 Introduction 11
1.1 Algorithme, mathématiques et enseignement 11
1.2 Contexte en France 12
2 Problématique et questions de recherche 13
2.1 Une approche épistémologique 14
2.2 L'enseignement de l'algorithmique en France 16
2.3 Développement de situations pour la classe 17

1 Introduction

1.1 Algorithme, mathématiques et enseignement

L'algorithme, un objet ancien...

Souvent considéré comme un objet de l'informatique, l'algorithme est parfois associé à la programmation. Pourtant c'est avant tout un objet des mathématiques. « Avant tout » car la notion d'algorithme précède de beaucoup l'informatique. Le terme trouve d'ailleurs son origine dans le nom « al-Khwarizmi », nom de l'auteur d'un des plus anciens traités d'algèbre connus, datant du IX^{e} siècle.
On peut penser que l'obtention de procédures de résolution de problèmes systématiques et facilement transmissibles est l'une des motivations premières de la recherche d'algorithmes. Les procédures de calcul les plus courantes sont d'ailleurs des algorithmes : addition, soustraction, multiplication et division de nombres entiers et décimaux.

...pour des questions nouvelles...

Bien entendu, la définition actuelle du concept doit beaucoup aux avancées de l'informatique théorique, avec les premiers modèles de calcul automatique, puis pratique, avec les premiers calculateurs. L'informatique qui a aussi enrichi les mathématiques de nouveaux objets et de nouveaux problèmes – le développement des mathématiques discrètes est fortement lié à celui de l'informatique – mais qui a aussi changé les pratiques des mathématiciens en apportant de nouveaux outils. Finalement, la recherche de procédures effectives est, aujourd'hui encore plus, une problématique centrale des mathématiques.
Le rôle de l'algorithme est si important qu'il est au centre d'une discipline propre, l'algorithmique, à l'intersection entre mathématiques et informatique et dont l'essor actuel est considérable.

...et qui interroge l'enseignement des mathématiques

Face à cette évolution de la science mathématique et à la présence grandissante de l'informatique et de ses applications autour de nous, l'enseignement des mathématiques doit être questionné, d'autant plus que l'informatique et l'algorithmique font leur apparition dans les curriculums en France et à l'étranger. En 1998, aux États-Unis, *National Council of Teachers of Mathematics* consacrait son *Yearbook* aux questions de l'apprentissage des algorithmes dans la classe de mathématiques (Morrow & Kenney, 1998). En France, en 2000, la commission Kahane [1] proposait l'introduction d'une « part d'informatique dans l'enseignement des sciences mathématiques et dans la formation des maîtres » (Kahane, 2002). Il faut donc s'attendre à de fortes évolutions des cursus en ce qui concerne l'informatique et l'algorithmique. Dès lors, il est important d'étudier les questions soulevées par une didactique de l'algorithmique.

1.2 Contexte en France

Algorithmique au lycée

Aujourd'hui, on peut trouver dans les programmes de mathématiques du lycée français une part d'algorithmique. Cette introduction a commencé en 2009 dans la classe de seconde, puis les années suivantes dans les classes de première puis terminale. Aujourd'hui l'algorithmique est présente dans les programmes de mathématiques de l'ensemble des filières générales du lycée.

Cette introduction dans l'enseignement mathématique peut entrainer des malentendus entre les communautés mathématique et informatique, notamment concernant la légitimité des enseignants en mathématiques à enseigner l'algorithmique. Et bien que ces derniers puissent penser que l'informatique est loin de leurs préoccupations, l'algorithmique relève bien de problématiques mathématiques.

Naturellement, on peut penser que les motivations pour enseigner l'algorithmique en classe de mathématiques sont multiples : introduction de problématiques récentes (comme cela existe en physique ou en SVT), vivier pour l'utilisation des TICE dans la classe, introduction de la démarche expérimentale, préparation à la création d'un enseignement d'informatique, « mathématiques citoyennes » (comme pour l'introduction des probabilités et statistiques), etc. Il est donc important de se demander ce qui peut être fait dans un enseignement d'algorithmique et dans quel but.

Dans les programmes, l'algorithmique fait l'objet d'un paragraphe à part. Ce paragraphe attribue à l'algorithmique un rôle prépondérant dans l'activité mathématique et transversal aux différents champs des mathématiques. Cependant, à première vue, l'accent est mis essentiellement sur la programmation, la mise en œuvre d'algorithmes et la description d'algorithmes, les différents objectifs attendus des élèves le montrent bien. À aucun moment le lien fondamental entre algorithme et résolution de problèmes n'est explicité et il semble que l'objectif principal d'un enseignement d'algorithmique soit l'apprentissage de la rigueur [2].

1. Réunie en 1999 par le ministère de l'éducation nationale autour de Jean-Pierre Kahane, à la demande d'associations de mathématiciens et de professeurs, pour produire une réflexion en amont de la refondation des programmes, sur l'enseignement des mathématiques de la maternelle à l'université.
2. Ce qui constitue, a priori, un objectif assez flou.

Il nous semble que, plus que l'idée de rigueur, c'est la notion de preuve qui est sous-jacente au concept d'algorithme. Dans le cadre d'un enseignement de mathématiques, il nous semble pertinent de questionner le rôle que peut jouer l'algorithmique dans l'apprentissage de la preuve.

Dans une perspective d'enseignement, il nous paraît fondamental de ne pas réduire les questions d'algorithmique à des questions de programmation, de rédaction, et d'exécution d'algorithmes. Les algorithmes et leur étude jouent un rôle important dans l'activité mathématique. L'activité algorithmique demande d'être analysée de manière détaillée afin de lui donner sa place dans la classe de mathématiques. Dans cette optique, il est important de savoir reconnaître les problèmes qui mettent réellement en jeu une activité algorithmique.

Les mathématiques discrètes (et celles liées à la discrétisation des objets du continu) entretiennent des liens privilégiés avec l'algorithmique. De tels objets sont présents dans les programmes depuis longtemps (en arithmétique par exemple) et d'autres ont été introduits plus récemment (théorie des graphes en série ES par exemple). Cependant, l'introduction de l'algorithme comme objet transversal à toutes les mathématiques pose la question de son rapport aux objets mathématiques des différents chapitres des programmes.

Enseignement de l'Informatique

De plus en plus de voix se font entendre pour demander une introduction de l'informatique comme discipline à part entière dans l'enseignement secondaire en France [3] et à l'étranger. Un premier pas dans ce sens a été fait avec l'ouverture d'un enseignement de spécialité en Terminale Scientifique intitulé *Informatique et Sciences du Numérique (ISN)* et dans lequel l'une des quatre parties du programme est intitulée *algorithmique*.

Si un véritable enseignement d'informatique venait à naître au lycée, l'algorithmique serait certainement un point d'interaction fort avec le cours de mathématiques. Il semble donc indispensable de comprendre les regards que mathématiques et informatique portent sur l'algorithmique. Cela est d'autant plus complexe que l'informatique et son influence sur les mathématiques sont relativement récents.
Une telle introduction soulève une autre question importante concernant les mathématiques nécessaires pour l'informatique et les implications qu'elles pourraient engendrer sur le curriculum.

2 Problématique et questions de recherche

Ce travail s'articule autour de trois axes : la construction d'un modèle épistémologique, l'étude de la transposition didactique et le développement de situations pour la classe. Pour chacun de ces axes, nous commencerons par préciser la problématique générale qui est en jeu. Nous exprimerons ensuite les questions de recherche. Enfin, nous expliciterons les outils méthodologiques associés.

3. Ces derniers mois, on peut citer la tribune du 22 juin 2012 de Serge Abiteboul, Colin de la Higuera et Gilles Dowek, *L'informatique est une science bien trop sérieuse pour être laissée aux informaticiens*, sur `lemonde.fr` et celle du 28 juin 2012 de Jean-Pierre Archambault, Gérard Berry et Maurice Nivat *L'informatique à l'école : il ne suffit pas de savoir cliquer sur une souris* sur `rue89.fr`.

2.1 Une approche épistémologique

Problématique

Pour questionner l'enseignement de l'algorithme, il nous semble nécessaire d'interroger ce qui donne son sens au concept : à quel besoin répond-il et quelles sont ses spécificités ? Cela relève de l'épistémologie de l'algorithme, au sens d'une « connaissance des processus par lesquels les concepts mathématiques se forment et se développent » (Artigue, 1990, p. 243) L'algorithmique, vue comme science des algorithmes, est aussi concernée par une telle analyse épistémologique. Cependant, on parle aussi d'une *activité algorithmique* ou d'une *pensée algorithmique* en mathématiques. La partie *algorithmique* dans les programmes du lycée commence d'ailleurs par cette phrase : « La démarche algorithmique est, depuis les origines, une composante essentielle de l'activité mathématique. » (seconde 2009). Cette activité algorithmique nécessite aussi d'être questionnée du point de vue de l'épistémologie de l'*activité mathématique*, c'est-à-dire dans le sens suivant :

> Au-delà de l'analyse conceptuelle, l'épistémologie intervient à ce niveau sur un plan plus général car ce que vise l'enseignement des mathématiques, ce n'est pas simplement la transmission de connaissances mathématiques, c'est plus globalement celle d'une culture. Il s'agit de faire entrer les élèves dans le jeu mathématique. Mais, qu'est ce jeu mathématique ? Quels sont les processus généraux de pensée qui le gouvernent ? C'est l'analyse épistémologique [...] qui est au premier chef concernée par ces questions. (Artigue, 1990, p. 246).

Nous pensons qu'il est important de mieux comprendre ce qu'est (et peut être) l'activité algorithmique et quel rôle elle joue (ou peut jouer) dans l'activité mathématique.
Pour aborder une telle problématique épistémologique, plusieurs approches existent en didactique des sciences. Les deux principales sont surement l'étude de la genèse historique d'un concept (épistémologie historique) et l'étude d'un concept tel qu'il vit actuellement dans le savoir savant (épistémologie contemporaine). La spécificité du concept d'algorithme nous pousse vers l'étude de son épistémologie contemporaine (et peut-être son histoire récente) tant il a évolué au cours du temps. Chabert (2010) montre bien cette évolution dans l'introduction à l'ouvrage *Histoire d'algorithmes*. Par exemple, concernant le concept actuel d'algorithme :

> Aujourd'hui, principalement sous l'influence de l'informatique, la finitude devient une notion essentielle contenue dans le terme algorithme, le distinguant de mots plus vagues comme procédé, méthode ou technique. [...] Par ailleurs, une autre notion apparaît dans l'idée d'algorithme : l'itération, la récurrence. Ainsi, dans les années 1950, le terme algorithme est essentiellement employé, avec un certain anachronisme, à propos de l'*algorithme d'Euclide*. (Chabert, 2010, p. 6-7)

L'introduction des auteurs à la seconde édition témoigne plus fortement encore de l'évolution actuelle du concept :

> Quinze ans après la première édition, [...] il leur a semblé important de tenir compte de l'évolution récente de l'usage du mot *algorithme* et d'évoquer quelques notions simples d'algorithmique couramment associées. (Chabert, 2010, p. 9-10)

Questions

Dans un premier temps, nous allons donc nous concentrer sur l'étude épistémologique du concept algorithme, de la discipline algorithmique et de l'activité algorithmique. Tout

cela passe par une étude du savoir savant qui permettra un regard sur la transposition didactique. Nous chercherons donc à apporter des réponses aux questions suivantes (nous les reformulerons ensuite dans des cadres théoriques appropriés) :

Question Q1. *Quels sont les éléments constitutifs spécifiques du concept d'algorithme ? Quels sont l'objet et les préoccupations fondamentales de la discipline algorithmique ? Quel lien entretiennent algorithme et preuve ?*

Question Q2. *Peut-on parler d'activité ou de pensée algorithmique dans les mathématiques ? Et dans l'informatique ? Quelles perspectives cela peut-il ouvrir pour une approche didactique ?*

Question Q3. *Peut-on construire un modèle de l'activité algorithmique permettant de questionner la transposition didactique ?*

La problématique d'étudier l'épistémologie contemporaine nécessite une confrontation aux conceptions et aux activités actuelles de la production du savoir savant. Il nous paraît bon que les modèles que l'on peut obtenir en tentant de répondre aux questions Q1, Q2 et Q3 soient mis en écho avec la recherche en mathématiques et informatique. Nous ajoutons la question suivante :

Question Q4. *Les réponses à ces questions, issues du texte du savoir savant, s'accordent-elles avec la conception qu'ont les chercheurs de l'algorithme et de l'algorithmique ? Quels apports à notre questionnement peuvent fournir les points de vue de chercheurs ? Comment interpréter les variations que l'on pourrait entrevoir dans les discours de chercheurs ?*

Méthodologie

Pour répondre aux questions Q1, Q2 et Q3 nous nous appuierons sur la littérature du savoir savant et des écrits réflexifs portant sur l'activité mathématique et l'influence de l'informatique sur celle-ci. À l'aide de ces références, nous proposerons dans un premier temps une courte étude du concept algorithme et de l'algorithmique.
Nous illustrerons cette étude par des exemples d'algorithmes, des problèmes mettant en jeu une activité algorithmique et des problèmes que soulèvent certains de ces algorithmes. Ces exemples permettront d'expliquer les analyses qui suivront.

Cette étude épistémologique du concept nous amènera, dans un second temps, à proposer une réponse à la question Q1 sous la forme d'une grille mettant en avant les aspects importants du concept algorithme. Nous classifierons ces aspects selon une dualité outil-objet. Cela nous permettra de proposer un premier outil pour analyser le rôle attribué à l'algorithme dans toute transposition didactique.

Ensuite, nous essaierons d'apporter un éclairage sur la question Q2, notamment en nous appuyant sur les écrits de chercheurs en mathématiques et en informatique ainsi que des recherches sur le *mathematical thinking*. Cela nous permettra de mettre en avant la nécessité de prendre en compte le point de vue informatique dans un modèle de l'activité algorithmique.

Enfin, il nous faudra dégager un cadre théorique adapté pour modéliser l'activité algorithmique, pour étudier les différentes transpositions didactiques et pour décrire les rapports d'individus ou d'institutions au concept algorithme.

Ce travail épistémologique s'appuyant sur le texte du savoir sera l'objet de la première partie.

Dans la deuxième partie, nous proposerons des réponses à la question Q4. Nous commencerons par reformuler la question à la lumière de la première partie. Pour répondre à ces nouvelles questions nous proposerons des entretiens avec des chercheurs en mathématiques et en informatique afin de mettre à l'épreuve notre modèle épistémologique par comparaison à leurs discours sur l'algorithmique. Cela nous permettra à la fois de confronter notre modèle et de repérer quelles conceptions sont dominantes chez les chercheurs en fonction de leurs disciplines de recherche.

2.2 L'enseignement de l'algorithmique en France

Problématique

Après avoir répondu aux questions épistémologiques soulevées par l'algorithme et l'algorithmique, nous pouvons questionner leur transposition didactique. Il s'agit donc de porter un regard sur le savoir enseigné à l'aide des outils épistémologiques développés :

> Dans cette direction, celle de la vigilance épistémologique, de la prise de distance par rapport à l'objet d'étude, l'analyse épistémologique permet également au didacticien de prendre la mesure des disparités existant entre savoir « savant » [...] et savoir « enseigné ». En effet, alors que l'école vit sur la fiction consistant à voir dans les objets d'enseignement des copies simplifiées mais fidèles des objets de la science, l'analyse épistémologique, en nous permettant de comprendre ce qui gouverne l'évolution de la connaissance scientifique, nous aide à prendre conscience de la distance qui sépare les économies des deux systèmes. (Artigue, 1990, p. 244-245)

Ici, la transposition est double. Il y a tout d'abord un passage du savoir savant au savoir à enseigner, savoir à enseigner qui se manifeste par les instructions officielles des programmes du lycée et par les documents d'accompagnement des programmes. On peut imaginer que cette transposition est elle-même influencée par les transpositions qui pré-existent dans l'enseignement universitaire où l'algorithme et l'algorithmique font l'objet d'enseignements depuis plusieurs dizaines d'années. Une autre transposition a lieu entre le savoir à enseigner et le savoir enseigné. Les manuels jouent un rôle dans cette transposition et leur étude peut mettre en avant une distance entre les instructions des programmes et les tâches attendues de l'élève. Notamment, l'étude quantitative de ces tâches devient possible et peut révéler une sur-représentation de certaines instructions ou au contraire une négligence de celles-ci.

En complément des manuels, les ressources en ligne guident souvent les enseignants. La transposition des instructions des programmes dans ces ressources peut être différente de celle des manuels. De plus, face à ce nouvel objet d'enseignement qu'est l'algorithme, nous pensons qu'une partie des enseignants va avoir besoin de se former "mathématiquement". Là encore, une transposition va entrer en jeu, mais son but et les contraintes qui pèsent sur elle seront bien différents. Étudier ces transpositions devrait permettre de mieux comprendre ce qui peut se passer dans la classe de mathématiques autour de l'algorithmique.

Questions

Nous allons donc nous intéresser à deux transpositions regardées usuellement : celle effectuée dans les programmes et documents d'accompagnement et celle qui apparaît dans les

manuels pour le lycée. Cela nous amène à soulever les questions suivantes :

Question Q5. *Quel rôle et quelle place sont attribués à l'algorithmique dans les programmes de mathématiques et documents d'accompagnement du lycée ?*
Quelle transposition a été effectuée par rapport au savoir savant ? Quel est l'écart avec la transposition en place dans les programmes de la spécialité ISN ?

Question Q6. *Comment cela est-il transposé dans les manuels du lycée ? Quels types de questions et de problèmes sont mis en jeu dans les exercices et activités relatifs à l'algorithmique ?*

Question Q7. *Quels types de ressources en ligne sont mis à la disposition des enseignants ? Pour se former ? Pour construire des activités en classe ?*

Méthodologie

Ces questions seront l'objet de la troisième partie. Pour y répondre, nous commencerons par en proposer une reformulation au regard des deux premières parties. Cela nous permettra de proposer des outils d'analyse adaptés.

Dans le chapitre 6, pour traiter la question Q5, nous proposerons une étude détaillée des programmes des filières générales du lycée ainsi que des documents d'accompagnement.

Pour aborder la question Q7, nous proposerons un corpus de ressources en ligne issues du site des IREM, comportant à la fois des documents pour la formation des enseignants et des activités pour la classe. L'étude de ces ressources sera l'objet du chapitre 7

Enfin, pour répondre à la question Q6, nous effectuerons une étude qualitative et quantitative de ce qui est proposé dans les manuels autour de l'algorithmique. Pour cela nous étudierons quelques collections de manuels proposées par différents éditeurs.

2.3 Développement de situations pour la classe

Problématique

Un des objectifs de ce travail est aussi de proposer des situations pour la classe autour de l'algorithmique. En particulier, nous souhaitons aboutir à la création de situations longues mettant en jeu une forte composante de recherche. Un objectif à long terme serait la constitution de situations fondamentales pour le concept algorithme.
Les résultats de l'étude épistémologique doivent fournir un support pour développer de telles situations. Ils doivent constituer un guide pour mieux comprendre les enjeux liés au concept d'algorithme et construire des situations adaptées. Un autre point important est la recherche de problèmes mathématiques à fort potentiel didactique pour mettre en jeu l'algorithme.

Les résultats concernant l'étude de la transposition au lycée et de celle dans les ressources en ligne peut aussi nous guider dans la construction de situations. Notamment, il nous semble pertinent de chercher à proposer des situations complémentaires à ce qui existe déjà dans l'enseignement et traitant les points que l'on pourrait repérer comme délicats ou peu abordés.

Questions

Nous concentrerons notre attention sur deux questions centrales pour le développement de ressources pour la classe :

Question Q8. *Comment caractériser les problèmes à fort potentiel pour l'algorithmique à l'aide du modèle épistémologique développé ? Quels critères doivent-il vérifier ? Quels problèmes répondant à ces critères peut-on proposer ?*

Question Q9. *Quels éléments épistémologiques sont à prendre en compte pour construire des situations autour de l'algorithme ? Quelles situations peut-on proposer ?*

Méthodologie

Pour répondre à ces questions, nous commencerons par les reformuler en fonction des modèles proposés et des résultats obtenus dans les parties précédentes. Nous proposerons une caractérisation des problèmes mathématiques qui ont un potentiel épistémologique pour l'algorithmique.

Ensuite nous développerons une proposition de situation autour d'un de ces problèmes. Nous montrerons pourquoi cette situation constitue un bon candidat et essaierons d'en déduire certaines caractéristiques.

Pour ce travail, nous nous appuierons sur la *Théorie des Situations Didactiques* (Brousseau, 1998) et sur les travaux autour des *Situations de Recherche en Classe*, en particulier ceux menés dans l'équipe *Maths à Modeler*.

Première partie

Une analyse épistémologique du concept algorithme

Chapitre 1

Un premier modèle épistémologique

Sommaire

Introduction 21
1 Autour du concept d'algorithme 22
1.1 Qu'est-ce qu'un algorithme ? 22
1.2 Une formalisation du concept mathématique 25
1.3 Qu'est-ce-que l'algorithmique ? 26
2 Exemples caractéristiques 29
2.1 Plus grand diviseur commun 30
2.2 Cycle eulérien dans un graphe 33
2.3 Tris et permutations 37
3 Aspects de l'algorithme 40
3.1 Cinq aspects fondamentaux 40
3.2 Dualité outil-objet 42
Conclusion et nouvelles questions 43

Introduction

Ce chapitre a pour but de présenter la notion d'algorithme et l'activité algorithmique afin d'introduire un premier modèle épistémologique.

Avant de répondre aux premières questions posées dans le chapitre d'introduction, nous proposons donc une discussion sur le concept d'algorithme et sur l'algorithmique. Ce sera l'occasion de fixer certaines définitions et d'introduire quelques concepts récurrents. Nous proposerons ensuite quelques exemples qui serviront d'illustrations tout au long du chapitre.

Intentions

Ce chapitre présente de nombreuses notions liées à l'algorithme et au champ de l'algorithmique. Un des objectifs est de présenter l'activité algorithmique et de montrer les grandes questions qui l'animent. Bien qu'une certaine expérience de l'algorithmique soit nécessaire à la lecture de ce chapitre (et de cette partie), un lecteur moins familier à la discipline peut aborder ce chapitre en essayant de retenir les éléments constitutifs importants d'un algorithme et les grandes lignes qui guident l'activité algorithmique. En particulier, il n'est pas nécessaire de maîtriser en détails les points développés dans la section 1 ni l'ensemble

des exemples, preuves, éléments de langages de programmation évoqués à la section 2, pour comprendre les enjeux présents et aborder la suite du document.

Remarque terminologique

Un vocabulaire spécifique à l'algorithmique, issu des mathématiques et de l'informatique, sera utilisé ici. La plupart des termes utilisés sont définis lors de leur introduction et apparaissent au glossaire. Cependant, certains termes classiques et ne pouvant pas prêter à confusion seront supposés connus du lecteur.

Traiter de l'algorithmique demande une vigilance particulière quant au vocabulaire utilisé, notamment certains usages communs du terme algorithme (parfois péjoratifs) sont à mettre de côté ici. De même, certains termes qui peuvent être parfois utilisés au quotidien comme synonymes seront clairement distingués ici et il importera de tenir compte de ces distinctions tout au long de la lecture du document.

1 Autour du concept d'algorithme : vers un modèle épistémologique

Nous commençons par présenter et discuter, dans cette première partie, certaines définitions et notions qui reviendront tout au long de ce chapitre et de la thèse.

1.1 Qu'est-ce qu'un algorithme ?

Plusieurs définitions

Il existe différentes définitions de ce qu'est un algorithme mais nous verrons qu'elles ont toutes un certain nombre de points communs.

Knuth, dans le premier volume de *The Art of Computer Programming* propose la définition suivante :

> The modern meaning for algorithm is quite similar to that of *recipe*, *process*, *method*, *technique*, *procedure*, *routine*, except that the word "algorithm" connotes something just a little different. Besides merely being a finite set of rules which gives a sequence of operations for solving a specific type of problem, an algorithm has five important features :
> **1) Finiteness**. An algorithm must always terminate after a finite number of steps. [...]
> **2) Definiteness**. Each step of an algorithm must be precisely defined ; the actions to be carried out must be rigourously and unambigously specified for each case. [...]
> **3) Input**. An algorithm has zero or more inputs, i.e., quantities which are given to it initially before the algorithm begins. These inputs are taken from specified sets of objects. [...]
> **4) Output**. An algortihm has one or more outputs, i.e., quantities which have a specified relation to the inputs. [...]
> **5) Effectiveness**. An algorithm is also generally expected to be *effective*. This means that all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time by a man using pencil and paper. (Knuth, 1973, p. 4-6)

Il ne s'agit pas tout à fait d'une définition, mais cela soulève les caractéristiques importantes d'un algorithme. La notion de **résolution de problème** est centrale et elle est mise en avant par beaucoup d'auteurs (Garey & Johnson, 1979 ; Beauquier, Berstel, & Chrétienne, 1992 ; Cormen, Leiserson, Rivest, & Cazin, 1994 ; Aho, Hopcroft, & Ullman, 1989). Elle est très liée aux notions d'entrée et de sortie. Les caractéristiques sont d'ailleurs présentes dans la plupart des définitions d'algorithme, mais toutes ne sont pas aussi détaillées. Par exemple, Sedgewick (1988) parle simplement des algorithmes comme des méthodes de résolution de problèmes, adaptées à une implémentation sur ordinateur :

> When one writes a computer program, one is generally implementing a method of solving a problem which has been previously devised. This method is often independent of the particular computer to be used : it's likely to be equally appropriate for many computers. In any case, it is the method, not the computer program itself, which must be studied to learn how the problem is being attacked. The term *algorithm* is universally used in computer science to describe problem-solving methods suitable for implementation as computer programs. (Sedgewick, 1988, p. 4)

Bien que le développement de l'informatique et l'arrivée des ordinateurs aient conféré un rôle de plus en plus grand aux algorithmes, il est important de noter la différence entre algorithme et programme soulevée par Sedgewik. Knuth (1996) insiste lui-aussi sur ce point :

> [...] algorithms are concepts that have existence apart from any programming language. To me the word *algorithm* denotes an abstract method for computing some output from some input, while a *program* is an embodiement of a computational method in some language. (Knuth, 1996, p. 1)

Une méthode effective

Les critères de "finiteness", "definiteness" et "effectiveness" que Knuth donne, se réfèrent tous au fait qu'un algorithme doit pouvoir être **effectivement** mis en œuvre, soit par un opérateur, soit par une machine. Aucune ambiguïté d'interprétation des étapes ne doit être possible, chaque étape doit faire référence à une action élémentaire pour l'opérateur et toute exécution de l'algorithme doit se terminer. La définition de Bouvier, George, et Le Lionnais (2005) met aussi ce côté effectif en avant :

> Un algorithme est une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver avec certitude (c'est-à-dire sans indétermination ou ambiguïté), en un nombre fini d'étapes, à un certain résultat et cela indépendamment des données. (Bouvier et al., 2005, p. 27)

Validité

Toutes ces propriétés d'un algorithme doivent être validées. Plus précisément, un algorithme peut être démontré. Généralement, la **preuve d'un algorithme** se fait selon deux points :

- Preuve de correction : quelle que soit l'entrée, l'algorithme produit une réponse valide.
- Preuve de terminaison : quelle que soit l'entrée, l'algorithme produit une réponse après un nombre fini d'étapes.

Ces preuves s'appuient très souvent sur l'explicitation d'un invariant, c'est-à-dire une propriété reliant divers éléments de l'algorithme qui est vraie à la première étape de l'algorithme et reste vraie tout au long de son exécution. Cela permet souvent de démontrer la correction.
On peut aussi exhiber un variant, c'est-à-dire une fonction à valeur dans $\mathbb{N}$ calculée à chaque étape de l'algorithme et dont la valeur décroit strictement au cours de l'exécution. Cela sert principalement à prouver la terminaison.

Différentes formes

Un algorithme peut être exprimé de diverses façons : langages de programmation, langage courant, diagrammes, etc. Cela dépend de l'opérateur à qui cet algorithme est destiné et de la technologie qui permet d'exécuter l'algorithme :

> Un algorithme peut être spécifié dans un langage humain ou en langage informatique, mais peut aussi être basé sur un système matériel. L'unique obligation est que la spécification fournisse une description précise de la procédure de calcul à suivre (Cormen et al., 1994, p. 3)

Ainsi, un algorithme peut être exprimé dans un langage de programmation, à destination d'un ordinateur. Dans ce cas, le langage peut être décrit de manière très précise par une grammaire et les opérations décrites sont des opérations formelles (elles sont de l'ordre de la manipulation de symboles) [1]. On peut cependant donner une signification à ces manipulations mais c'est l'utilisateur "humain" qui interprète le sens de ces manipulations.
On retrouve dans la plupart des langages de programmation des instructions permettant de contrôler l'enchaînement des opérations. Le plus souvent elles comportent :

- l'instruction conditionnelle de type "IF C THEN I_1 ELSE I_2" qui permet de réaliser la séquence d'instructions I_1 si la condition C est vraie et I_2 sinon,
- la boucle de répétition de type "FOR" qui permet de répéter une séquence d'instructions un certain nombre de fois,
- la boucle conditionnelle de type "WHILE" qui permet de répéter une séquence d'instructions tant qu'une certaine condition reste vraie.

Un algorithme peut aussi être exprimé dans un langage bien moins formalisé, tel que le langue française ou le langage mathématique. Assez souvent, des algorithmes sont proposés dans un langage intermédiaire, inspiré des instructions des langages informatiques mais libéré de certaines contraintes et manipulant directement les objets mathématiques. L'important pour un tel langage est qu'il ne laisse aucune ambigüité pour le destinataire de l'algorithme. Différents niveaux de langage peuvent être utilisés. On parle de pseudo-code lorsque le langage choisi reste proche des langages de programmation.

Certaines preuves, dites preuves algorithmiques, contiennent dans leur expression la description d'un algorithme. Une preuve de ce genre contient un algorithme de résolution du problème posé ainsi que la preuve de l'algorithme. Le même algorithme, présenté avec une preuve de sa correction et de sa terminaison constituerait une preuve complétement équivalente. C'est un aspect de l'algorithme qui questionne la pratique des mathématiciens :

> A characteristic feature of recent development in combinatorics and graph theory is the increasingly important role of algorithms. The main reason for

1. En particulier, un programme ne manipule que des représentations des objets mathématiques qui peuvent être en jeu.

> this is certainly the mushrooming use of computers, but probably in part it is also due to the intrinsic development of the subject. Just how far the "Theorem-Proof" style of mathematics we learned at school will shift to the "Algorithm-Analysis of the algorithm" style is difficult to predict. (Lovász & Plummer, 2009, p. 16-17)

Choix d'une définition

Pour la suite du travail, nous devons fixer une définition du terme *algorithme*. Nous nous appuyons sur les définitions étudiées précédemment, en essayant de prendre en compte tous les éléments importants que nous avons identifiés. Nous posons donc la définition suivante :

> Un algorithme est une procédure de résolution de problème, s'appliquant à une famille d'instances du problème et produisant, en un nombre fini d'étapes constructives, effectives, non-ambigües et organisées, la réponse au problème pour toute instance de cette famille.

Cette définition met bien en évidence la résolution de problèmes et l'effectivité des algorithmes. Nous insistons sur le fait que cette définition est proposée hors de tout système de représentation des algorithmes (ni langage de programmation, ni mots-clés, ni diagrammes, etc).
Les étapes d'un algorithme doivent être organisées, au sens où une série de critères doit permettre de décider de quelle étape doit être appliquée à chaque instant (non-ambigüité de la méthode elle-même).

1.2 Une formalisation du concept mathématique

Toutes les définitions vues précédemment sont souvent suffisantes pour concevoir et étudier des algorithmes. Cependant, au début du XX^{e} siècle, une nécessité de formaliser la notion d'algorithme a fait émerger divers modèles théoriques de ce qu'est une procédure effective :

> Pourquoi la définition formelle du concept d'algorithme s'impose-t-elle ? Pour démontrer justement qu'il existe certains problèmes auxquels aucun algorithme ne peut répondre. (Chabert, 2010, p. 508)

Parmi ces modèles théoriques, tous équivalents, certains sont définis par des familles de fonctions qui représentent les algorithmes (modèles de Kleene, Church et Gödel), d'autres s'appuient sur l'idée de machines qui modélisent les algorithmes (travaux de Turing ou de Post). Aujourd'hui le modèle de machine de Turing et celui de fonctions récursives sont couramment utilisés [2].

Ces modèles ont permis de mettre en évidence que tous les problèmes mathématiques ne peuvent pas être résolus algorithmiquement : il existe des problèmes indécidables. Savoir si un problème est décidable ou non constitue actuellement une question importante pour les mathématiques. Par exemple, en 1970 fut prouvé que le 10^{e} problème de Hilbert est indécidable : il n'existe aucun algorithme pour décider de l'existence de solutions pour les équations diophantiennes.

Le modèle de machine de Turing a aussi permis de classifier les problèmes selon leur difficulté. En particulier, il permet de définir les classes de problèmes P et NP : les problèmes

2. Pour une introduction à ces modèles et aux notions évoquées dans la section 1.2, on pourra, par exemple, consulter (Wolper, 2006)

pour lesquels il existe un algorithme de complexité [3] polynomiale (classe P) et ceux pour lesquels il existe un algorithme polynomial pour vérifier qu'une solution donnée en est bien une (classe NP). La question de savoir si les classes P et NP sont différentes est un des problèmes du millénaire de l'Institut Clay.

Cette notion de modèle théorique se place aussi sur un plan allant au-delà des mathématiques car il est impossible de prouver que les modèles proposés sont équivalents à l'idée de calcul effectif :

> La question de l'équivalence entre tous ces modèles de calculabilité énoncés entre 1931 et 1936 est essentielle. D'une part elle assure la cohérence mathématique des différents résultats obtenus. D'autre part elle intervient dans la question métamathématique suivante : est-ce que le concept d'algorithme correspond à la notion d'algorithme et à l'idée intuitive même d'algorithme ? (Chabert, 2010)

La réponse, positive, à cette question est aujourd'hui appelée thèse de Church. C'est elle qui garantit la cohérence de ces modèles avec l'idée de calcul automatique effectué par un opérateur quelconque.

Il faut remarquer que ces modèles sont bien antérieurs aux premiers ordinateurs et que les modèles proposés de calcul automatisé sont théoriquement réalisables sans électronique : la machine de Turing, par exemple, est un modèle qui peut être réalisé de manière totalement mécanique [4].

1.3 Qu'est-ce-que l'algorithmique ?

L'algorithmique est la science dont l'objet d'étude est l'algorithme. Cette discipline, à la frontière entre les mathématiques et l'informatique, s'intéresse à la création, la description et l'analyse des algorithmes.

> Avec l'introduction du concept d'algorithme [via les modèles théoriques], l'histoire des algorithmes se transforme en l'histoire d'un nouveau domaine scientifique : l'algorithmique. Il ne s'agit pas de rechercher un algorithme pour un problème particulier, mais de chercher à résoudre les problèmes posés par l'étude générale des algorithmes. Cette étude s'est particulièrement développée avec la construction des ordinateurs et l'invention des langages de programmation. (Chabert, 2010, p. 534)

Développement d'algorithmes

Une question fondamentale de l'algorithmique est de savoir créer un algorithme adapté pour résoudre un problème. Cela est souvent lié à la validation de cet algorithme et l'étude de ses propriétés.

Pour le développement d'algorithmes, il existe des principes de fonctionnement génériques qui permettent de décrire des familles d'algorithmes. On peut citer, par exemple, *diviser*

3. Nous précisons ce qu'est la complexité dans la section 1.3.

4. Récemment, les étudiants du projet Rubens, à l'ENS Lyon, ont construit en lego, une machine de Turing 100% mécanique que l'on peut voir fonctionner en ligne : `http://rubens.ens-lyon.fr`.

pour régner, la *programmation dynamique* ou encore les algorithmes *gloutons*. Ces méthodes[5] constituent des guides qui permettent de créer de nouveaux algorithmes.

Bien qu'assez récentes, beaucoup de techniques de « design » d'algorithmes sont déjà standardisées et font l'objet d'enseignements universitaires de premier cycle :

> Cet ouvrage vous enseignera des techniques de conception et d'analyse d'algorithme, de façon que vous puissiez créer des algorithmes de votre cru, prouver qu'ils fournissent la bonne réponse et comprendre leur efficacité. (Cormen et al., 1994, p. 7)

Bien entendu, la création d'algorithmes pose la question de leur expression et de leur interprétation par l'opérateur. En particulier, la description d'algorithmes à destination de machines demande la spécification de langages particuliers et leur étude. Cela constitue une branche importante de l'informatique.

Variable informatique et affectation

Très souvent, l'expression d'un algorithme utilise un type de variable spécifique appelée variable informatique. Une variable informatique modélise un emplacement dans la mémoire et son contenu peut changer. On modifie le contenu d'une variable informatique par l'affectation d'une nouvelle valeur à la variable (ou assignation). On écrit souvent « `var:=val` » ou « `var←val` » pour affecter la valeur `val` à la variable `var`. Les langages de programmation utilisent parfois d'autres symboles pour la représenter, parfois même le symbole « `=` ». Il faut cependant noter que l'opération d'affectation d'une valeur à une variable est non-symétrique, contrairement à l'opérateur d'égalité.

La notion de variable informatique permet de ne conserver que l'information nécessaire au déroulement de la suite de l'algorithme. Elle constitue une modélisation de la mémoire d'une machine.

Dans certaines preuves algorithmiques ou dans certaines preuves d'algorithmes, on peut retrouver des suites qui représentent les valeurs successives prises par une variable informatique. Bien qu'extrêmement présente en algorithmique, la notion de variable informatique n'est pas indispensable à l'expression d'algorithmes.

Complexité

Une question spécifique à l'algorithmique est celle de la complexité des algorithmes. Il s'agit de l'étude du comportement de l'algorithme en fonction de ses entrées. Lorsqu'il s'agit d'étudier le nombre d'étapes de calcul (ou d'opérations considérées comme élémentaires) nécessaires à la résolution du problème on parle de complexité en temps. Lorsqu'il s'agit d'étudier la quantité de mémoire nécessaire, on parle de complexité en espace (la notion de variable informatique permet alors de préciser quelle est l'information que l'on garde accessible et celle dont on n'a plus besoin).

Cette complexité s'exprime en fonction de la taille de l'entrée (souvent la taille nécessaire à la coder). Cette complexité peut être étudiée dans le pire des cas (combien d'étapes demande la division euclidienne de a par b dans le pire des cas en fonction de la grandeur des entiers a et b ?) ou en moyenne (pour tous les entiers a à n chiffres et b à p chiffres,

5. Voir le glossaire pour plus de précisions.

combien d'étapes demande en moyenne la division euclidienne de a par b ?).

De telles questions peuvent être posées dans différents contextes : nombre de lignes nécessaires dans la division euclidienne "papier-crayon", nombre d'additions et de soustractions effectuées par l'ordinateur, etc.

De nombreux outils et techniques sont développés en algorithmique pour pouvoir étudier avec précision ces questions de complexité. Cela entraine aussi l'apparition de nouvelles questions en mathématiques, comme par exemple l'étude de certains types de suites.

Ces différents types de complexité peuvent aussi être définis dans les modèles théoriques. En particulier, avec les classes P et NP présentées précédemment, mais aussi avec d'autres outils apportant une classification des problèmes selon leur difficulté à être résolus algorithmiquement.

Structuration de l'information et codage des données

Les questions de développement d'algorithme et de complexité mènent naturellement à se poser la question de la représentation des instances et celle de l'information manipulée. Par exemple, on ne calculera pas de la même manière le *pgcd* d'entiers donnés en base 10 et d'entiers donnés par la liste des puissances de leurs facteurs premiers.

La structuration de l'information et le choix des opérations autorisées sur ces structures influencent fortement la construction d'un algorithme et modifient la manière dont on va évaluer sa complexité. Par exemple, diviser par deux un entier écrit en base 2 revient simplement à effacer le chiffre des unités.

Le fonctionnement des ordinateurs joue un rôle prépondérant dans les choix de ces structures. Il s'agit tout d'abord de coder l'entrée et les objets en jeu afin de les rendre manipulables par l'ordinateur, mais il s'agit aussi de faire des choix pertinents pour rendre l'algorithme le plus efficace possible. Beaucoup de modèles théoriques d'organisation des informations en algorithmique s'inspirent souvent des structures de données utilisées pour la programmation.

D'autres modèles d'ordinateurs ou de machines sont étudiés et amènent à considérer de nouveaux algorithmes pour des problèmes connus : algorithmes parallèles (où plusieurs opérations peuvent être menées en même temps), algorithmes distribués (ou différents processus agissent indépendamment sur les objets), algorithmes quantiques (pour le modèle théorique d'ordinateur quantique), etc.

Comparaison et optimalité

La comparaison de solutions algorithmiques différentes devient alors un enjeu important. La complexité est un des critères utiles pour cela, mais d'autres critères peuvent être utilisés. Le choix de la représentation de l'information y joue aussi un rôle crucial.

Pour un critère fixé, l'algorithmique s'intéresse aussi à chercher le meilleur algorithme permettant de résoudre un problème. En montrant qu'il n'existe aucun algorithme plus performant que celui proposé pour un problème donné, on obtient un critère permettant de parler de la complexité intrinsèque d'un problème.

Classes de complexité de problèmes

Cette notion de complexité d'un problème permet de classifier les problèmes mathématiques. Les modèles théoriques jouent un rôle central dans cette théorie de la complexité. Nous ne détaillerons pas cet aspect.

Certains problèmes peuvent alors être prouvés comme difficiles : il n'est pas possible de rendre leur résolution rapide. Se pose alors la question de proposer des algorithmes plus efficaces qui, ne pouvant pas donner la solution exacte à toutes les instances, garantissent au moins une solution relativement raisonnable. Le contrôle et les garanties que l'on peut avoir sur les solutions produites devient fondamental. Ces questions sont notamment l'objet des disciplines *optimisation combinatoire* et *recherche opérationnelle* dans lesquelles l'algorithme joue un rôle central. Là encore, les modèles théoriques permettent une classification des problèmes selon leur faculté à être plus ou moins bien approximables.

Algorithmique et informatique

Knuth (1985) perçoit l'algorithmique comme étant l'informatique toute entière.

> For many years I have been convinced that computer science is primarily the study of algorithms. My colleagues don't all agree with me, but it turns out that the source of our disagreement is simply that my definition of algorithms is much broader than theirs : I tend to think of algorithms as encompassing the whole range of concepts dealing with well-defined processes, including the structure of data that is being acted upon as well as the structure of the sequence of operations being performed [...] However, if I had a chance to vote for the name of my own discipline, I would choose to call it Algorithmics.(Knuth, 1985, p. 170)

Selon lui, l'algorithmique peut être vue comme l'étude de ce qui est automatisable. Une telle vision de l'algorithmique s'étend au-delà des mathématiques.

D'autres points de vue présentent plutôt l'algorithmique comme une branche de l'informatique (commune aux mathématiques) parmi d'autres : théorie des langages, architecture des machines, représentation de l'information, etc. Ce point de vue s'appuie sur les spécialisations qui ont eu lieu autour de ces questions. Cependant, l'algorithme est le trait d'union entre toutes ces branches :

> I have emphasized algorithms because they are realy the central core of the subject, the common denominator that underlies and unifies the different branches. (Knuth, 1974, p. 7)

Dans la suite, nous utiliserons le terme *algorithmique* dans un sens plus restreint que celui de Knuth mais nous garderons à l'esprit que les algorithmes sont un élément central de l'informatique. Nous reviendrons sur la position de Knuth et sur cette discussion au chapitre 2.

2 Exemples caractéristiques

Avant d'aller plus loin dans la discussion sur l'épistémologie de l'algorithme. Il nous semble utile de développer quelques exemples pour donner une vision plus concrète des points évoqués précédemment mais aussi pour servir d'illustration aux futures discussions de ce chapitre et des suivants. Pour cela, nous avons choisi trois problèmes classiques pour

lesquels nous allons montrer les algorithmes et questions algorithmiques qui peuvent être abordés. Cet ensemble de problèmes, algorithmes et questions nous semble mettre en avant les principales caractéristiques de l'algorithmique.

Pour chacun des problèmes et thèmes développés, nous allons montrer comment peut s'articuler un questionnement algorithmique. Pour cela nous présenterons des algorithmes et étudierons quelques questions que ces algorithmes peuvent soulever, nous essaierons de montrer comment l'on peut expliciter les algorithmes présents dans certaines preuves algorithmiques et nous évoquerons les questions que pose l'implémentation d'un algorithme dans un langage de programmation.
Le point de vue adopté dans ces exemples est un point de vue mathématique et nous nous attacherons à prouver les résultats présentés.

2.1 Plus grand diviseur commun

Preuve

L'existence du *pgcd* de 2 nombres peut être prouvée comme suit :

Démonstration. Soient a et b deux entiers avec $a \geqslant b$. Notons D l'ensemble des diviseurs communs à a et b. Comme $1|a$ et $1|b$, on a $D \neq \emptyset$, et $D \subset \{1, ..., b\}$. D est borné non vide, donc possède un plus grand élément d noté $pgcd(a, b)$. □

Obtention du pgcd

Le *pgcd* peut être obtenu par l'algorithme d'Euclide. Pour décrire cet algorithme et les suivants nous utiliserons quelques notations propres à l'informatique et à la logique pour simplifier la présentation, notamment :

- $\leftarrow$ ou `:=` pour l'affectation,
- du texte en gras pour les instructions de contrôle,
- des indentations et de lignes verticales pour faciliter la lecture en délimitant les séquences d'instructions,
- les termes **Données** et **Résultat** pour repérer les entrées et sorties de l'algorithme,
- des notations logiques et ensemblistes courantes.

L'algorithme d'Euclide peut alors être décrit de la manière suivante :

Données : a, b, deux entiers
$r_1 \leftarrow a$
$r_2 \leftarrow b$
tant que $r_2 \neq 0$ **faire**
  la division euclidienne donne $r_1 = q.r_2 + r$
  $r_1 \leftarrow r_2$
  $r_2 \leftarrow r$
Résultat : r_1

Algorithme 1 : Euclide

Il n'est pas évident que cet algorithme fonctionne et construise le *pgcd* de a et b. Il est nécessaire de le prouver :

Démonstration. Preuve de correction [6]. Notons a_k, b_k et q_k les valeurs successives prises par r_1, r_2 et q après k pas de la boucle *tant que*. On a pour tout k : $a_k = q_k.b_k + b_{k+1}$ et

6. C'est-à-dire : preuve que l'algorithme donne bien le pgcd.

$a_{k+1} = b_k$. Donc pour tout entier d, $(d|a_k \; et \; d|b_k) \Leftrightarrow (d|a_{k+1} \; et \; d|b_{k+1})$
D'où $pgcd(a_k, b_k) = pgcd(a_{k+1}, b_{k+1})$. Ainsi lorsque l'algorithme s'arrête après n pas : $pgcd(a, b) = pgcd(a_n, b_n) = pgcd(a_n, 0) = a_n$.

Preuve de terminaison [7]. On remarque que la suite (b_k) d'entiers naturels est strictement décroissante. Donc la condition d'arrêt $b_k = 0$ sera vérifiée après un nombre fini d'itérations et l'algorithme s'arrêtera. □

Preuve algorithmique

On aurait aussi pu obtenir ce même algorithme en traduisant, par exemple, la preuve par récurrence de l'existence du *pgcd* ci-dessous :

Démonstration. Démontrons la propriété "$\forall (m, n) \in \mathbb{N}^2$, $pgcd(m, n)$ existe", par récurrence sur $\min(m, n)$.

Initialisation : Soient $(a, b) \in \mathbb{N}^2$ tel que $\min(a, b) = 0$, i.e. $a = 0$ ou $b = 0$. Alors $pgcd(a, b)$ existe et vaut $\max(a, b)$.

Hérédité : Supposons que $\forall (m, n)$ tel que $\min(m, n) \leqslant p$, $pgcd(m, n)$ existe.
Soit un couple (a, b) tel que $\min(a, b) = p+1$. Supposons, par exemple, $b \leqslant a$ et posons r le reste de la division de a par $b = p + 1$. Alors (cf preuve précédente) $pgcd(a, b) = pgcd(b, r)$ et comme $r < p + 1$, par hypothèse de récurrence, $pgcd(a, b)$ existe. □

Pour ce premier exemple, précisons un peu les choses. L'hérédité nous dit que calculer le *pgcd* de deux entiers a et$b$ $(a \leqslant b)$ revient à calculer celui de b avec le reste r de la division euclidienne de a par b. De plus, on a la garantie que le maximum des deux entiers dont on calcule le *pgcd* a diminué strictement.
L'initialisation précise que si l'un des deux entiers est nul, on peut expliciter le *pgcd* qui est égal à l'autre entier.
Cela donne une méthode de calcul du *pgcd* de deux entiers qui correspond à l'algorithme 1.

Programmation

L'algorithme du *pgcd* (algorithme 1) peut être décrit dans un langage de programmation. Par exemple, dans le langage Caml [8], cela peut s'écrire comme suit (pour ce premier programme, nous ajoutons des commentaires, entre `(*` et `*)`, pour clarifier les instructions

7. C'est-à-dire : preuve que l'algorithme donne le résultat après un nombre fini d'étapes.

8. Nous choisissons de présenter les algorithmes dans le langage Caml. Ce langage présente plusieurs aspects qui nous paraissent adaptés à une activité de programmation dans le cadre d'un enseignement de l'algorithmique en mathématiques : typage fort, algorithmes présentés comme des fonctions, facilité à manipuler la récursivité... Ce choix est arbitraire et s'appuie sur un point de vue personnel. La caractérisation (nécessaire !) des langages adaptés à un tel enseignement demanderait une étude poussée qui n'est pas l'objet de cette thèse.

utilisées) :

```
let pgcd_it(a,b) =
  let r = ref (a mod b) in     (* on définit les variables utilisées *)
  let u = ref a in
  let v = ref b in
  while !r<>0 do                    (* tant que le reste est non-nul *)
    u:=!v;
    v:=!r;                     (* les 3 variables sont nécessaires pour
    r:=!u mod !v;                        ne pas perdre d'information *)
  done;
  !v;;
```

Algorithme 2 : Une version itérative de l'algorithme du pgcd en Caml

L'algorithme, une fois programmé permet de faire calculer le *pgcd* de deux entiers à l'ordinateur de manière rapide. Il est ici programmé de façon itérative : l'algorithme est décrit comme une suite d'opérations qui ne font pas d'appel à l'algorithme lui-même.

À l'opposé, on peut programmer une version récursive de l'algorithme d'Euclide de la façon suivante [9] :

```
let rec pgcd_rec(a,b) =
match (a,b) with
  |(0,_) -> b
  |(_,0) -> a
  |(_,_) -> pgcd_rec(b,a mod b);;
```

Algorithme 3 : Une version récursive de l'algorithme du pgcd en Caml

Plusieurs questions pourraient être soulevées concernant ces versions programmées : ces deux programmes sont-ils des expressions correctes de l'algorithme d'Euclide ? Produisent-ils le même résultat quelles que soient leurs données en entrée ? Font-ils les mêmes étapes de calcul ? L'un est-il plus rapide que l'autre ? Plus économe en mémoire ?

L'algorithme récursif 3 peut tout à fait être exprimé hors d'un langage de programmation. On pourrait par exemple l'écrire comme une fonction :

Algorithme 3bis :

$$pgcd(a,b) = \begin{cases} b \text{ si } a = 0 \\ a \text{ si } b = 0 \\ pgcd(b,r) \text{sinon, où } r \text{ est le reste de la division euclidienne de } a \text{ par } b \end{cases}$$

Utilisation de l'algorithme dans une preuve

L'algorithme du *pgcd* peut être réutilisé dans la preuve d'existence des coefficients de Bézout : soient a et b deux entiers et $d = pgcd(a,b)$, il existe deux entiers u et v appelés coefficients de Bézout tels que : $au + bv = d$.

9. la commande `match with` permet de traiter par *reconnaissance de motifs* les différents cas de la forme récursive. Le symbole _ représente une valeur quelconque et les cas sont toujours testés suivant une logique temporelle.

Démonstration. On écrit l'algorithme d'Euclide pour a et b, avec les notations de la preuve précédente. On a :

$$\begin{aligned} a_0 &= q_0.b_0 + b_1 \\ a_1 &= b_0 = q_1.b_1 + b_2 \\ &\vdots \\ a_{n-1} &= b_{n-2} = q_{n-1}.b_{n-1} + b_n \\ a_n &= b_{n-1} = q_n.b_n + 0 \text{ avec } b_n = d \end{aligned}$$

En remplaçant les a_k dans cette suite d'égalités, en partant de la fin on a :

$$\begin{aligned} d = b_n &= b_{n-2} - q_{n-1}b_{n-1} \\ &= b_{n-2} - q_{n-1}(b_{n-3} - q_{n-2}b_{n-2}) = (q_{n-1}q_{n-2} + 1)b_{n-2} - q_{n-1}b_{n-3} \\ &\vdots \\ &= au + bv \end{aligned}$$

□

2.2 Cycle eulérien dans un graphe

Soit $G = (V, E)$ un graphe. Un *cycle eulérien* est un cycle passant une et une seule fois par chaque arête du graphe.

Propriété. *Un graphe admet un cycle eulérien si et seulement si il est connexe et que tous ses sommets sont de degré pair.*

Démonstration. La condition est nécessaire : soit G un graphe admettant un cycle eulérien $\mathcal{C}$. Il est clair que pour que $\mathcal{C}$ parcoure toutes les arêtes le graphe doit être connexe. De plus lorsque $\mathcal{C}$ passe par un sommet il entre par une arête non parcourue et ressort par une autre arête non parcourue. Le nombre d'arêtes incidentes au sommet est donc un nombre pair.

La condition est suffisante : soit G un graphe connexe dont tous les sommets sont de degré pair. On raisonne par induction sur le nombre d'arêtes de G.
Si G n'a pas d'arête alors il ne contient qu'un seul sommet et admet un cycle eulérien.
Sinon tous les sommets de G sont de degré égal ou supérieur à 2. Alors il existe un cycle $\mathcal{C}$ dans G[10]. Si l'on enlève les arêtes de $\mathcal{C}$ du graphe G, on obtient un sous-graphe G' dont tous les sommets sont de degré pair et qui est formé d'un ensemble de composantes connexes $\{G_1, G_2, \ldots, G_n\}$. Chacune de ces composantes connexes G_i a un nombre d'arêtes strictement plus petit que celui de G et ses sommets sont de degré pair. Donc par induction, chaque G_i contient un cycle eulérien $\mathcal{C}_i$. Reste à reconstruire un cycle eulérien pour G. Pour cela on parcourt le cycle $\mathcal{C}$ et lorsque que l'on rencontre un sommet d'une composante connexe G_i non parcourue, on parcourt $\mathcal{C}_i$ puis on reprend le parcours de $\mathcal{C}$. □

10. Tout graphe dont les sommets sont de degré supérieur ou égal à 2 contient un cycle. Cela peut se montrer avec un algorithme.

Algorithme de construction

Cette preuve cache en fait un algorithme de construction d'un cycle eulérien que l'on peut écrire comme suit :

Données : $G = (V, E)$ un graphe
`Cycle-eulérien(`G`)`=
(On va construire un cycle eulérien $\mathcal{C} = [u_1, ..., u_m]$ *avec* $u_i \in V$*)*
s un sommet de G
$\mathcal{C} := [s]$
tant que *il existe* $(q, t) \in E$ *avec* q *le dernier sommet de* $\mathcal{C}$ *et* $(q, t) \notin \mathcal{C}$ **faire**
  ajouter t à $\mathcal{C}$
(on a un cycle de G*)*
pour chaque *composante connexe* G_i *de* G *privé des arêtes de* $\mathcal{C}$ **faire**
  $\mathcal{C}_i$:=`Cycle-eulérien(`G_i`)`
parcourir $\mathcal{C}$ et insérer $\mathcal{C}_i$ dans $\mathcal{C}$ au premier sommet de G_i rencontré
Résultat : $\mathcal{C}$

Algorithme 4 : Construction d'un cycle eulérien

Algorithme de reconnaissance

De plus, à partir de la propriété que nous venons de démontrer et qui donne une caractérisation des graphes contenant un cycle eulérien, on peut construire un algorithme de reconnaissance. Il suffit de vérifier que chaque sommet est de degré pair et que le graphe est connexe :

Données : $G = (V, E)$ un graphe
(on construit l'ensemble T *des sommets qui sont dans la même composante connexe et de degré pair)*
s un sommet de G
$T := \{s\}$
si $deg(s)$ *impair* **alors**
  Résultat : $faux$
sinon
  tant que *il existe* $(u, v) \in E$, $u \in T$, $v \notin T$ *et* $deg(v)$ *pair* **faire**
    ajouter v à T
si $T = E$ **alors**
  Résultat : $vrai$
sinon
  Résultat : $faux$

Algorithme 5 : Test de l'existence d'un cycle eulérien

Complexité et types de données abstraits

En revenant sur l'algorithme 4, on peut se poser la question de sa complexité. C'est-à-dire, on peut se demander combien d'étapes sont nécessaires à la construction d'un cycle eulérien en fonction de la taille du graphe donné [11]. L'algorithme 4 prend en entrée un graphe et manipule cet objet mathématique sans nécessité de préciser quelles sont les opérations

11. En ce qui concerne les graphes comme données d'un algorithme, la taille est représentée soit par le nombre de sommets, soit par le nombre d'arêtes. Les deux cas sont liés par le fait que le nombre d'arêtes est au plus de l'ordre du carré du nombre de sommets.

élémentaires que l'on peut faire avec. Dès lors que l'on se pose la question de la complexité de l'algorithme, il faut préciser quelles opérations seront considérées comme opérations "de base" ou unitaires pour l'évaluation.

Pour décrire la complexité des algorithmes de ce chapitre, nous utiliserons les notations asymptotiques de Landau $\mathcal{O}$, Ω et Θ, issues de l'analyse asymptotique de fonctions et courantes en algorithmique [12]

Supposons que tester si une arête est dans $\mathcal{C}$ est une opération de base, ainsi que l'ajout d'un élément à $\mathcal{C}$, l'insertion d'un cycle dans $\mathcal{C}$ à un endroit donné.
Supposons aussi que l'on puisse déterminer les composantes connexes d'un graphe linéairement en fonction du nombre d'arêtes [13]. Notons $c(n)$ le nombre d'étapes élémentaires effectuées par l'algorithme pour un graphe à n arêtes. L'algorithme effectue une première boucle qui effectue autant d'étapes qu'il y aura d'éléments dans $\mathcal{C}$. Ensuite, il recherche les composantes connexes G_i du graphe G avec les arêtes restantes, ce qui demande un nombre d'étapes en $\mathcal{O}$ de ce nombre d'arêtes restantes, et applique l'algorithme à chacune des composantes. Enfin il parcourt $\mathcal{C}$ pour insérer les cycles eulériens obtenus dans les G_i.
Finalement, on a la relation :
$c(n) = 2|\mathcal{C}| + \mathcal{O}(n) + \sum_{G_i} c(n_i)$ où n_i est le nombre d'arêtes dans G_i.
Si un graphe G ne produit pas de G_i il vérifie $c(n) = 2n + \mathcal{O}(n) = \mathcal{O}(n)$. Et par récurrence, on obtient que quel que soit G, $c(n) = \mathcal{O}(n)$.

Pour éviter d'avoir recours à la recherche de composantes connexes, on peut modifier l'algorithme 4 en notant que l'on peut rechercher les cycles eulériens des composantes connexes non parcourues en recherchant simplement une arête qui n'a pas été visitée et qui n'est donc pas encore dans le cycle $\mathcal{C}$. Cet algorithme est appelé algorithme de Hierholzer.
On peut aussi préciser l'algorithme pour qu'il dise si le graphe est eulérien ou non. Cela donne l'algorithme ci-dessous, qui teste si oui ou non le graphe est eulérien, et qui retourne un cycle eulérien $\mathcal{C}$ dans le cas favorable et un sommet de degré impair s ou deux sommets de composantes connexes différentes dans le cas défavorable. Le cycle eulérien, le sommet de degré impair, ou le couple de sommets retournés par l'algorithme permettent de vérifier que la réponse de l'algorithme est correcte. Ils sont appelés certificats [14].

Cet algorithme s'appuie sur l'algorithme ci-dessous de recherche dans un graphe de cycles

12. Rappelons leur définition (en $+\infty$) :
$f(x) = \mathcal{O}(g(x)) \Leftrightarrow \exists A$ et $k > 0$ tels que $\forall x > A, |f(x)| \leq k\,|g(x)|$,
$f(x) = \Omega(g(x)) \Leftrightarrow \exists A$ et $k > 0$ tels que $\forall x > A, |f(x)| \geq k\,|g(x)|$,
$f(x) = \Theta(g(x)) \Leftrightarrow \exists A$, $k_1 > 0$ et $k_2 > 0$ tels que $\forall x > A, k_1\,|g(x)| \leq |f(x)| \leq k_2\,|g(x)|$.
On pourra consulter (Beauquier et al., 1992) pour plus de détails.

13. Ce qui est réalisable par un parcours en profondeur par exemple.

14. De manière générale, lorsqu'un algorithme est exécuté sur une instance d'un problème, il peut parfois fournir une information supplémentaire permettant de vérifier "rapidement" que la solution est correcte (dans notre cas, le cycle ou le sommet de degré impair). Une telle information est appelée un *certificat*. Les algorithmes avec certificats jouent un rôle important dans la théorie de la complexité.

n'ayant que des sommets de degré pair ou d'un sommet de degré impair :

Données : G un graphe, v_0 un sommet, e une arête incidente à v_0
`Chercher-cycle(`G,e,v_0`)`=
(On cherche à construire un cycle $\mathcal{C}$ de G contenant e et uniquement des sommets pairs ou à trouver un sommet de degré impair.)
tant que *cela est possible* **faire**
  Parcourir G depuis le sommet v_0 en commençant par e en retirant chaque arête visitée.
si *le dernier sommet visité est* v_0 **alors**
  Résultat : (Oui, $\mathcal{C}$) où $\mathcal{C}$ est le cycle parcouru.
sinon
  Résultat : (Non, w) où w est le dernier sommet visité.

Algorithme 6 : Une recherche de cycle

On peut maintenant décrire l'algorithme voulu pour tester si un graphe est eulérien :

Données : $G = (V, E)$ un graphe
(On cherche à construire un cycle eulérien $\mathcal{C}$)
$\mathcal{C} := \emptyset$
tant que *il existe* $(u, v) \in E$ *avec* $u \in \mathcal{C}$ *et* $(u, v) \notin \mathcal{C}$ **faire**
  $H := G$ privé des arêtes de $\mathcal{C}$
  si `Chercher-cycle(`G,u,(u, v)`)` *est de la forme (Oui, $\mathcal{C}'$)* **alors**
    insérer $\mathcal{C}'$ dans $\mathcal{C}$
  sinon
    Résultat : `Chercher-cycle(`G,u,(u, v)`)`
si *l'ensemble des arêtes de G est dans $\mathcal{C}$* **alors**
  Résultat : (Oui,)
sinon
  Résultat : (Non, (s, t)) où s est un sommet de $\mathcal{C}$ et t un sommet qui n'est pas dans $\mathcal{C}$.

Algorithme 7 : Reconnaissance des graphes eulériens avec certificats

La correction et la terminaison de cet algorithme demanderaient à être prouvées. Nous ne donnerons pas les détails ici mais seulement un invariant qui permet ces preuves :
Après chaque pas de la boucle *tant que*, soit on a trouvé un sommet de degré impair, soit $\mathcal{C}$ est un cycle sans répétition d'arête dont la longueur a augmenté strictement.

On peut aussi évaluer la complexité de cet algorithme. On trouve une complexité linéaire en fonction du nombre d'arêtes, en considérant comme élémentaires les opérations suivantes :

- trouver les arêtes non-parcourues adjacentes à un sommet,
- trouver un sommet de $\mathcal{C}$ incident à une arête non-visitée,
- insérer un nouveau cycle dans $\mathcal{C}$.

Un ensemble muni d'opérations élémentaires est appelé un type de données abstrait. C'est un modèle qui permet d'évaluer la complexité (comme nous l'avons fait pour étudier la complexité de l'algorithme 4) sans entrer dans les détails de codage des données et de programmation [15]. Ici, deux types de données abstraits sont en jeu : celui permettant de représenter le graphe G et celui permettant de représenter le cycle $\mathcal{C}$.

15. Pour plus de détails sur les types de données abstraits, voir (Aho et al., 1989)

Programmation et structures de données

Pour pouvoir programmer l'algorithme précédent en gardant la complexité linéaire "théorique", il est nécessaire de trouver une représentation du graphe et des objets manipulés sous la forme de structures de données qui respectent les contraintes des types de données abstraits du paragraphe précédent. Un tel programme peut être réalisé, Fleischner (1991) en propose une version qui utilise des listes doublement chaînées.

Notons que la version de l'algorithme donnée ici n'est pas directement due à Hierholzer. Elle est dérivée de sa preuve (algorithmique), publiée en 1873, de la caractérisation des graphes eulériens (énoncée en 1736 par Euler).

2.3 Tris et permutations

Générateurs du groupe des permutations

Rappelons quelques propriétés classiques du groupe des permutations $\mathfrak{S}_n$ (ou groupe symétrique). $\mathfrak{S}_n$ peut être vu comme l'ensemble des bijections de $[\![1, n]\!]$ dans $[\![1, n]\!]$. On écrira une permutation σ de $\mathfrak{S}_n$ sous la forme $[\sigma(1), ..., \sigma(n)]$. On notera (i, j) la permutation qui envoie i sur j et j sur i en laissant fixes les autres éléments. Une telle permutation, qui échange deux éléments, est appelée transposition. Une transposition de la forme $(i, i+1)$ est dite élémentaire.

Propriété. *Le groupe des permutations est engendré par les transpositions.*

Démonstration. Par induction sur n, montrons que pour tout $n \geqslant 2$, toute permutation s'écrit comme produit de transpositions.
Si $n = 2$, alors il n'y a que deux permutations : l'identité et la transposition $(1, 2)$. La propriété est vraie.
Soit p le plus petit entier tel que $\mathfrak{S}_p$ ne soit pas engendré par les transpositions.
Alors $p > 2$. Soit $\sigma = [\sigma(1), ..., \sigma(p)] \in \mathfrak{S}_p$ et i l'indice qui vérifie $\sigma(i) = p$. Posons $\tau = \sigma \circ (i, p)$. On a $\tau(p) = p$ donc $\tau_{|[\![1,p-1]\!]}$ est un élément de $\mathfrak{S}_{p-1}$. Par hypothèse, $\tau_{|[\![1,p-1]\!]}$ peut donc se décomposer en produit de transpositions :

$$\tau_{|[\![1,p-1]\!]} = \prod_k t_k$$

Comme les transpositions de $\mathfrak{S}_{p-1}$ peuvent être vues comme des transpositions de $\mathfrak{S}_p$, on obtient :

$$\sigma = \prod_k t_k \circ (i, p)$$

C'est une contradiction avec l'existence de p. Toute permutation se décompose comme produit de transpositions. □

Propriété. *Le groupe des permutations est engendré par les transpositions élémentaires.*

Démonstration. (version 1)
Il suffit de montrer que toute transposition peut s'écrire comme produit de transpositions élémentaires, le reste découle de la propriété précédente.
Soit (i, j) une transposition de $\mathfrak{S}_n$ (avec $i < j$).
On peut écrire :
$(i, j) = (i, i+1) \circ (i+1, i+2)...(j-1, j) \circ (j-1, j-2)...(i+2, i+1) \circ (i+1, i)$ □

Démonstration. (version 2)
On adapte la démonstration pour les transpositions aux transpositions élémentaires. Par induction sur n, on montre que pour tout $n \geqslant 2$, toute permutation s'écrit comme produit de transpositions élémentaires. Il suffit de remplacer $\tau = \sigma \circ (i, i+1) \circ (i+1, i+2)...(p-1, p)$ dans la démonstration. □

Algorithmes sous-jacents et tris

Ces démonstrations constituent toutes trois des preuves algorithmiques. On pourrait facilement expliciter les algorithmes qu'elles sous-tendent pour produire la décomposition d'une permutation. On peut aussi y voir des algorithmes de tri [16] : étant donnée une liste d'entiers désordonnée, on veut produire la liste ordonnée de ces nombres (les opérations autorisées sont la comparaison de deux entiers et leur transposition dans la liste).

Nous n'allons pas détailler ces algorithmes, mais seulement préciser que la première démonstration induit l'algorithme du *tri par sélection* et les deux suivantes des algorithmes proches du *tri à bulles*. Pour trier n éléments, ces tris demandent un nombre de comparaisons d'éléments de l'ordre de n^2.

Tri rapide, complexité au pire et en moyenne

Voici un autre algorithme de tri. Il prend en entrée une liste $L = (a_1, a_2, ..., a_n)$ d'entiers et retourne la liste triée de ces entiers.
Le *tri rapide* est un algorithme du type *diviser pour régner*. Son principe est de prendre le premier entier de la liste, qu'on appellera le *pivot*, et de séparer les éléments de la liste qui sont plus petits que le pivot de ceux qui sont plus grands.
On trie ensuite séparément ces deux listes d'entiers et il n'y a plus qu'à les réécrire bout à bout avec le pivot au milieu.
Le tri de ces deux listes se fait en réutilisant le tri rapide sur chacune d'elles. Cela donne l'algorithme suivant :

Données : $L = (a_1, a_2, ..., a_n)$ une liste d'entiers
`Tri-rapide(`L`)`$=$
si $|L| \leqslant 2$ **alors**
  Trier L en L' en comparant les éléments si nécessaire
  Résultat : L'
sinon
  pour *i allant de 2 à n* **faire**
    si $a_i \leqslant a_1$ **alors**
      Mettre a_i dans $L_{\leqslant}$
    sinon
      Mettre a_i dans $L_{>}$
    $L_1 =$`Tri-rapide(`$L_{\leqslant}$`)`
    $L_2 =$`Tri-rapide(`$L_{>}$`)`
    Résultat : (L_1, a_1, L_2)

Algorithme 8 : Tri rapide, version récursive

Cet algorithme peut être prouvé par récurrence sur le nombre d'éléments dans L.

16. À condition de préciser comment l'on devine la position attendue i d'un élément en position $\sigma(i)$ avant le tri.

On peut prouver que la complexité au pire de cet algorithme est en $\mathcal{O}(n^2)$ comparaisons pour trier une liste de n éléments (ce cas est atteint pour le cas d'une liste déjà triée, par exemple [17]). Cependant, cet algorithme est très utilisé car sa complexité en moyenne est est en $\mathcal{O}(n\log(n))$ si l'on suppose toutes les permutations d'éléments équiprobables [18].

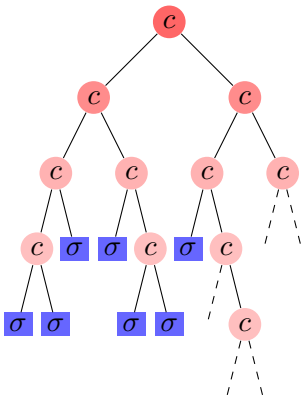
L'algorithme, tel qu'il est écrit ici, crée de nombreuses listes tout au long de son exécution. On peut améliorer sa complexité en espace en remarquant que l'on peut faire ce tri *sur place*, c'est-à-dire en utilisant uniquement l'espace de la liste de départ. On peut aussi réécrire l'algorithme pour qu'il soit sous une forme itérative.

Algorithmes optimaux

Une autre raison de l'utilisation courante du tri rapide est qu'il est optimal. On ne peut pas résoudre le problème du tri par comparaison en mieux que « $n\log(n)$ ». Autrement dit le problème du tri est de complexité « $n\log(n)$ » :

Propriété. *Tout algorithme de tri par comparaison a une complexité au pire et en moyenne en* $\Omega(n\log(n))$.

Démonstration. Tout algorithme de tri par comparaison peut être représenté par son arbre de décision. C'est un arbre binaire où chaque nœud de l'arbre représente une comparaison "c" et chaque branche, le résultat "$\leqslant$" (à gauche) ou "$>$" (à droite) de cette comparaison. Voici un exemple d'arbre de décision :



Les feuilles (carrés) représentent les réponses de l'algorithme "σ". Lorsque il trie une liste de n éléments, un algorithme doit pouvoir différencier toutes les permutations possibles des éléments. Le nombre de feuilles est donc au moins $n!$.
Or, un arbre binaire à p feuilles a toujours une profondeur maximale en $\Omega(\log(p))$ et une profondeur moyenne en $\Omega(\log(p))$.
Ici, $p = n!$ et l'on peut conclure en utilisant $\log(n!) = \Theta(n\log(n))$ que l'on a une profondeur au pire et en moyenne en $\Omega(n\log(n))$ □

Propriété. *Pour la complexité en moyenne, le tri rapide est optimal pour le problème du tri par comparaison.*

17. Pour éviter que cela ne se produise trop souvent, on peut tirer au hasard l'élément qui va servir de pivot.
18. Pour plus de détails, voir par exemple (Cormen et al., 1994).

3 Aspects de l'algorithme

Ces premiers exemples et premières discussions sur le concept d'algorithme nous permettent de relever un certain nombre de points essentiels regroupés en cinq grands groupes. Nous les appellerons les **Aspects** de l'algorithme. Ils constituent un premier modèle épistémologique du concept qui doit permettre de mettre en avant les éléments constitutifs spécifiques du concept d'algorithme en répondant à Q1 :

Question Q1. *Quels sont les éléments constitutifs spécifiques du concept d'algorithme ? Quels sont l'objet et les préoccupations fondamentales de la discipline algorithmique ?*

3.1 Cinq aspects fondamentaux

L'aspect problème

Un algorithme résout un problème, c'est-à-dire, répond à une question précise posée pour une famille d'instances. Pour chaque instance, le même algorithme donne une réponse à la question du problème. On parle d'entrée et de sortie de l'algorithme : l'entrée représente l'instance à traiter, la sortie la réponse à la question pour cette instance.
Les points importants sont :

- les notions d'entrée et de sortie
- la notion d'instance d'un problème auquel répond l'algorithme
- le fait qu'un algorithme réponde à une question pour toutes les instances du problème

L'aspect effectivité

Un algorithme apporte une solution effective. C'est une méthode systématique qu'un opérateur peut mettre en œuvre. Sur des données finies, en suivant des règles non-ambigües, un algorithme s'exécute en un nombre fini d'étapes. L'opérateur peut être de toute sorte mais bien souvent il s'agit d'un ordinateur pour lequel l'algorithme a été exprimé sous forme d'un programme dans un langage extrêmement précis.
La notion d'algorithme englobe d'autres notions pouvant être constructives et finies : formules mathématiques, programmes de calcul, constructions géométriques, activité de la vie courante...
Les points importants sont :

- un algorithme peut être mis en œuvre par un opérateur quelconque,
- un algorithme est exécutable par un ordinateur/une machine,
- un algorithme peut être exprimé sous forme d'un programme,
- un algorithme agit sur des données finies,
- un algorithme s'exécute en un nombre fini d'étapes,
- la non-ambiguïté de la description des étapes,
- un algorithme permet de décrire des formules, des programmes de calcul ou de construction.

L'aspect preuve

Comme nous l'avons vu dans les exemples, preuve et algorithme interagissent de manière importante. Tout d'abord, pour présenter un algorithme et affirmer qu'il résout un problème donné, il convient d'en donner une démonstration. Il faut être certain qu'il aboutit

au bon résultat quelle que soit l'instance, c'est ce que l'on appelle la *correction*, et pouvoir garantir que ce résultat sera atteint en un nombre fini d'étapes, c'est la *terminaison*. C'est ce que l'on a fait pour prouver l'algorithme d'Euclide au paragraphe 2.1.
Ces preuves d'algorithme s'appuient souvent sur la preuve d'un invariant ou d'un variant de l'algorithme qui garantissent la correction et la terminaison.
De plus, un algorithme peut être utilisé au cours d'une preuve en tant qu'inférence : cela a été le cas lorsque l'on a utilisé l'algorithme d'Euclide pour prouver l'existence des coefficients de Bézout. La validité de l'algorithme utilisé est alors un point indispensable à celle de la démonstration concernée.
Les preuves dites constructives (la preuve par induction en est un exemple) peuvent en général donner lieu à un algorithme. Nous en avons vu une illustration pour la preuve de l'existence de cycles eulériens (paragraphe 2.2). Il peut être nécessaire d'expliciter ces algorithmes sous-jacents.
Ce type de preuves constructives concerne le plus souvent des problèmes d'existence d'objets ou de reconnaissance d'une classe d'objets.
Afin de démontrer l'existence d'un objet ayant une certaine propriété (P), les méthodes constructives exhibent de tels objets. En mathématiques, la question de la caractérisation de tous les objets vérifiant (P) s'avère une question de recherche souvent difficile. Un moyen de répondre à cette question est l'énumération de tous ces objets : cette technique peut aussi faire appel à un algorithme. Cependant, le problème d'énumération est parfois encore trop difficile. De plus, l'algorithmique permet d'aborder des questions de reconnaissance des objets vérifiant (P) : étant donné un objet O, un algorithme de reconnaissance permet de dire si O vérifie (P). Ces questions de reconnaissance sont clairement de nature mathématique et ne peuvent être abordées sans le concept d'algorithme.
Dans le cas du cycle eulérien, nous avons vu qu'il existe un algorithme de construction d'un cycle ainsi qu'un algorithme de reconnaissance de la classe des graphes eulériens [19].
Dans les problèmes d'optimisation, pour un algorithme fixé, la preuve peut aussi intervenir pour étudier l'optimalité des solutions produites, les instances pour lesquelles la solution est optimale ou encore la marge d'erreur entre les solutions produites et les solutions optimales.
La preuve d'algorithmes revêt aussi une importance particulière depuis l'utilisation d'algorithmes dans des preuves récentes, en tant que substitut au mathématicien pour des tâches non accessibles à l'homme (théorème des 4 couleurs ou conjecture de Kepler par exemple). Des programmes permettent donc maintenant de prouver des théorèmes ou de valider d'autres programmes, de manière plus ou moins automatique, par des méthodes de preuves formelles ou de vérification.
Les points importants sont :

- la preuve de correction,
- la preuve de terminaison,
- la notion d'invariant,
- l'utilisation d'algorithmes dans des preuves,
- la preuve d'existence,
- la preuve d'une propriété,
- les preuves algorithmiques, par récurrence, par induction,
- un algorithme avec sa preuve équivaut à une preuve constructive,
- la preuve d'optimalité de la solution,

19. Qui contiennent un cycle eulérien.

– les preuves formelles par ordinateur et la vérification de preuves et de programmes.

L'aspect complexité

La notion de complexité est essentielle et spécifique à l'algorithmique. La complexité d'un algorithme se mesure en fonction de la taille de l'instance (par exemple pour un graphe, son nombre de sommets et/ou d'arêtes). Elle représente le nombre d'opérations élémentaires effectuées par l'algorithme (il faut donc décider de ce que l'on choisit comme opération élémentaire) qu'effectue l'algorithme. Ce nombre peut être calculé pour le pire des cas (le maximum des nombres d'opérations effectuées pour les instances d'une taille fixée) ou en moyenne (le nombre moyen d'opérations effectuées pour les instances d'une taille fixée). La complexité peut aussi être étudiée en espace : il s'agit de mesurer la quantité d'information maximale ou moyenne nécessaire à l'exécution de l'algorithme pour les instances d'une taille donnée.
Cette notion de complexité peut être mesurée en temps ou en taille de mémoire nécessaire dans le cas d'algorithmes sous forme de programmes informatiques ou étudiée dans le cadre de modèles de traitement et stockage de l'information (structures de données) proches du fonctionnement des ordinateurs.
Cette notion de complexité permet de comparer des algorithmes entres eux et de rechercher l'algorithme le plus efficace pour un problème donné. Cela permet aussi de classifier les problèmes selon la complexité des algorithmes permettant de les résoudre.
Pour des problèmes dont on sait qu'il n'existe probablement pas d'algorithme efficace, on peut rechercher des heuristiques : des algorithmes à la complexité convenable apportant des solutions approchées dont on maîtrise l'erreur produite.
Les points importants sont :

– la complexité en temps, en espace,
– la complexité calculée au pire, en moyenne,
– la complexité comme critère de comparaison d'algorithmes,
– la recherche d'algorithmes optimaux,
– les heuristiques d'approximation de solution.

L'aspect modèles théoriques

Les questions de classification des algorithmes et des problèmes peuvent s'appuyer sur des modèles théoriques de ce qui est calculable. Pour cela plusieurs modèles ont été proposés tels que les machines de Turing ou les fonctions récursives de Kleene. Ces modèles peuvent être considérés comme d'autres définitions, plus théoriques, de ce qu'est un algorithme. Ils permettent de mettre en évidence que tous les problèmes ne sont pas algorithmiquement résolubles (décidabilité ou indécidabilité). Pour le cas des problèmes résolubles algorithmiquement, ces modèles théoriques permettent une classification selon leur complexité.
Les points importants sont :

– les machines de Turing, fonctions récursives et autres modèles,
– les classes de complexité (P, NP, etc...),
– les notions de décidabilité et d'indécidabilité.

3.2 Dualité outil-objet

Ces cinq aspects peuvent être articulés selon une dualité outil-objet, au sens de Douady :

> Ainsi, nous disons qu'un concept est outil lorsque nous focalisons notre intérêt sur l'usage qui en est fait pour résoudre un problème. Un même outil peut être adapté à plusieurs problèmes, plusieurs outils peuvent être adaptés à un même problème. Par objet, nous entendons l'objet culturel ayant sa place dans un édifice plus large qui est le savoir savant à un moment donné, reconnu socialement. (Douady, 1986, p. 9)

Parmi les différents aspects de l'algorithme relevés ci-dessus, certains font référence à l'outil et d'autres à l'objet. Regarder l'algorithme en tant qu'objet c'est s'intéresser aux questions de bon fonctionnement, de domaine de validité, de complexité et de description des algorithmes. Ce sont les problématiques de l'algorithmique. Regarder l'algorithme en tant qu'outil, c'est s'intéresser à l'utilisation que l'on en fait pour résoudre des problèmes. Les aspects PROBLÈME et EFFECTIVITÉ se réfèrent à l'outil, les aspects PREUVE, COMPLEXITÉ et MODÈLES THÉORIQUE se réfèrent à l'objet :

Outil	**Objet**
PROBLÈME EFFECTIVITÉ	PREUVE COMPLEXITÉ MODÈLES THÉORIQUES

Cette dualité nous semble particulièrement utile en ce qui concerne l'algorithmique : la particularité des algorithmes d'être des méthodes de résolution de problème (aspect PROBLÈME) faciles à transférer (aspect EFFECTIF) implique un risque de ne considérer l'algorithmique uniquement sur ce point de vue outil (notamment dans un contexte d'enseignement). Or l'analyse ci-dessus laisse percevoir un objet riche avec un potentiel fort d'enrichissement de l'activité mathématique (en particulier du point de vue de la preuve). Nous reviendrons plus loin sur la dualité outil-objet autour de l'algorithme.

Conclusion et nouvelles questions

Ces aspects constituent pour nous un premier modèle épistémologique pour l'algorithmique et fournissent une réponse à la question Q1 :

Question Q1. *Quels sont les éléments constitutifs spécifiques du concept d'algorithme ? Quels sont l'objet et les préoccupations fondamentales de la discipline algorithmique ? Quel lien entretiennent algorithme et preuve ?*

Cette analyse du concept sous forme d'aspects peut constituer un outil relativement souple d'analyse : en relevant les aspects présents dans un discours ou un document, on peut étudier quelle vision de l'algorithme est en jeu et si les aspect outils et objets sont présents. En particulier, cet outil peut permettre de fournir certaines réponses aux questions Q5, Q6 et Q7.
Cela nous amène à reformuler ces questions en fonction de ce premier modèle :

Question $\mathbf{Q5_{aspects}}$. *Quels aspects de l'algorithme sont présents dans les programmes de mathématiques et documents d'accompagnements du lycée ? Les aspects outils et objets sont-ils représentés ? Qu'en est-il dans les programmes de la spécialité ISN ?*

Question $\mathbf{Q6_{aspects}}$. *Quels aspects de l'algorithme sont présents dans les manuels de mathématiques ? Les aspects outils et objets sont-ils représentés ?*

Question $\mathbf{Q7_{aspects}}$. *Quels aspects de l'algorithme sont présents dans les ressources en ligne ? Les aspects outils et objets sont-ils représentés ? Y-a-t-il une différence entre les documents de formation et ceux pour la classe ?*

La question (Q4) concerne entre autres le modèle épistémologique développé pour (Q1). On peut maintenant reformuler cette question :

Question $\mathbf{Q4_{aspects}}$. *Notre modèle des aspects de l'algorithme s'accorde-il avec la perception qu'ont les chercheurs ? Le modèle peut-il nous informer sur les variations que l'on pourrait entrevoir dans les discours de chercheurs ?*

Chapitre 2

Réflexion sur la pensée algorithmique : apport d'un point de vue extérieur aux mathématiques

Sommaire

1 La « Pensée » ? 46
2 La pensée algorithmique en tant que pensée mathématique . . 46
2.1 Considérations épistémologiques 46
2.2 Perspectives pour l'enseignement 48
3 Pensée algorithmique versus pensée mathématique 48
3.1 Considérations épistémologiques 48
3.2 Perspectives pour l'enseignement 51
Conclusion et nouvelles questions 53

Nous avons présenté dans le chapitre précédent, une analyse épistémologique de l'algorithme sous forme d'ASPECTS. Cela fournit un modèle du concept-algorithme, mais cela ne permet pas de décrire l'activité algorithmique elle-même. Les exemples présentés précédemment ont montré des facettes de cette activité algorithmique qui semble partager beaucoup avec l'activité mathématique.

Cela soulève la question de l'existence d'un mode de pensée algorithmique, proche de celui des mathématiques, et de ses spécificités. Cela était l'objet de la question Q2 :

Question Q2. *Peut-on parler d'activité ou de pensée algorithmique dans les mathématiques ? Et dans l'informatique ? Quelles perspectives cela peut-il ouvrir pour une approche didactique ?*

Nous reformulons cette question en trois sous-questions auxquelles nous allons essayer de répondre par la suite.

- Q2a : Qu'est-ce que la pensée algorithmique et quelles sont ses caractéristiques ?
- Q2b : Quelles relations entretiennent la pensée mathématique et la pensée algorithmique ? La pensée algorithmique et les diverses pensées mathématiques ?
- Q2c : Quels intérêts didactiques à distinguer la pensée algorithmique de/dans la pensée mathématique ?

Nous nous focaliserons essentiellement sur la relation entre pensée mathématique et algorithmique d'un point de vue épistémologique et nous soulèverons certains aspects importants pour la mise en œuvre et l'apprentissage de ces pensées. Notre objectif étant de produire un modèle de l'activité algorithmique, nous essaierons de mettre en avant quelques critères importants pour ce futur modèle.

L'algorithmique peut être vue comme une branche des mathématiques, mais nous l'avons vu précédemment, on peut voir l'algorithmique dans un sens plus large comme l'informatique toute entière (c'est en particulier ce que propose Knuth). Nous articulerons donc notre questionnement suivant ces deux visions de la pensée algorithmique : la première comme une pensée des mathématiques, la seconde comme la pensée de la science informatique.

1 La « Pensée » ?

Commençons par préciser ce que nous entendons par pensée (mathématique ou algorithmique). Le terme provient des recherches anglo-saxonnes sur le *mathematical thinking*. Concernant le sens des termes *thinking* ou pensée nous nous référons notamment à (Tall, 1991) et (Rasmussen, Zandieh, King, & Teppo, 2005) :

> [We] take a critical look at advanced mathematical thinking as part of the living process of human thought rather than the immutable final product of logical deduction. (Tall, 1991, p. 21)

Nous faisons donc référence à la pensée (mathématique, algorithmique, etc.) comme activité de l'esprit humain et non comme son produit. D'autre part, nous utiliserons, comme le font Rasmussen et al. (2005), le terme d'activité (mathématique, algorithmique...) de manière équivalente au terme pensée, afin d'insister sur le fait que nous n'opposons pas ici la pensée à l'action, mais que nous parlons de pensée en tant qu'activité globale, à la fois intellectuelle et pratique :

> We also use the term activity, rather than thinking. This shift in language reflects our characterization of progression in mathematical thinking as acts of participation in a variety of different socially or culturally situated mathematical practices. [...] We view the relationship between doing and thinking to be reflexive in nature, not dichotomous. (Rasmussen et al., 2005, p. 51)

On retrouve aussi souvent le terme de « advanced mathematical thinking », qu'il faut comprendre au sens de la pensée (ou l'activité) du chercheur en mathématiques. Nous nous placerons par rapport à cela dans la même optique que Harel et Sowder (2005) :

> Advanced mathematical thinking, usually conceived as thinking in advanced mathematics, might profitably be viewed as advanced thinking in mathematics (advanced mathematical-thinking). (Harel & Sowder, 2005, p. 1)

Nous concevrons l'*advanced mathematical thinking* de cette façon, comme la pensée avancée en mathématique. Nous donnerons un sens équivalent à un *advanced algorithmic thinking*.

2 La pensée algorithmique en tant que pensée mathématique

2.1 Considérations épistémologiques

La pensée algorithmique, contrairement à la pensée mathématique, n'a été que très peu étudiée. Dans les travaux où on la rencontre, elle est souvent abordée comme une pensée

mathématique en particulier. C'est ce point de vue que nous allons adopter dans un premier temps.

Mingus et Grassl (1998) étudient la vision d'enseignants de mathématiques en activité et en formation concernant la pensée algorithmique, la pensée récursive et le problem-solving. Ils se placent implicitement à l'intérieur de la pensée mathématique et définissent l'*algorithmic thinking* de la manière suivante :

> Algorithmic thinking is a method of thinking and guiding thought processes that uses step-by-step procedures, requires inputs and produces outputs, requires decisions about the quality and appropriateness of information coming and information going out, and monitors the thought processes as a means of controlling and directing the thinking process. In essence, algorithmic thinking is simultaneously a method of thinking and a means for thinking about one's thinking. (Mingus & Grassl, 1998, p. 34)

Les auteurs donnent comme modèle (au sens de modélisation) de la pensée algorithmique, l'heuristique de résolution de problèmes proposée par Pólya (1945). Ceci pose la question de la différence entre *problem-solving* et pensée algorithmique, ainsi que la différence avec l'activité mathématique en général (l'heuristique de Pólya concernant la résolution de n'importe quel problème mathématique). Il nous semble que les auteurs font plutôt référence ici à ce que l'on pourrait traduire par une pensée « algorithmisée », c'est-à-dire un effort pour forcer sa pensée, en mathématique par exemple, à suivre un cheminement établi, systématique, avec un contrôle sur son produit.

Nous nous distinguons d'une telle approche. Nous nous intéressons ici, non pas à rendre la pensée algorithmique (à l'"algorithmiser"), mais bien à l'étude de la pensée (ou l'activité) en algorithmique [1]. Dans cette optique, la définition précédente nous semble trop ambigüe et non effective pour notre questionnement.

Hart (1998) parle, quant à lui, d'*algorithm problem solving*, un thème central dans toutes les branches des mathématiques discrètes, qu'il définit comme l'étude de la question : une solution peut-elle être construite de manière effective ? On trouve ici encore le problem-solving et l'algorithmique mêlés. La pensée algorithmique serait alors une façon d'aborder un problème en essayant de systématiser sa résolution, de se questionner sur la façon dont des algorithmes pourraient ou non le résoudre. On peut rapprocher cette définition de la vision constructiviste des mathématiques.

Notre vision de la pensée algorithmique se rapproche de celle de Hart (1998). L'activité mathématique étant centrée sur la résolution de problèmes, la pensée algorithmique, en tant que pensée mathématique parmi d'autres, serait une approche particulière des problèmes mathématiques (de certains en tout cas).

Ces problèmes sont généralement issus d'une branche particulière des mathématiques, les mathématiques discrètes ou bien d'une discrétisation d'objets du continu (problèmes d'approximation en analyse, etc.).

1. De même que les pensées algébrique ou statistique ne sont pas la conduite de sa pensée de manière algébrique ou statistique mais les modes de pensée en jeu dans les branches concernées.

2.2 Perspectives pour l'enseignement

Rasmussen et al. (2005) s'intéressent à l'*advanced mathematical thinking* dans une perspective éducative. Dans une description de ce qu'ils nomment *advancing mathematical activity*, ils se focalisent sur trois grandes pratiques dans l'activité mathématique :

> Students' symbolizing, algorithmatizing, and defining activities are three examples of such social or cultural practices. These three mathematical practices are not meant to be exhaustive, but represent a useful set of core practices that cut across all mathematical domains. Another significant mathematical practice, one that we leave to later analysis, is justifying. (Rasmussen et al., 2005, p. 52)

Attardons-nous sur ce qu'ils nomment l'*algorithmatizing*. Bien qu'ils utilisent le terme algorithme dans un sens plus large que le notre, « as a reference for a generalized procedure that is effective across a wide range of tasks » (Rasmussen et al., 2005, p. 63), leur approche nous paraît enrichissante :

> Keeping this activity perspective of algorithms in the forefront suggests that instead of focusing on the acquisition of these algorithms, we can characterize learning to use and understand algorithms as participating in the practice of algorithmatizing. By examining the activity that leads to the creation and use of artifacts, as opposed to the acquisition of the artifacts, we view mathematical learning of and in reference to algorithms through a different lens. (Rasmussen et al., 2005, p. 63)

L'activité algorithmique ne se résume donc pas à l'apprentissage et la mise en œuvre d'algorithmes mais englobe aussi leur production, leur compréhension et leur étude. Rasmussen et al. (2005) montrent ensuite qu'il est possible et pertinent de mettre des étudiants en situation d'*algorithmitazing*.

En mathématiques, l'activité algorithmique peut se définir comme une partie de l'activité en jeu dans la résolution d'une certaine catégorie de problèmes dans laquelle on recherche un procédé systématique et effectif de résolution. Il est possible d'aborder l'objet algorithme et la pensée algorithmique en classe dans le cadre d'une réelle activité mathématique (résolution de problèmes, preuve de solutions, etc).

Cependant, il est peut-être restrictif de voir la pensée algorithmique comme une pensée des mathématiques et il nous semble que l'étude de la pensée algorithmique comme la pensée en jeu dans la science informatique (l'"algorithmics" de Knuth) peut enrichir notre approche.

3 Pensée algorithmique versus pensée mathématique

3.1 Considérations épistémologiques

Donald Knuth, mathématicien de formation, devenu l'un des pionniers de l'informatique et auteur des ouvrages de référence *The Art of Computer Programming*, s'est beaucoup intéressé aux liens qu'entretiennent mathématiques et informatique. Dans l'article *Algorithmic thinking and mathematical thinking* (Knuth, 1985), il aborde la question du rôle de l'algorithmique dans les sciences mathématiques et se demande ce qui différencie *algorithmic thinking* et *mathematical thinking*[2]. Il formule « Do most mathematicians have

2. Comme précisé précédemment, pour Knuth, *algorithmics* prend le sens d'informatique (*computer science*)

	Formula manipulation	Representation of reality	Behavior of function values	Reduction to simpler problems	Dealing with infinity	Generalization	Abstract reasoning	Information structures	Algorithms
1 (Thomas)	**	**	**						
2 (Lavrent'ev/Nikol'skiĭ)	**		*		**				
3 (Kelley)	*					**	**		
4 (Euler)	**		**	*		**			*
5 (Zariski/Samuel)	*			*	**	*	**	**	
6 (Kleene)	*					**	**		*
7 (Knuth)	**	*		*					
8 (Pólya/Szegő)	**		**	**	**				
9 (Bishop)	**		**	**		*	**	**	*
"Algorithmic thinking"	*	**		**			**	**	**

FIGURE 2.1 – Les différentes pensées mathématiques selon Knuth et leur présence dans divers ouvrages mathématiques.

an essentially different thinking process from that of most computer scientists ? » (Knuth, 1985).

Knuth s'intéresse donc à la pensée algorithmique comme distincte des mathématiques. Il s'appuie sur des ouvrages de mathématiques et sur des preuves qui y sont présentées pour analyser la pensée mathématique et la comparer à la pensée algorithmique. Il divise la pensée mathématique en neuf « modes de pensée » qu'il résume dans le tableau de la figure 2.1 :

- la manipulation de formules,
- la représentation d'une réalité,
- le comportement des valeurs de fonctions,
- la réduction à des problèmes plus simples,
- la manipulation de l'infini,
- la généralisation,
- le raisonnement abstrait,
- les structures d'information,
- les algorithmes.

Précisons que Knuth ne considère pas ce découpage comme parfait ou immuable : « Thus, I am not all certain of the categories ; they are simply put forward as a basis for discussion » Knuth (1985, p. 180)

En s'appuyant sur ce découpage, il analyse ce qu'il considère comme la pensée algorithmique. Selon lui elle englobe, parmi les manières de raisonner précédentes, la représentation d'une réalité, la réduction à des problèmes plus simples, le raisonnement abstrait, les structures d'information, les algorithmes et, dans une moindre mesure, la manipulation de formules.

On pourrait reprocher à Knuth de mélanger la pensée (algorithmique ou mathématique) et les questions auxquelles cette pensée s'applique ou les objets dont elle traite. Cependant, son analyse peut enrichir notre questionnement : n'est-ce pas davantage le type de

questionnement et les objets étudiés qui distinguent mathématique et algorithmique ?

Wilf (1982) semble aller dans ce sens : dans un article intitulé *What is an Answer*, il analyse ce qui est une réponse acceptable dans un problème de dénombrement et montre notamment qu'une formule de dénombrement peut être parfois plus complexe [3] à évaluer que l'ensemble étudié à énumérer. Dans un tel cas, peut-on considérer la réponse produite comme une réponse acceptable du point de vue mathématique ? Du point de vue algorithmique ?

De tels exemples ne sont pas uniquement liés aux mathématiques discrètes. Par exemple, une formule exacte donnant la valeur d'une intégrale est-elle toujours plus simple à évaluer que l'intégrale elle-même ? Un tel exemple est développé dans Maurer (1998).

Il semblerait que ce qui caractérise la pensée algorithmique soit plutôt une préoccupation quant à l'effectivité des résultats recherchés qu'une réelle différence dans l'activité.

Cependant, Knuth (1985) note aussi que deux aspects fondamentaux de la pensée algorithmique ne sont pas présents dans sa liste des modes de pensée mathématique : la notion de complexité et l'opération d'affectation qu'il symbolise par « := » (et que nous avons aussi notée ← précédemment).

Selon lui, ces deux aspects pourraient être ce qui différencie les pensées mathématiques et algorithmiques.

La notion de **complexité** est relative à l'efficacité d'un algorithme ou d'une procédure effective. Elle englobe toutes les questions se rapportant au nombre d'étapes de calcul, au temps, ou à l'espace mémoire, nécessaires à l'exécution d'un algorithme, dans le pire des cas ou en moyenne. La notion de complexité est clairement liée à des problématiques de mathématiques constructives ou effectives, cependant elle précise ces notions et permet une hiérarchisation des algorithmes selon certains critères d'effectivité réelle, ou d'efficacité.

La notion d'**affectation** (ou assignation) évoque ici le concept de variable informatique, par opposition à variable mathématique. Cette notion d'affectation (de même que le concept de variable informatique) est en lien étroit avec la notion de complexité. En effet, c'est l'utilisation de variables informatiques qui permet la production d'algorithmes efficaces en terme de complexité en espace.

L'absence de ces deux aspects dans la pensée mathématique peut se résumer par les remarques de Knuth concernant Bishop [4] :

> Bishop's mathematics is constructive, but it does not have all the ingredients of an algorithm because it ignores the "cost" of the constructions. [...] The assignement operation in Bishop's constructions aren't really assignements, they are simply definitions of quantities, and those definitions won't be changed. (Knuth, 1985, p. 181)

En d'autres termes, la pensée des mathématiques constructives diffère de la pensée algorithmique par le fait qu'elle ne questionne pas l'efficacité des algorithmes qu'elle produit,

3. au sens de la complexité algorithmique

4. Bishop est un mathématicien américain, défenseur du constructivisme en mathématiques et fondateur de l'analyse constructive.

qu'elle ne cherche pas toujours à décrire avec précision ces algorithmes et qu'elle n'utilise pas la notion de variable informatique.

Dans un autre article traitant des mêmes questions, Knuth (1974) aborde les interactions entre informatique et mathématiques et évoque les diverses façons dont l'informatique influence les mathématiques. Il note que l'informatique permet d'effectuer certains calculs inaccessibles et de valider ou d'invalider certaines conjectures, que l'informatique a fait mettre l'accent sur les constructions en mathématiques (notamment par le remplacement de preuves constructives par des algorithmes), que l'informatique permet la construction de certaines bijections et que l'informatique a complètement étendu et enrichi le champ des mathématiques discrètes. Cependant, l'impact le plus notable selon Knuth (1974) est que l'informatique apporte de nouveaux problèmes.

> The most significant thing is that the study of algorithms themselves has opened up a fertile vein of interesting new mathematical problems ; it provides a breath of life for many areas of mathematics that had been suffering from a lack of new ideas. (Knuth, 1974, p. 328)

Le mathématicien Herbert S. Wilf, dans l'article *Mathematics : An Experimental Science* (2005), évoque lui aussi les changements apportés par l'informatique aux mathématiques. Il se concentre principalement sur les interactions possibles lors des phases d'expérimentation et de conjecture dans l'activité mathématique. La pensée mathématique se trouve donc changée par l'informatique, et son aspect expérimental s'en trouve renforcé :

> A mathematician can act in concert with a computer to explore a world within mathematics. From such explorations there can grow understanding, and conjectures, and roads to proofs, and phenomena that would not have been imaginable in the pre-computer era. (Wilf, 2005, p. 999)

Finalement, l'informatique est source de questions d'ordre mathématique mais aussi de questions sur l'essence des mathématiques. Ces questionnements doivent aussi atteindre l'enseignement des mathématiques, dans ses pratiques mais aussi dans ses contenus.

> The most surprising thing to me, in my own experiences with applications of mathematics to computer science, has been the fact that so much of the mathematics has been of a particular discrete type. [...] Such mathematics was almost entirely absent from my own training, although I had a reasonably good undergraduate and graduate education mathematics. Nearly all of my encounters with such techniques during my student days occurred when working problems from the American Mathematical Monthly. I have naturally been wondering whether or not the traditional curriculum – the calculus courses, etc. – should be revised in order to include more of these discrete mathematical manipulations, or whether computer science is exceptional in its frequent application of them. (Knuth, 1974, p. 329)

3.2 Perspectives pour l'enseignement

Une vision de la pensée algorithmique, non seulement comme *une* pensée mathématique mais aussi comme *la* pensée informatique semble élargir les perspectives pour son enseignement.

Les aspects que Knuth présente comme absents de la pensée mathématique – description effective d'algorithmes et étude de leur complexité – nous paraissent être propices

à une véritable mise en œuvre de la pensée mathématique. De plus, ces aspects sont de plus en plus pris en compte par les mathématiciens qui travaillent avec des algorithmes, et la frontière entre pensée algorithmique et pensée mathématique n'en devient que plus floue.

Selon le mathématicien Lovász (1988, 2007), l'activité mathématique a subi des changements importants durant les cinquante dernières années et le développement de l'informatique en est la source la plus importante. Il résume son questionnement ainsi :

> Is algorithmic mathematics of higher value than classical, structure-oriented, theorem-proof mathematics, or does it just hide the essence of things by making them more complicated then necessary ? Does teaching of an algorithm lead to a better understanding of the underlying structure, or is it a more abstract, more elegant setting that does so ? (Lovász, 1988, p. 1)

Il insiste sur le fait que l'algorithmique peut apporter un regard nouveau sur des problèmes anciens et se base sur divers exemples. Il s'interroge alors sur les implications de tels changements sur l'enseignement des mathématiques, et propose des pistes pour faire entrer l'algorithmique dans l'enseignement des mathématiques. Lovász insiste sur le fait que la création d'algorithmes devrait être abordée avant l'apprentissage de leur exécution. Dans un deuxième temps, il considère que l'étude d'algorithmes et de leur analyse devrait être abordée au même titre que celle de théorèmes et de leur preuve. Il met aussi en garde contre une utilisation trop rapide de l'ordinateur dans l'étude des algorithmes.

> The route from the mathematical idea of an algorithm to a computer program is long. It takes the careful design of the algorithm ; analysis and improvements of running time and space requirements ; selection of (sometimes mathematically very involved) data structures ; and programming. In college, to follow this route is very instructive for the students. But even in secondary school mathematics, at least the mathematics and implementation of an algorithm should be distinguished. (Lovász, 1988, p. 10)

D'autre part, il insiste sur le fait que l'activité mathématique est en plein changement pour plusieurs autres raisons et insiste sur le fait que l'enseignement des mathématiques doit suivre cette évolution :

> In addition, as we will see, we should put more emphasis on (which also means giving more teaching time to) some non-traditional mathematical activities like algorithm design, modeling, experimentation and exposition. I also have to emphasize the necessity of preserving problem solving as a major feature of teaching mathematics. (Lovász, 2007, p. 3)

Il semble donc important que la pensée algorithmique prenne une place dans l'enseignement des mathématiques. Certaines branches des mathématiques paraissent plus favorables que d'autres au développement de cette pensée. Les mathématiques discrètes, qui se situent à la frontière entre mathématiques et informatique, peuvent être un champ fertile pour développer la pensée algorithmique et sont un domaine favorable à l'expérimentation, à la modélisation, au *problem-solving* et à la preuve (Ouvrier-Buffet, 2009).

Des expérimentations ont été menées en classe avec des résultats convaincants, concernant l'algorithmique en théorie des graphes, optimisation combinatoire ou dans d'autres branches des mathématiques discrètes (Schuster, 2004 ; Hart, 1998).

Conclusion et nouvelles questions

La pensée algorithmique peut être vue comme faisant partie de la pensée mathématique et nous avons évoqué certaines de ses caractéristiques. Cependant, voir la pensée algorithmique comme pensée majeure de l'informatique permet un réel enrichissement de son analyse. Nous avons montré en quoi cette pensée algorithmique influe sur la pensée mathématique. Il nous semble donc que, pour aborder la pensée algorithmique, il soit indispensable d'appréhender simultanément les deux points de vue, intra-mathématique mais aussi extra-mathématique. Cela constitue une première réponse à Q2.

Pour poursuivre le travail, une étude détaillée des différences et similarités entre pensée mathématique et algorithmique reste à développer et nous l'avons vu, ce sont davantage les objets et les questions en jeu que l'activité elle-même qui diffère. Notamment, il nous semblerait approprié d'étudier en quoi les modèles pour la pensée mathématique et son enseignement (par exemple ceux de Rasmussen et al. (2005) ou de Dreyfus (1991)) peuvent se différencier de modèles pour la pensée algorithmique.

Pour autant, les conclusions que nous apportons ici fournissent déjà des repères importants pour construire un modèle de l'activité algorithmique C'est ce que nous proposerons dans le chapitre suivant en abordant la question Q3. Le modèle que nous proposerons peut donner aussi un certain éclairage pour poursuivre l'étude de la pensée algorithmique.

Chapitre 3

Un modèle de conceptions pour l'algorithme

Sommaire

1 La notion de conception et le modèle cK¢ 55
2 Problèmes et problèmes d'algorithmique 57
2.1 Une notion de problème adaptée 57
2.2 Différentes familles de problèmes 58
2.3 Dialectique outil-objet 59
3 Trois fois deux conceptions pour l'algorithme 60
3.1 Trois paradigmes pour l'algorithmique 60
3.2 Six conceptions pour l'algorithme 62
3.3 Relation entre ces conceptions 64
3.4 Relation aux ASPECTS 66
Conclusion et nouvelles questions 66

Nous avons déjà détaillé de nombreux points de l'épistémologie de l'algorithme et de l'algorithmique dans les sections précédentes. En particulier, le chapitre 2 a mis en avant des éléments importants pour comprendre et décrire l'activité algorithmique. Nous voulons maintenant proposer un modèle de cette activité algorithmique. Cela était l'objet de la question Q3 :

Question Q3. *Peut-on décrire un modèle de l'activité algorithmique permettant de questionner la transposition didactique ?*

Nous présentons maintenant un tel modèle, qui s'appuie sur la notion de *conception* du modèle cK¢, développé par Balacheff et qui complète les analyses du concept algorithme du chapitre 1.

1 La notion de conception et le modèle cK¢

La notion de *conception*, définie par Balacheff, dérive de la notion de *concept* définie par (Vergnaud, 1990).

Vergnaud définit un concept C comme un triplet $(S, I, \mathbf{S})$ où :

– S est l'ensemble des situations qui donnent du sens au concept,

– I est l'ensemble des invariants opératoires,

– **S** est l'ensemble des représentations du concept.

Une conception est, pour Vergnaud, l'équivalent du concept mais du côté de l'individu.

Balacheff propose de décrire les conceptions en précisant deux niveaux d'invariants : les *opérateurs* qui permettent d'agir sur la situation et les structures de contrôle qui justifient et valident l'utilisation des opérateurs.
Une conception C est alors décrite par un quadruplet (P, R, L, Σ) où :

– P est un ensemble de problèmes sur lesquels C est opératoire,

– R est un ensemble d'opérateurs,

– L est un système de représentation qui permet d'exprimer les éléments de P et de R,

– Σ est une structure de contrôle qui assure la non contradiction de C. Un problème p de P sera résolu s'il existe r de R et s de Σ tel que $s(r(p))$ est vrai.

> Nous appelons *opérateur* ce qui permet la transformation des problèmes ; ces opérateurs sont attestés par des productions et des comportements.
> Un *système de représentation* (langagier ou non) permet l'expression des problèmes et des opérateurs.
> Enfin, une *structure de contrôle* assure la non contradiction de la conception et contient au moins sous la forme d'oracles les outils de décision sur la légitimité de l'emploi d'un opérateur ou sur l'état (résolu ou non) d'un problème. (Balacheff & Margolinas, 2005, p. 80)

La notion de conception de Balacheff, par rapport au concept de Vergnaud, nous permettra de mettre en avant une dualité outil-objet à l'aide de la relation entre opérateurs et structures de contrôle. Nous verrons aussi qu'en choisissant la notion de problème utilisée par Giroud (2011), on peut préciser cette dualité.

Dans notre cas, il s'agit de définir les grandes conceptions autour du concept algorithme. Nous ne chercherons donc pas à décrire en détail les ensembles P et R, comme cela peut être fait dans le cas de conceptions plus précises et spécifiques (comme celles, par exemple, liées au concept de *tangente à une courbe* de *fonction* ou encore de *cercle*).
Nous nous limiterons plutôt à une caractérisation de ces ensembles de problèmes et d'opérateurs, dans le but de montrer en quoi les conceptions que nous présentons nécessitent d'être distinguées.

L'objectif est de caractériser des conceptions pour une étude épistémologique de l'algorithme. Nous nous situons donc ici du côté du *savoir-savant*. Cela correspond aux μ-conceptions décrites par Balacheff et Margolinas (2005) :

> On appellera C_μ une conception rendant compte du texte du savoir mathématique.
> Contrairement à la conception générale dans l'absolu, C_μ est plus à portée. La conception peut être dérivée du corpus des savoirs consensuels (académiques) en mathématiques dont on sait qu'ils le sont du point de vue des systèmes de représentation qu'ils manipulent et des structures de contrôle (i.e. la démonstration).
> On appellera μ-objet (ou objet mathématique) la classe d'équivalence de conceptions C_μ. (Balacheff & Margolinas, 2005, p. 98)

Nous pouvons reformuler notre objectif : décrire le μ-objet *algorithme*. Comme précisé précédemment, il s'agira pour nous d'étudier le corpus du savoir en mathématique et en informatique.

2 Problèmes et problèmes d'algorithmique

2.1 Une notion de problème adaptée

Balacheff et Margolinas (2005) définissent la notion de problème comme suit :

> Nous appelons *problèmes* les perturbations du système. Le domaine de la validité de la conception, ou sphère de pratique, est constitué de l'ensemble des problèmes que la conception permet de résoudre et qui ne conduisent pas à une rupture de l'équilibre du [sujet<>milieu]. (Balacheff & Margolinas, 2005, p. 80)

Cette définition correspond à un besoin de décrire les conceptions d'un sujet dans une situation particulière. Elle n'a pas de sens lorsqu'il s'agit de décrire les μ-conceptions. Cela ne permet pas de rendre compte des problèmes identifiés comme tels par le savoir savant. Il est nécessaire ici de proposer une autre définition du terme problème, adaptée à la description de μ-conceptions (en mathématiques ou en mathématique-informatique en tout cas). Pour cela, nous nous appuierons sur la notion de problème telle qu'elle est définie dans la théorie de la complexité algorithmique. Une définition de ce type a notamment été utilisée par Giroud pour étudier la notion de *concept-problème* dans sa thèse (Giroud, 2011). Un problème, dans la théorie de la complexité algorithmique, peut être décrit ainsi :

> For our purposes, a *problem* will be a general question to be answered, usually possessing several *parameters*, or free variables, whose values are left unspecified. A problem is described by giving : (1) a general description of all its parameters, and (2) a statement of what properties the answer, or *solution*, is required to satisfy. An *instance* of a problem is obtained by specifying particular values for all the problems parameters. (Garey & Johnson, 1979, p. 4)

Pour la suite, nous considérerons donc qu'un problème p est un couple (I, Q), où I est l'ensemble des instances (pouvant être décrit par plusieurs paramètres) du problème et Q une question portant sur ces instances (spécifiant les propriétés de la solution attendue). Lorsque l'on fixe une instance $i \in I$ du problème, nous parlerons d'instanciation du problème. Lorsque l'on réduit l'ensemble de définition des instances $I' \subset I$, (I', Q) on parlera de sous-problème de (I, Q).
Notons qu'au sens de Turing, tout problème peut être ramené à un *problème de reconnaissance*, c'est-à-dire, à un problème de test de l'appartenance de l'instance à un ensemble donné, autrement dit un problème dont la question attend une réponse de type *oui* ou *non*. Il est clair que dans un but didactique, réduire tout problème à une question de reconnaissance est trop restrictif, c'est pourquoi nous faisons le choix de permettre tout type de question pour un problème (du moment qu'elle définit clairement les caractéristiques de la réponse attendue).

Exemple. Avec cette définition de problème, on peut décrire les problèmes suivants :

- p_{CE}, le problème de l'existence d'un cycle Eulérien :
  I_{CE} = l'ensemble des graphes finis
  Q_{CE} =“Contient-il un cycle eulérien ?”

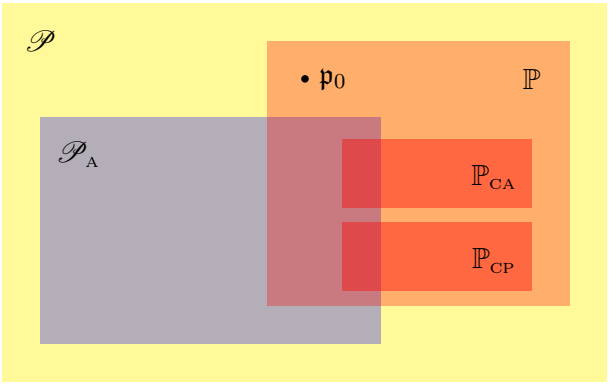
- p_{Tri}, le problème du tri :
  I_{Tri} = l'ensemble des listes L d'entiers
  Q_{Tri} =“Quelle est la liste ordonnée des éléments de L ?”

Pour résoudre un problème $p = (I, Q)$, un algorithme peut être donné. Un tel algorithme doit pouvoir recevoir en entrée n'importe quelle instance $i \in I$ du problème p et répondre à la question Q posée. Il est clair qu'un enjeu de preuve est alors présent : l'algorithme proposé répond-il toujours à la question et donne-t-il une réponse valide ?

Une autre façon de répondre à un problème p est d'énoncer un théorème qui permet de répondre de manière générique à p. Dans ce cas aussi, c'est la preuve qui garantit la validité de la réponse proposée. Cette preuve peut être (mais ce n'est pas toujours le cas) algorithmique : elle contient une description constructive de la réponse à toute instance de p qui se ferait de manière finie. Dans une preuve constructive, on peut dire que l'algorithme implicite et sa preuve sont présentés simultanément.
Lorsqu'il existe un algorithme (ou un théorème et une preuve algorithmique) résolvant un problème, on dira qu'il est algorithmiquement résoluble.

2.2 Différentes familles de problèmes

Nous noterons $\mathscr{P}$ l'ensemble des problèmes tels que nous les avons définis précédemment. L'ensemble des problèmes pouvant être résolus algorithmiquement sera noté $\mathscr{P}_{\text{A}}$.
Nous définissons aussi le sous-ensemble $\mathbb{P} \subset \mathscr{P}$ des problèmes d'algorithmique. Ce ne sont pas les problèmes pour lesquels l'algorithme intervient en tant que moyen de résolution (même si cela peut être le cas) mais ce sont les problèmes qui se placent dans le domaine de l'algorithmique. Autrement dit, $\mathbb{P}$ contient les problèmes dont la question porte sur un algorithme (i.e. I contient un ensemble d'algorithmes) ou dont la question spécifie que la réponse doit être de type algorithmique ou avoir un lien avec des notions d'algorithmique (de complexité par exemple).
Le schéma suivant résume ces ensembles :



Dans l'ensemble $\mathbb{P}$, on peut distinguer les éléments et sous-ensembles suivants :

- le problème $\mathfrak{p}_0$ qui consiste à savoir si un problème p appartient à $\mathscr{P}_{\text{A}}$
  autrement dit : $\mathfrak{p}_0 = (I, Q)$ avec
  $I = \mathscr{P}$ et $Q =$ *“Est-il résoluble algorithmiquement ?”*
- la famille $\mathbb{P}_{\text{CA}}$ des problèmes prenant pour instance un algorithme `a` et dont la question porte sur la complexité de `a`.
- la famille $\mathbb{P}_{\text{CP}}$ des problèmes prenant pour instance un problème p et dont la question porte sur la complexité de la résolution algorithmique de p.

Exemples :

- p_{CE}, p_{Tri} sont des problèmes de $\mathscr{P}_{\text{A}}$.
- On peut instancier $\mathfrak{p}_0$ avec le problème p_{CE}, cela revient à se poser la question : "p_{CE} est-il résoluble algorithmiquement ?".
- $p_{Cmoyenne} = (\{\textit{algorithmes en Caml}\}, \text{“}\textit{quelle est sa complexité en moyenne ?}\text{”})$ est un problème de $\mathbb{P}_{\text{CA}}$. On pourrait par exemple l'instancier sur l'algorithme 2.
- $p_{poly} = (\mathscr{P}, \textit{“Est-il polynomial ?”})$ est un problème de la famille $\mathbb{P}_{\text{CP}}$. On peut l'instancier sur p_{CE} ou p_{Tri} lorsque l'on se demande s'il existe un algorithme polynomial pour les résoudre.

2.3 Dialectique outil-objet

L'algorithme est un outil de résolution de problèmes. C'est aussi un objet d'étude des mathématiques. Nous voulons ici clarifier ces deux aspects en nous appuyant sur la dualité outil-objet introduite par Douady (1986) et déjà citée précédemment.

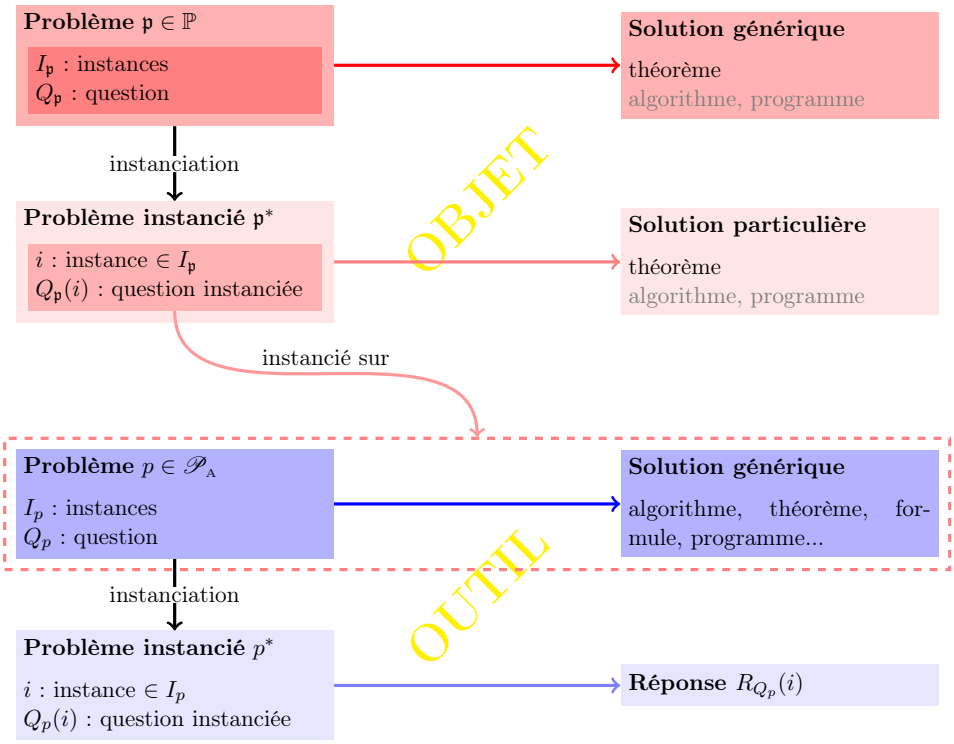
> Ainsi nous disons qu'un concept est *outil* lorsque nous focalisons notre intérêt sur l'usage qui en est fait pour résoudre un problème. Un même outil peut être adapté à plusieurs problèmes, plusieurs outils peuvent être adaptés à un même problème. Par *objet*, nous entendons l'objet culturel ayant sa place dans un édifice plus large qui est le savoir savant à un moment donné, reconnu socialement. (Douady, 1986, p. 9)

Cette dualité outil-objet est particulièrement importante concernant le concept d'algorithme. C'est seulement l'aspect objet de l'algorithme qui le différencie d'autres processus de résolution de problème. Le concept d'algorithme n'a de sens que si l'on prend du recul sur la résolution générique d'un problème par un algorithme, via sa validation, son analyse, son questionnement.

Cette dualité s'articule avec la notion de problème et de problème instancié tels que nous les avons définis et avec les différentes familles de problèmes présentées dans les paragraphes précédents. On peut aussi la mettre en lien avec les notions d'opérateurs et de systèmes de contrôle de cK¢.

Notre point de vue sur la dualité outil-objet de l'algorithme est le suivant. On dira que le concept algorithme est objet, lorsque l'on s'intéresse à des problèmes où l'algorithme apparaît dans l'instance ou dans la question. Que le problème soit instancié ou générique, on considérera que c'est l'aspect objet qui est en jeu. Ainsi, ce sont les problèmes de $\mathbb{P}$ qui mettent en jeu l'algorithme comme objet.

Nous considérerons que le concept algorithme est outil, lorsqu'il est utilisé pour la résolution d'un problème ou d'une instance d'un problème. C'est-à-dire que le concept est outil lorsqu'il est utilisé dans la résolution de problèmes de $\mathscr{P}_{\text{A}}$. Notons cependant que c'est seulement dans l'utilisation d'un algorithme `A` pour résoudre le problème ou une instance du problème que l'algorithme `A` est outil. Dès lors que sont questionnées sa validité, sa construction ou ses propriétés, on s'intéresse en réalité à un problème de $\mathbb{P}$ instancié pour l'algorithme `A`.



En termes de conceptions, on peut considérer qu'il y a un glissement d'outil vers objet lorsque les structures de contrôles deviennent opérateurs.

> Le modèle cK¢ propose une distinction explicite entre des opérateurs, qui permettent l'action, et des contrôles qui jugent de la pertinence et de la validité de l'action — une méta-action. Une des difficultés de ce modèle est de caractériser clairement cette distinction[...]. Le caractère implicite des contrôles posera essentiellement un problème d'observables. En revanche on voit souvent, dans les travaux s'appuyant sur cK¢, ressurgir la gêne que suscite cette perméabilité. [...] La perméabilité peut en fait s'interpréter de deux manières.
> La première consiste en l'évolution du statut des objets considérés, qu'on pourrait rapprocher de leur existence sous forme d'outil ou d'objet (Douady, 1986). La perméabilité reflète alors une multiplicité des points de vue possibles de l'observateur, et nous retiendrons qu'un même objet de connaissance pourra donner lieu à différents opérateurs et contrôles. (Mithalal, 2010, p. 106)

Le problème des observables et de la perméabilité opérateur-contrôle est moins gênant pour décrire des μ-conceptions, étant donné que l'on reste, à ce niveau, dans des considérations théoriques. Cependant le lien à la dialectique outil-objet nous pousse à tenir compte de cette perméabilité. Cela peut se faire en "doublant" chaque conception en une conception du côté outil et une autre du côté objet dans laquelle les contrôles de la conception outil deviennent opérateurs.

3 Trois fois deux conceptions pour l'algorithme

3.1 Trois paradigmes pour l'algorithmique

Nous distinguons tout d'abord trois cadres dans lesquels vont être décrites les conceptions. Nous les appelons des *paradigmes*. Ils sont la *preuve algorithmique*, l'*algorithme mathématique* et l'*algorithme informatique*. Ces paradigmes correspondent en quelque sorte à des

“mode de vie” de l’algorithme dans l’activité algorithmique.

Le paradigme *Preuve algorithmique* (PA) correspond à une activité de la forme Problème-Théorème-Preuve où la preuve est de type constructif et fini, donc de type algorithmique. Les preuves par récurrence entrent notamment dans ce cadre. Dans ce paradigme, l’algorithme et la preuve sont indissociables : l’algorithme n’est pas explicité directement mais est présent dans la preuve du théorème. Il peut être explicité à la suite de la preuve, comme conséquence ou corollaire de celle-ci : on se rapproche alors du paradigme AM.

Le paradigme *Algorithme mathématique* (AM) correspond à une activité de la forme Problème-Algorithme-Preuve. Pour un problème donné, on explicite un algorithme résolvant toutes les instances du problème et on fournit une preuve que l’algorithme répond pour toute instance et fournit une solution valide (terminaison et correction de l’algorithme). L’algorithme et la preuve sont ici complétement dissociés.

Le paradigme *Algorithme informatique* (AI) correspond à une activité de la forme Problème-Programme-Vérification. Le terme vérification est à prendre au sens de la vérification formelle en informatique. L’algorithme répondant au problème est exprimé sous-forme d’un programme que l’on souhaite pouvoir exécuter. La validation du programme n’est pas de l’ordre mathématique mais plutôt informatique, c’est-à-dire suivant des méthodes d’ordre formel (par exemple les analyses lexicales et syntaxiques effectuées dans la compilation ou encore la vérification automatique de certaines propriétés du programme). Cependant, la validation mathématique, dans le paradigme AM de l’algorithme sous-jacent au programme, peut fournir une forme de validation du programme dans AI.

Un même problème peut être traité dans différents paradigmes. Son étude peut même faire appel successivement à plusieurs paradigmes. Ce qui distingue ces paradigmes n’est donc pas les problèmes eux-mêmes mais leur traitement et la validation des solutions proposées. Ils se distinguent aussi par la forme sous laquelle sont exprimés les algorithmes. On entrevoit ici l’intérêt du modèle de conception pour exprimer ces différences en termes d’opérateurs, de systèmes de représentation et de structures de contrôle.

Les systèmes de représentation pour l’algorithme sont un élément important de distinction entre les trois paradigmes mais n’en constituent pas l’unique critère. Il faut cependant reconnaître l’importance de leur rôle dans l’expression des algorithmes. Des questions sémiotiques apparaissent dès lors que l’on s’intéresse aux différentes formes que peut prendre l’algorithme.

Reste à s’intéresser à la question des problèmes qui donnent du sens au concept d’algorithme. Nous l’avons déjà soulevée dans les sections précédentes. On a pu voir notamment que les distinctions, entre les problèmes de $\mathscr{P}_{\text{A}}$ et de $\mathbb{P}$ d’une part, et entre problèmes instanciés et non-instanciés d’autre part, donnent un éclairage sur la dialectique outil-objet.

Nous distinguerons donc deux conceptions dans chaque paradigme : une pour les problèmes de $\mathscr{P}_{\text{A}}$ et une pour les problèmes de $\mathbb{P}$. Nous présentons maintenant une description détaillée de chacune. Pour chaque terme en italique dans la conception, nous explicitons juste après le sens que nous lui donnons.

Terminologie Le choix du terme *paradigme* nous a semblé adapté pour décrire ces "modes de vie" de l'algorithme, qui peuvent avoir chacun leur autonomie et qui peuvent englober diverses conceptions. Ces paradigmes partagent beaucoup avec la notion de conceptions, on pourrait les voir comme des "macro"-conceptions. Mais il nous a paru plus commode pour la suite de proposer un terme qui fait référence plus nettement à cet aspect très étendu.
Le terme peut être mis en lien avec la définition des *paradigmes géométriques* de Houdement et Kuzniak. Cependant, pour la simple raison que l'épistémologie de l'algorithmique n'est pas celle de la géométrie, il ne serait pas raisonnable d'essayer de mettre en parallèle chacun des paradigmes géométriques avec les nôtres.

3.2 Six conceptions pour l'algorithme

Conceptions du paradigme PA

Conception PA-outil

P :	Problèmes de $\mathscr{P}_{\text{A}}$
R :	Les opérateurs sont la preuve algorithmique, les raisonnements "constructifs" : récurrence, induction, descentes infinies, existence de majorants/minorants d'ensembles finis...
L :	Le système de représentation est le *langage mathématique*. Les objets en jeu sont des objets mathématiques, dans cette conception on manipule de l'information. À tout moment, toute l'information de l'instance et toute celle déduite est utilisable. Les seules variables présentes sont des *variables mathématiques*.
Σ :	La structure de contrôle est celle de la *logique mathématique* et les propriétés des objets en jeu.

Par *langage mathématique* nous entendons le langage utilisé usuellement par les écrits mathématiques.
Par *variables mathématiques* nous nous référons aux différents types de variables utilisées en mathématiques.
Par *logique mathématique* nous entendons l'ensemble des règles de logique et de raisonnement utilisées implicitement dans l'activité mathématique. Ce terme est choisi par opposition à la logique formelle.

Exemples :
- Le traitement du problème du cycle eulérien par la preuve algorithmique. Les opérateurs sont les opérations constructives et la structure du raisonnement permet le contrôle de cette construction.

Conception PA-objet

P :	Problèmes de $\mathbb{P}$
R :	Les opérateurs sont la preuve mathématique et les opérations sur les modèles théoriques (réductions algorithmiques, réductions polynomiales...)
L :	Le système de représentation est le langage mathématique, et des modèles du concept d'algorithme (automates, machines de Turing, fonctions récursives...)
Σ :	La structure de contrôle est celle de la logique mathématique et les propriétés des objets en jeu.

Conceptions du paradigme AM

Conception AM-outil

P :	Problèmes de $\mathscr{P}_{\text{A}}$
R :	Les opérateurs sont des algorithmes explicités et des opérations constructives finies.
L :	Le système de représentation est un pseudo-code, mélange de langage mathématique et d'éléments de langage de programmation. Les objets manipulés sont des objets mathématiques, parfois issus de l'informatique et qui peuvent être munis d'opérations de base (types de données abstraites). Les variables sont des *variables informatiques* et des variables mathématiques.
Σ :	La structure de contrôle est la preuve de l'algorithme (correction et terminaison) pouvant utiliser des invariants.

Les *variables informatiques* sont des variables qui jouent le rôle d'un emplacement en mémoire en pouvant changer de valeur au cours du temps : lorsque l'information contenue dans la variable est modifiée, l'ancienne information est alors perdue. C'est un **modèle** de la mémoire d'une machine, issu de l'informatique, afin de mettre de côté les questions d'ordre technologique.

Exemple : Le traitement du problème du pgcd avec l'algorithme d'Euclide, le traitement du problème du tri d'une liste d'entiers avec l'algorithme du tri rapide relèvent de AM-outil.

Conception AM-objet

P :	Problèmes de $\mathbb{P}$
R :	Les opérateurs sont la preuve mathématique : preuve d'algorithme, de propriétés d'algorithme, d'invariants, de complexité...
L :	Le système de représentation est le langage mathématique.
Σ :	La structure de contrôle est celle de la logique mathématique, les règles de raisonnement et les propriétés des objets en jeu.

Exemple : L'étude de la complexité de l'algorithme de recherche de cycles eulériens (algorithme 4) effectuée précédemment relève de cette conception.

Conceptions du paradigme AI

Conception AI-outil

P :	Problèmes de $\mathscr{P}_{\text{A}}$
R :	Les opérateurs sont des programmes constitués des opérations informatiques (boucles, tests, gestions des données...).
L :	Le système de représentation est un langage de programmation. Les objets manipulés sont des données qui codent les objets en jeu à l'aide de structures de données permettant certaines opérations sur l'information.
Σ :	La structure de contrôle est de l'ordre de la vérification formelle : analyse lexicale, analyse syntaxique, vérification automatique ou preuve formelle de propriétés...

Exemple : Le calcul du pgcd par l'algorithme d'Euclide proposé sous forme de programme relève de cette conception.

Conception AI-objet

P :	Problèmes de $\mathbb{P}$
R :	Les opérateurs sont ceux de la vérification formelle.
L :	Le système de représentation est le langage mathématique et celui de la logique formelle.
Σ :	La structure de contrôle est celle de la logique formelle.

3.3 Relation entre ces conceptions

Continuité outil-objet Revenons tout d'abord sur le lien entre la conception-outil et la conception-objet d'un même paradigme. Leur relation a déjà été décrite dans la section 2.3. Nous souhaitons préciser que la frontière entre ces conceptions n'est pas aussi nette que notre découpage pourrait le laisser penser. Il n'y a pas rupture entre les deux conceptions mais plutôt une continuité qui se fait suivant la dialectique outil-objet, ou plutôt un mouvement de l'outil vers l'objet. Ce mouvement est accompagné, comme nous l'avons montré, par le passage du problème instancié au problème générique. Cependant il peut exister en pratique plusieurs problèmes intermédiaires à mesure que l'ensemble d'instances du problème étudié s'agrandit. Il faut remarquer aussi que la structure de contrôle de la conception-outil joue un rôle particulier dans ce glissement de l'outil vers l'objet : plus elle prend une place importante dans l'activité plus on s'approche de la conception-objet. Le problème $\mathfrak{p}_0$ joue aussi un rôle central dans cette articulation outil-objet.
D'autre part, il faut noter que lorsque nous parlons de mouvement ou de glissement de l'outil vers l'objet, il ne s'agit pas de dire que l'activité algorithmique est linéaire ou à sens unique. Il y a clairement des allers-retours permanents, comme dans l'activité mathématique, entre outil et objet. Simplement, il nous semble que le mouvement global est un mouvement de l'outil vers l'objet, au sens où tout outil soulève des questions qui portent sur lui-même et font de lui un objet d'étude. On peut aussi voir cela comme un mouvement qui, au sens des conceptions, amène les structures de contrôle à devenir opérateurs dans de nouveaux problèmes.

Continuité mathématique-informatique De même, il nous faut préciser que le découpage entre PA, AM et AI n'est pas non plus aussi tranché qu'on pourrait croire. Ainsi la séparation en trois familles de conceptions témoigne d'une graduation continue de conceptions entre mathématiques et informatique. Le découpage en trois paradigmes nous semble une bonne granularité pour fournir un outil d'analyse [1] et permettre une bonne représentation des μ-conceptions du texte du savoir savant et des réflexions sur sa production. Il existe donc aussi une continuité suivant l'axe PA-AI et il se peut que certaines activités se trouvent à cheval entre deux de nos conceptions. Notons tout de même qu'on peut voir une hiérarchie "temporelle" [2] entre PA, AM et AI. Ainsi, pour un même problème de $\mathscr{P}_{\text{A}}$, l'étude suit plutôt un mouvement de PA vers AI en passant par AM. C'est un mouvement qui part d'un point de vue constructif de l'algorithme (PA) vers un point de vue effectif (AM) et enfin vers un point de vue exécutoire (AI) qui s'accompagne d'un mouvement vers une spécification de plus en plus grande de l'algorithme.

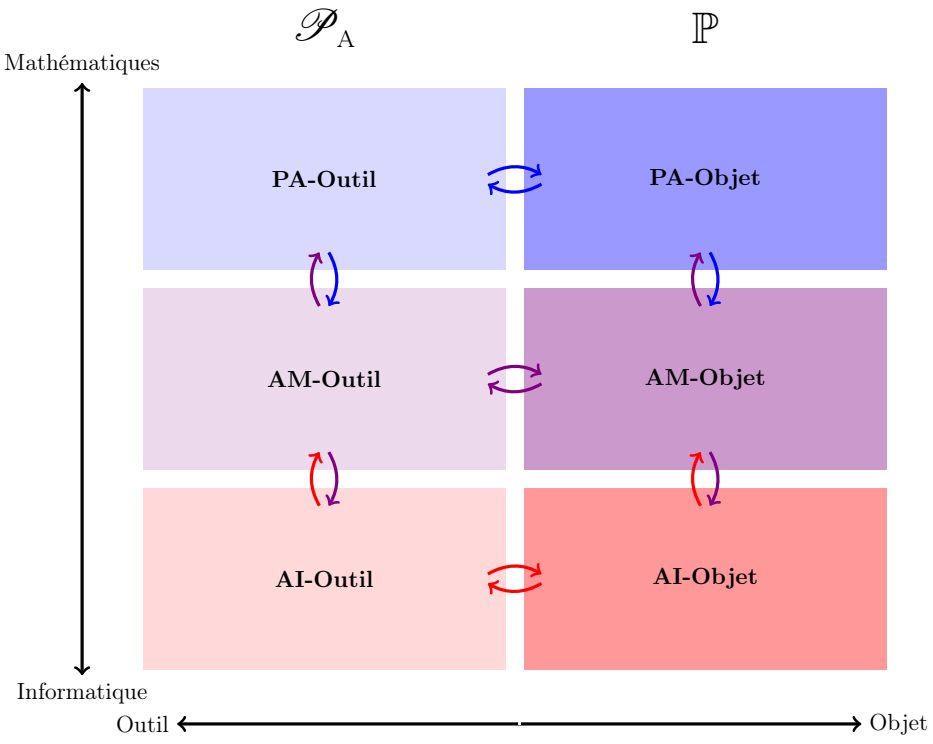
La continuité de l'axe PA-AI s'accompagne aussi d'une séparation grandissante de la preuve, de la sémantique et de la syntaxe. Ainsi, dans le paradigme PA, les trois sont

1. Cela sera questionné dans les chapitres suivants.
2. Qu'il ne faut surtout pas considérer comme une hiérarchie de valeur.

mêlés. Dans AM, on sépare la méthode (syntaxe et sémantique) de sa validation. Enfin, le paradigme AI vise à proposer une méthode purement syntaxique (l'ordinateur n'opère que des manipulations symboliques) à laquelle le concepteur donne un sens et qu'il peut valider. Ces rapports entre validation, sens et représentations des algorithmes, différents selon les paradigmes, mettent en jeu des éléments relevant de la sémiotique. Cela pourra, dans un raffinement du modèle, faire l'objet d'une étude sémiotique spécifique [3].

Nouveaux problèmes L'idée d'une continuité entre les paradigmes, dans le traitement d'un même problème, ne doit pas occulter le fait que l'activité algorithmique dans l'un des paradigmes conduit souvent à formuler et étudier de nouveaux problèmes. Cela fait aussi partie de la relation entre les μ-conceptions, de pouvoir soulever de nouveaux problèmes dans une autre μ-conception. L'étude épistémologique du *concept-problème* par Giroud (2011) peut apporter un éclairage sur la relation entre les différents problèmes en jeu dans une activité algorithmique, et leur formulation dans les différents paradigmes.

Récapitulatif En résumé, il ne faut voir dans ce modèle ni frontières nettes et immuables, ni échelle de valeur entre les conceptions. Ces remarques faites, nous pouvons introduire le schéma récapitulatif ci-dessous.



3. Les approches sémiotiques, notamment celles développées en didactique des mathématiques, devraient permettre cela. Il est cependant à noter qu'une telle approche ne peut être que complémentaire à l'étude épistémologique proposée ici. En effet, comme nous l'avons montré dans cette première partie, l'algorithme répond avant tout à une problématique constructive et systématique de résolution de problèmes, les questions d'expression des algorithmes venant se greffer ensuite par la nécessité d'un langage pour exprimer ces algorithmes. Une vigilance devra donc être exercée pour une étude sémiotique, afin de ne pas réduire l'algorithmique à une activité uniquement langagière, ce qui est, épistémologiquement, une vision réductrice de l'algorithme (mais courante, comme la suite nous le montrera).

3.4 Relation aux ASPECTS

Le modèle des μ-conceptions que nous proposons ici, reprend et précise de nombreux ASPECTS du chapitre 1.

L'aspect EFFECTIVITÉ est présent transversalement à chaque conception. En particulier ce sont les opérateurs des conceptions-outil qui portent cet aspect. Nous avons précisé au paragraphe précédent que différents niveaux d'effectivité sont en jeu dans chaque paradigme (du constructif à l'exécutoire).

L'aspect PROBLÈME est présent de manière très forte, par notre choix de formuler les problèmes en instances et questions, mais aussi par la précision des différentes familles de problèmes. Le nouveau modèle précise la relation problème-algorithme et la place des problèmes dans l'activité algorithmique.

Les aspects COMPLEXITÉ et MODÈLES THÉORIQUES sont présents dans les conceptions-objet du modèle. Leur rôle est précisé grâce à la notion de conception elle-même. La complexité relève essentiellement d'une famille de problèmes de $\mathbb{P}$. On pourrait aussi, dans une analyse plus poussée associer des opérateurs et des structures de contrôle spécifiques aux problèmes de complexité.
Les MODÈLES THÉORIQUES relèvent plutôt des structures de contrôle et des opérateurs des conceptions-objet. Cependant, des problèmes spécifiques peuvent être posés relativement à ces modèles.
Nous avions vu que MODÈLES THÉORIQUES et COMPLEXITÉ sont des aspects caractéristiques de l'algorithmique, et spécifiques. Le modèles précise bien leur rapport au concept algorithme.

L'aspect PREUVE est sans doute celui dont le rapport à l'algorithme était le plus délicat dans le modèle du chapitre 1. En particulier l'utilisation de l'algorithme comme outil de preuve, rendait difficile la classification outil-objet. De plus, le rapport du concept algorithme à la preuve d'algorithmes n'est pas du même ressort que celui à la preuve algorithmique ou celui aux preuves formelles.
Le modèle des μ-conceptions précise ces divers liens et la preuve apparaît à plusieurs niveaux :

- Elle est structure de contrôle des conceptions-outil et opérateur dans les conceptions-objet.
- Elle est aussi un paradigme pour l'algorithme : PA.
- Elle peut adopter diverses formes et s'appuyer sur des logiques plus ou moins formelles selon les paradigmes.

Enfin, ce modèle de μ-conceptions nous à permis de raffiner la dualité outil-objet, proposée pour classifier les ASPECTS. Notamment, grâce à aux notions de problème et d'instanciation, nous avons pu décrire une gradation dans cette relation outil-objet.

Conclusion et nouvelles questions

Le modèle cK¢ nous a permis de proposer un modèle de conceptions pour l'algorithme. Ces conceptions tiennent compte de la relation entre mathématiques et informatique en ce qui concerne l'algorithme. Le modèle proposé complète le modèle d'ASPECTS précédent, en précisant notamment le rôle des problèmes et de la preuve. Il apporte aussi une dimension

d'activité algorithmique qui n'apparaissait pas dans le précédent modèle.

Ce modèle par conceptions s'appuie aussi sur les diverses formes que peuvent prendre les algorithmes, et, en ce sens, permet de prendre en compte une dimension sémiotique, ce qui était peu présent dans le modèle par aspects. Nous nous sommes concentré sur une description générale des différents systèmes de représentation afin de proposer une classification en grandes conceptions.

Ce nouveau modèle doit permettre d'étudier les transpositions en place dans d'autres institutions. En particulier, nous proposons de traiter les questions Q5, Q6 et Q7 au regard de ce modèle. Ce qui nous permet de reformuler :

Question $\mathbf{Q5_{conceptions}}$. *Quelles conceptions sont représentées dans les programmes de mathématiques et documents d'accompagnements du lycée ? Les conceptions présentes sont-elles complètes ? Sont-elles modifiées par rapport à celles du savoir savant ? Relèvent-elles de l'outil ou de l'objet ? Quelle place est donnée aux structures de contrôle ? Qu'en est-il dans les programmes de la spécialité ISN ?*

Question $\mathbf{Q6_{conceptions}}$. *Quelles conceptions sont représentées dans les manuels de mathématiques du lycée ? Les conceptions présentes sont-elles complètes ? Sont-elles modifiées par rapport à celles du savoir savant ? Relèvent-elles de l'outil ou de l'objet ? Quelle place est donnée aux structures de contrôle ?*

Question $\mathbf{Q7_{conceptions}}$. *Quelles conceptions sont représentées dans les ressources en ligne ? Les conceptions présentes sont-elles complètes ? Sont-elles modifiées par rapport à celles du savoir savant ? Relèvent-elles de l'outil ou de l'objet ? Quelle place est donnée aux structures de contrôle ? Y-a-t-il une différence entre les documents de formation et ceux pour la classe ?*

Le modèle de conceptions que nous avons développé est concerné par la question Q4. On peut maintenant reformuler cette question :

Question $\mathbf{Q4_{conceptions}}$. *Notre modèle de μ-conceptions pour l'algorithme s'accorde-t-il avec les conceptions des chercheurs ? Le modèle peut-il nous informer sur les variations que l'on pourrait entrevoir dans les discours de chercheurs ?*

Conclusion, retour sur les questions de recherches et nouvelles questions

Résultats

Nous avons proposé dans cette partie une étude épistémologique du concept d'algorithme. L'étude des textes du savoir savant et l'analyse d'exemples choisis nous a permis de dégager un premier modèle pour décrire le concept algorithme et les aspects fondamentaux qui lui sont liés. Cela nous a permis de répondre à la question Q1 et de proposer une caractérisation de la dualité outil-objet pour l'algorithme.

En étudiant pensée algorithmique et pensée mathématique à l'aide des cadres théoriques du "mathematical thinking", nous avons pu éclaircir la relation entre ces deux pensées. En particulier, cela nous a permis de mettre en avant et d'étudier l'activité algorithmique. Cela constitue une réponse à la question Q2.

En nous appuyant sur les deux analyses précédentes, nous avons proposé un modèle plus complet de l'activité algorithmique (prenant en compte l'activité algorithmique en mathématique et en informatique). Le choix d'un cadre théorique adapté à nos questions théoriques et pratiques constitue un premier résultat important. Ce cadre théorique est le modèle cK¢, plus précisément la notion de μ-conception, qui a été rarement utilisée. Nous avons adapté ce modèle à nos problématiques en définissant une notion de problème spécifique à la description de μ-conceptions et adaptée au concept algorithme. Dans ce cadre, nous avons décrit un modèle permettant de répondre à Q3.

Retour méthodologique

La construction successive de ces deux modèles épistémologiques (ASPECTS et μ-conceptions), ainsi que la réflexion intermédiaire sur la pensée algorithmique montrent une évolution des modèles proposés. Leur ordre de présentation correspond à l'ordre chronologique de nos recherches.

Le premier modèle peut sembler naïf, mais il joue un rôle important dans la construction du second et dans sa compréhension. Nous conserverons ce premier modèle dans les outils pour les chapitres suivants, car il peut constituer un premier niveau d'analyse propre à soutenir l'étude des conceptions.

La définition choisie d'un problème et l'étude de la dualité outil-objet dans le modèle cK¢, sont des outils méthodologiques susceptibles d'être adaptés à l'étude épistémologique de concepts. Il nous semble utile de pointer le potentiel de ces outils pour d'autres études.

Suite du travail

Le travail épistémologique développé ici vise à fournir des outils pour étudier les conceptions individuelles et institutionnelles, cela dans le but de mieux comprendre les phénomènes de transposition didactique de l'algorithme. Une autre raison de cette étude est de fournir un support épistémologique pour construire des situations autour de l'algorithmique.

Ces questions seront abordées, au regard des résultats que nous venons de présenter, dans les troisièmes et quatrièmes parties. Les principaux éléments de cette étude épistémologique que nous utiliserons pour cela sont :

- le modèle par aspects et la classification entre aspects-outil et aspects-objet, qui nous permettront en particulier d'étudier les discours sur l'algorithme,
- les différents paradigmes du modèle de μ-conception, qui permettront notamment l'étude des formes sous lesquelles on retrouve l'algorithme,
- les familles $\mathscr{P}_{\text{A}}$ et $\mathbb{P}$, pour étudier les problèmes proposés et distinguer l'algorithme-outil de l'algorithme-objet.

Dans un premier temps, nous avons souhaité confronter nos modèles aux conceptions de chercheurs, afin de valider "expérimentalement" nos résultats. C'est l'objet de la partie suivante.

L'utilisation du modèle cK¢ pour la construction d'un modèle du savoir-savant, via la notion de μ-conception, soulève des questions théoriques et des perspectives dont il nous faudra discuter.

Deuxième partie

Entretiens avec des chercheurs

Motivations et questions de recherche

Comme nous l'avons souligné précédemment, l'étude des aspects principaux du concept d'algorithme et le modèle de μ-conceptions pour l'algorithme (lui-même basé sur l'analyse de l'*algorithmic thinking*) sont plutôt théoriques. Pour les développer, nous nous sommes essentiellement basé sur le savoir savant (ouvrages de références en mathématiques ou informatique), et sur le discours sur l'activité algorithmique des chercheurs dans des écrits réflexifs sur leur pratique.

Avant d'utiliser ces modèles pour analyser programmes, manuels ou autres ressources pour l'enseignement secondaire, il nous semble bon de les confronter aux conceptions des chercheurs. Cela doit permettre de tester la robustesse et la validité de nos modèles du point de vue épistémologique. Mais cela doit aussi nous permettre de réaliser une première analyse avec ces modèles et de tester leur effectivité pour répondre concrètement à nos questions de transposition. Plus précisément :

1. L'ensemble des conceptions de chercheurs autour de l'algorithme et de l'algorithmique constitue un aperçu des μ-conceptions. En considérant ces chercheurs comme représentants de l'institution productrice du savoir savant, on devrait révéler un ensemble de conceptions très proches de nos μ-conceptions. Étudier les conceptions de chercheurs constitue alors une **validation expérimentale** de nos modèles épistémologiques.
2. Les modèles (aspects fondamentaux et μ-conceptions) que nous proposons ont été construits pour répondre à des questions de recherche concernant la transposition didactique. Il s'agit donc de pouvoir les utiliser pour analyser les conceptions d'une institution au regard du savoir savant. Appliquer ces outils pour étudier la conception de chaque chercheur doit pouvoir permettre, non seulement de **repérer des conceptions dominantes** selon les disciplines et les spécificités des chercheurs, mais surtout de tester notre modèle et les méthodes d'analyse afin de **proposer un outil d'analyse systématique**. Les conceptions des chercheurs n'étant, a priori, pas trop éloignées des μ-conceptions, nous devrions pouvoir raffiner nos outils d'analyse grâce à elles.

Cette deuxième partie se veut donc une transition entre l'analyse épistémologique du concept présentée dans la partie précédente de cette thèse et la mise en œuvre d'outils effectifs pour l'étude de la transposition du concept de la partie 3 (que nous appliquerons au cas de l'enseignement secondaire français).

Rappelons les questions de recherche qui motivent cette partie telles que nous les avons formulées à la fin de la première partie :

Question $\mathbf{Q4_{aspects}}$. *Notre modèle des aspects de l'algorithme s'accorde-il avec la perceptions qu'ont les chercheurs ? Le modèle peut-il nous informer sur les variations que l'on*

pourrait entrevoir dans les discours de chercheurs ?

Question Q4$_{\textbf{conceptions}}$. *Notre modèle de μ-conceptions pour l'algorithme s'accorde-t-il avec les conceptions des chercheurs ? Le modèle peut-il nous informer sur les variations que l'on pourrait entrevoir dans les discours de chercheurs ?*

Chapitre 4

Entretien avec des chercheurs - Confrontation des modèles

Sommaire

1 Construction et gestion des entretiens 75
1.1 Choix de l'étude par des entretiens 75
1.2 Construction des entretiens 76
1.3 Réalisation et traitement des entretiens 79
2 Analyse des entretiens 79
2.1 Méthodologie pour la validation des modèles 79
2.2 Aspects fondamentaux dans les entretiens 80
2.3 μ-conceptions dans les entretiens 85
Conclusion 89

1 Construction et gestion des entretiens

Dans cette section, nous allons discuter des méthodes employées et des données étudiées. En particulier, nous aborderons le choix des entretiens pour l'étude des conceptions des chercheurs, la construction, la réalisation et le traitement de ces entretiens avant l'étude des données recueillies.

1.1 Choix de l'étude par des entretiens

Pour répondre aux questions ci-dessus, nous avons choisi de réaliser des entretiens avec des chercheurs en mathématiques fondamentales, en mathématiques appliquées et en informatique. Il nous a semblé pertinent de nous concentrer aussi sur des champs à l'intersection entre mathématiques et informatique, tels que la théorie des graphes, la combinatoire, la géométrie algorithmique, etc [1]. Les chercheurs de ces disciplines ont une activité relevant souvent des mathématiques (c'est-à-dire qu'ils ont une démarche visant à la preuve mathématique) mais leur lien privilégié à l'informatique devrait nous apporter des éléments riches concernant le concept d'algorithme.

1. En France, ces champs sont regroupés dans le GDR Informatique Mathématique.Le site du GDR-IM en donne une description assez précise qui montre bien l'intérêt que peuvent avoir ces disciplines pour notre étude : `www.gdr-im.fr`. Le GDR Recherche Opérationnelle couvre aussi de telles disciplines.

Pour étudier les conceptions des chercheurs, l'outil que constitue l'entretien nous a semblé la méthode la plus pertinente pour notre objectif d'étude des conceptions de l'algorithme et des pratiques des chercheurs. En effet, interroger en face-à-face les chercheurs devrait permettre de toucher à leur rapport au concept en permettant une expression plus libre qu'elle ne le serait par écrit. L'aspect individuel des entretiens et l'anonymat des résultats devraient aussi contribuer à une plus grande liberté de parole. Concernant ce choix, citons Robert (1992) qui souligne, concernant les problèmes méthodologiques en didactique des mathématiques, la pertinence des entretiens pour l'étude des représentations des sujets. Citons aussi Burton (2004) qui a étudié, par des entretiens avec des chercheurs en mathématiques, leurs pratiques (pour chercher, apprendre, etc.).

Plus précisément, notre choix se porte sur des entretiens semi-directifs qui permettent une certaine liberté dans la discussion tout en la guidant par une liste de questions et de points à aborder. Cela permet aussi de poser des questions supplémentaires en réaction à certaines réponses et de faire reformuler ou détailler certaines autres ou encore de reformuler les questions lorsque l'interviewé le demande ou que sa réponse montre une interprétation de la question différente de celle attendue.

Pour permettre l'étude, les entretiens seront enregistrés et retranscrits. Les chercheurs devront remplir un court questionnaire pour fournir des informations personnelles [2] (situation, domaine de recherche, enseignements, lien éventuel à l'enseignement secondaire, etc.). Les chercheurs ne connaissent pas à l'avance le sujet de l'entretien pour qu'ils ne réfléchissent pas trop à la question ou consultent des ouvrages de référence. Nous leur dirons seulement qu'il s'agit d'une discussion autour d'un concept mathématique et de son enseignement. Pour des raisons de disponibilité des chercheurs, les entretiens ne devront pas excéder une trentaine de minutes.

1.2 Construction des entretiens

La construction des entretiens et leur réalisation a eu lieu avant la production du second modèle épistémologique. C'est-à-dire qu'elles ne se sont appuyées que sur l'analyse du concept algorithme sous forme des aspects fondamentaux et sur les questionnements à l'origine de notre travail. Les questions posées aux chercheurs sont donc plutôt marquées par cela. Cependant, les μ-conceptions sont une relecture et une précision de cette précédente analyse et les entretiens proposés restent, comme nous le verrons, pertinents pour valider et mettre en œuvre tous les résultats de la première partie.

Nous avons conçu les questions pour les entretiens selon deux axes :

- le concept algorithme dans les sciences mathématiques et informatiques, sa définition et son rôle,
- le concept algorithme dans l'enseignement des mathématiques dans le secondaire.

Il nous semble en effet intéressant de voir si les mêmes conceptions sont soulevées concernant l'enseignement de l'algorithmique (en particulier dans les mathématiques). De plus, le fait d'aborder l'enseignement peut faire soulever certains points par l'interviewé qui lui semblent évidents (ou partagés avec l'intervieweur) et qu'il n'avait pas explicités concernant le concept d'un point de vue purement scientifique.

2. Ce questionnaire est en annexe B.

Première partie de l'entretien

Pour cette partie, trois points ont été abordés :

- Définition d'algorithme. Critère de reconnaissance des algorithmes. Exemples et non-exemples d'algorithmes.
- Place et rôle des algorithmes dans l'activité mathématique, dans le champ du chercheur interrogé et dans son travail.

En particulier, concernant le premier point, la discussion sur la définition de l'algorithme doit permettre de soulever les points importants du concept algorithme selon le chercheur. La question de la reconnaissance des algorithmes a pour but d'aborder la question du langage et des représentations des algorithmes (et leur éventuel lien à la définition).
La demande d'exemples et de non-exemples d'algorithmes doit contribuer à la discussion autour de la définition et des critères importants que doit vérifier un algorithme. Cela peut aussi encourager et faciliter la discussion au cas où la question de donner une définition ne l'enclencherai pas.
La discussion autour de la définition, des exemples et des non-exemples a aussi pour objectif d'amener à parler de la frontière entre algorithmique et non-algorithmique. Pour cela nous proposons aux chercheurs de discuter de la "technique" de recherche des racines d'un polynôme de degré 2 par le calcul du discriminant [3].
L'autre point se décompose en deux questions portant directement sur l'utilisation des algorithmes dans les différentes disciplines et l'activité personnelle du chercheur interrogé.

Deuxième partie de l'entretien

Cette deuxième partie porte sur l'enseignement de l'algorithmique. Il est clairement précisé au chercheur le glissement vers cette nouvelle partie.
La trame pour cette partie est d'aller du général au particulier : de l'enseignement de l'algorithmique au secondaire (en général) aux contenus précis des nouveaux programmes du lycée, en passant par l'idée d'enseigner l'algorithmique dans le cours de mathématiques. Il ne s'agissait pas, bien entendu, de savoir si le chercheur était pour ou contre telle ou telle idée mais de favoriser l'expression des conceptions de l'algorithme au travers de celles qu'il pense présentes ou indispensables dans l'enseignement.

Nous faisons l'hypothèse que, dans certains cas, aborder le concept par la question de son enseignement pousse à formuler certains aspects du concept qui peuvent être considérés comme implicites dans une discussion sur le concept du point de vue purement scientifique. Nous pensons aussi qu'au travers des questions d'enseignement, le chercheur va être amené à insister sur les points qu'il considère centraux et importants pour l'enseignement de l'algorithme et sur les caractéristiques principales de l'algorithmique. Nous reviendrons sur ce point dans une conclusion concernant les outils méthodologiques utilisés.

À la fin de cette partie nous avons ajouté une réaction aux nouveaux contenus de « Notations et raisonnement mathématiques » des programmes. En demandant une réaction quant aux liens possibles entre ce contenu et celui d'algorithmique. Il nous semble a priori possible d'utiliser ce paragraphe pour discuter du lien entre activité algorithmique et raisonnement, et par extension du lien algorithme-preuve.

3. Cet exemple soulève la question de considérer tout ce qui est effectif comme étant algorithmique ou non. C'est une description effective d'une méthode de résolution, mais d'une part il s'agit simplement d'une formule avec une condition, d'autre part sa complexité est constante.

Liste des questions de l'entretien

La trame d'entretien obtenue est la suivante. Rappelons que les questions pouvaient être reformulées, des questions ajoutées et l'ordre plus ou moins respecté.

Algorithme et recherche :

(a) Comment définiriez-vous ce qu'est un algorithme ?

(b) Comment reconnaît-on un algorithme ?

(c) Exemples d'algorithmes, de non-algorithmes, cas limite.
Questionner les exemples. Proposer le discriminant.

(d) À quoi servent les algorithmes ?
Quel rôle jouent-ils dans votre discipline ?

(e) Rencontre-t-on des algorithmes dans vos travaux de recherche ?
Concevez-vous des algorithmes ? À quel niveau ?

Algorithme et enseignement :

(a') Pensez-vous qu'il est important d'introduire de l'algorithmique dans l'enseignement secondaire ? À quel niveau ? Dans quelle discipline ? Quels objectifs ? Quels contenus ?

(b') Une part d'algorithmique a été introduite dans l'enseignement de **mathématiques** en classe de seconde. Qu'en pensez-vous ? Voyez-vous un intérêt concernant l'apprentissage des mathématiques ?

(c') *Faire lire l'extrait « algorithmique » des programmes.*
Commentaires ?

(d') *Faire lire l'extrait « Notations et raisonnement mathématiques » des programmes.*
Commentaires ? Voyez-vous un lien avec l'algorithmique ?

Un entretien basé sur les ASPECTS mais adapté aux μ-conceptions

L'objectif à l'origine de ces entretiens, comme nous l'avons dit, était la validation du modèle des ASPECTS. Les questions ont particulièrement été choisies pour cela. Le but étant de vérifier que ces aspects étaient bien fondamentaux pour l'algorithme, les entretiens ne devaient pas être trop directifs et ne pas aborder les aspects de manière frontale.

Cela a mené à la production d'un corpus suffisamment riche pour valider aussi une grande partie des éléments du modèle des μ-conceptions.

En particulier, les questions de reconnaissance d'algorithmes peuvent faire référence aux systèmes de représentation ou aux opérateurs des conceptions. La demande d'exemples et la question du rôle de l'algorithme peuvent amener à préciser les différents problèmes dans lesquels l'algorithme intervient. La demande de non-exemples est très adaptée à faire aborder le paradigme PA.

Ainsi, non-seulement les entretiens devraient permettre de valider nos modèles mais aussi d'étudier les conceptions des chercheurs.

1.3 Réalisation et traitement des entretiens

Les entretiens ont été réalisés entre janvier et avril 2010. Nous avons interrogé 22 chercheurs se répartissant ainsi (selon leur propre considération [4]) :

	Mathématiques	Informatique
Recherche fondamentale	11	5
Recherche appliquée	7	5

Nous avons essayé de constituer un panel assez varié de chercheurs suivant les critères d'âge, de discipline (mathématique/informatique, fondamental/appliqué), de champs dans la discipline, de sexe, d'avancement dans la recherche et de relation à l'enseignement. Nous résumons ici par quelques statistiques les informations des 22 chercheurs interviewés [5] :

Hommes	Femmes
27	5

26-35 ans	36-45 ans	46-62 ans
5	11	6

Ens. Ch.		Chercheurs	
MC	PU	CR	DR
10	7	4	1

Section		
25e	26e	27e
7	6	9

Les chercheurs interrogés sont issus de laboratoires de l'agglomération Grenobloise. Il faut souligner la difficulté à trouver des chercheurs acceptant de se prêter à l'exercice.
Les premiers entretiens ont permis de valider la trame des questions, du point de vue du temps nécessaire, de la compréhension des questions et de la richesse des réponses des chercheurs.

Les entretiens ont été entièrement retranscrits et sont en annexe B. Certaines tournures orales ont été atténuées pour favoriser la compréhension du lecteur et lever certaines ambiguïtés liées à l'absence des intonations, modulations de la voix, hésitations, etc. Compte-tenu de nos questions de recherche, ceci ne gênera pas l'analyse et n'introduira pas de biais à l'interprétation, au niveau où nous nous plaçons.

2 Analyse des entretiens

2.1 Méthodologie pour la validation des modèles

Notre méthode d'analyse est simple. Il s'agit de repérer la présence des éléments de nos modèles dans les discours des chercheurs. Ici, nous sommes très proches du savoir savant et le modèle peut nous servir directement d'outil d'analyse, d'autant plus qu'il s'agit de valider expérimentalement ce modèle.
En ce qui concerne le modèle par ASPECTS, nous verrons si les aspects sont soulignés par les chercheurs (EFFECTIVITÉ, PROBLÈME, PREUVE, COMPLEXITÉ et MODÈLES THÉORIQUES). Nous préciserons quels éléments de ces aspects sont rencontrés majoritairement.

En ce qui concerne le modèle de μ-conceptions, notre objectif sera de repérer des éléments de chaque conception. Pour cela nous essaierons de repérer les différents problèmes, opérateurs, systèmes de représentations et structures de contrôle. Bien sûr, cela est très lié aux

4. Certains chercheurs considèrent qu'ils ont des thèmes de recherches dans différents champs. Voir en annexe B pour le questionnaire d'informations personnelles.
5. L'intégralité des données se situe en annexe B.

aspects que nous aurons rencontrés.

Nous illustrerons notre analyse par des citations des entretiens. Nous utiliserons le code (B$i.j$) pour faire référence à l'entretien avec le chercheur i, à l'intervention j. Nous nous autoriserons parfois à retirer certains termes oraux (*Heu*, *bref*, *bon*, etc) et cafouillages dans les citations pour permettre d'insister sur les points importants qu'elles illustrent.

2.2 Aspects fondamentaux dans les entretiens

Aspect EFFECTIVITÉ

L'EFFECTIVITÉ est un point évoqué par tous les chercheurs interrogés. Cet aspect apparaît à chaque fois, dès la définition que donne le chercheur de l'algorithme. On le retrouve aussi au travers des exemples de non-algorithmes (en négatif), dans la description de la limite entre algorithme et non-algorithme ou encore la discussion sur le discriminant. Cependant on peut constater des nuances dans la façon dont est évoqué cet aspect.

Dans ces définitions ou dans les critères de reconnaissance d'un algorithme, on retrouve presque toujours la notion de succession d'étapes ou d'instructions élémentaires (20 chercheurs [6]) :

> Alors un algorithme, c'est un enchaînement d'instructions qui permet à partir d'une entrée de produire une sortie [...] Donc ces instructions sont suffisamment élémentaires pour pouvoir ensuite être codée sur ordinateur. (B2.2-4)

Le lien à la machine et à la programmation est aussi fait très régulièrement dans les entretiens (19 ch.) :

> Un algorithme c'est, pour moi, une suite d'instructions qu'on donne à un ordinateur pour résoudre un problème qu'on a discrétisé auparavant. (B15.2)

Le côté automatique ou systématique de l'algorithme revient un peut moins souvent (10 ch.) mais on peut le considérer comme sous-entendu dans la notion de machine. On retrouve aussi le terme *effectif* qui peut faire référence à ce côté automatique.

> Un algorithme. Pour moi ça a un sens assez général. C'est toute procédure effective qui permet de mener à bien un calcul ou une tâche déductive automatisée. (B10.2)

On retrouve aussi la notion de construction (8 ch.), qui est parfois contenue dans le terme effectif.

> Je dirais que c'est un moyen constructif de... Faire quelque chose. Une construction, un calcul... (B17.2)

Un autre point souvent relevé est la non-ambiguïté d'un algorithme (10 ch.). Certains chercheurs notent que cela dépend du destinataire de l'algorithme (machine, humain, spécialiste, etc.). Cette notion est donc très liée au fait qu'un algorithme peut être exécuté ou appliqué par d'autres personnes, en particulier comme outil :

> Si on la donne à n'importe qui, cette personne-là sera capable, avec cet algorithme, de calculer les sorties à partir des entrées. Donc c'est quelque chose que

6. Pour alléger le texte, nous utiliserons l'abréviation ch. pour chercheur(s) lorsque nous citerons un effectif.

> l'on peut donner à droite à gauche et qui ne laisse pas d'ambiguïté sur la façon dont on fait les calculs. (B2.8)
>
> S'il y avait un technicien à ce moment-là [...] on peut lui mettre à plat quelque chose. Et effectivement, même s'il n'a pas des connaissances élevées, il sera capable de dérouler le fonctionnement. (B18.12, à propos du discriminant)

Cet aspect revient souvent concernant l'enseignement, pour lequel beaucoup de chercheurs précisent que l'enseignement de l'algorithmique ne doit pas être la transmission d'algorithmes aux élèves et les algorithmes qu'écrivent les élèves doivent être à destination d'autrui [7] :

> D'un côté, tu ne peux pas enseigner des maths sans jamais enseigner des algorithmes. D'un côté ça me paraît clair. De l'autre côté par contre, je trouve que [...] C'est souvent tentant d'enseigner trop d'algorithmes, trop de méthodes. (B21.44)
>
> Et ce n'est peut-être pas soi-même qui va devoir l'exécuter. Donc soit on le confie à un ordinateur, soit on le confie au voisin et on regarde s'il est capable de suivre l'algorithme que moi j'ai fait. [...] Alors qu'à présent, en maths, on voit des éléments de ce qu'est un algorithme mais c'est juste comme une suite de recettes pour les faire soi-même. Donc c'est là où c'est un peu moins profond (B18.42)

Le fait de manipuler des données finies et que la solution doit être donnée en un nombre fini d'étapes n'est pas très souvent signalé et apparaît assez rarement dans les définitions proposées. Cependant, on peut penser que certains chercheurs l'englobent dans le terme effectif, et cela est nécessairement contenu dans les notions de terminaison et de complexité dont nous parlerons plus loin. Cependant, certains points de vue présentent l'algorithme comme quelque chose de non-nécessairement fini. En analyse numérique, par exemple :

> Il y a des démonstrations que l'on construit de façon, on va dire, algorithmique. C'est-à-dire qu'on construit des suites qui vont tendre vers des objets mathématiques et on va démontrer que ces objets mathématiques sont au bon endroit. Mais d'abord il faut faire l'algorithme, c'est-à-dire la suite qui va te permettre d'avoir une idée de l'objet. (B14.39)

Aspect PROBLÈME

L'aspect PROBLÈME est évoqué par tous les chercheurs. Lui aussi est très souvent évoqué dans les définitions proposées. On le retrouve aussi beaucoup dans les réponses sur le rôle des algorithmes. Cet aspect est évoqué avec plus ou moins de précisions. Certains chercheurs évoquant, dans la définition, une tâche confiée à l'algorithme :

> Pour moi un algorithme, c'est une procédure automatisée qui accomplit une tâche qu'on souhaite lui voir accomplir. (B1.2)
>
> C'est un procédé systématique pour obtenir une certaine classe de résultats. Les algorithmes pour quelque chose... (B6.2)

7. Rappelons à nouveau que les questions des entretiens sur l'enseignement n'ont pas pour vocation de donner des pistes pour l'enseignement. Elles sont à voir comme un moyen de faire apparaître les conceptions des chercheurs.

Majoritairement, la notion de problème est évoquée de manière directe (18 ch.). Un algorithme résout un problème :

> Algorithme ça me fait penser, moi, à programmation. Manière d'implémenter, manière de résoudre un problème de manière automatique. (B7.4)

L'algorithme est donc souvent reconnu comme outil de résolution de problème, mais il est difficile de savoir quelle est la notion de problème qui est derrière le terme. En particulier, l'utilisation d'algorithmes pour aider le chercheur à résoudre un problème correspond à une notion de problème très différente de celle que nous voulons valider. On retrouve ce point de vue dans les positions de certains chercheurs, par exemple :

> Je ferais peut-être une distinction entre calcul numérique et algorithme. [...] C'est que justement, un algorithme, il y a certes une forme mais il répond avant tout à un problème. On pose une question et on veut une réponse à ce problème. Et de ce point de vue, le calcul numérique, souvent, n'entre pas exactement dans ce cadre parce que ce que rend le programme qui a été écrit n'est pas forcément lié de manière très claire à la question qui a été posée. [...] C'est du domaine de ce que j'appellerais plus largement de la simulation. (B20.14)

Les notions d'entrée et sortie, assez présentes (13 ch.) précisent un peu plus la relation algorithme-problème :

> Et ensuite on effectue une suite d'opérations. En prenant cette entrée et en transformant, une suite bien précise d'opérations, c'est ça l'algorithme. [...] Il y a une sortie aussi. (B14.8)

D'autres chercheurs (7 ch.) sont encore plus précis en évoquant la notion d'instance, le fait que l'algorithme réponde à toutes les instances d'un problème :

> Premier rôle, ils servent à avoir la solution concrètement quand on prend des instanciations de notre problème. (B8.20)

> C'est-à-dire, quel que soit ce que tu me donnes, ce que j'ai créé [un algorithme] va permettre de résoudre le problème, c'est pas quelque chose qui va dépendre de... Enfin ça va dépendre mais ça va pas... Quel que soit ce que tu mets ça va être résolu quoi. (B1.32)

Aspect PREUVE

L'aspect PREUVE se retrouve dans une majorité d'entretiens (16 ch.) . Il apparaît souvent concernant le rôle des algorithmes (question d). Cet aspect apparaît assez souvent aussi lorsque les chercheurs abordent les algorithmes présents dans leurs travaux (lorsqu'il y en a). Enfin, il semble que les questions de la deuxième partie sur l'enseignement soient favorables à faire émerger cet aspect.

La notion de preuve d'algorithme est très présente (12 ch.) et concerne essentiellement la correction des algorithmes :

> Faire un algorithme qui se tient et voir pourquoi est-ce qu'il fait effectivement le bon travail et prouver que ça marche, c'est clairement une bonne façon d'approcher la logique, ce genre de choses. (B3.26)

La terminaison est bien moins souvent abordée et est toujours présente avec la correction :

> Alors en algorithmique on aime bien montrer qu'un algorithme est exact, qu'il s'arrête toujours. Il y a quand même la même rigueur, la même notion de preuve [qu'en mathématiques]. (B11.36)

L'algorithme est aussi présenté comme outil de preuve (9 ch.), il permet de démontrer des résultats :

> Quand je m'adresse aux étudiants, je mets toujours les algorithmes en avant. Et je fais tomber la théorie, après les algorithmes. Donc je démontre que l'algorithme est juste et après on en tire des propriétés. (B5.16)

On retrouve aussi, plus précisément, la notion de preuve constructive ou algorithmique (11 ch.) et l'exemple spécifique de la récurrence (7 ch.) :

> Contrairement à certaines preuves qui sont complètement constructives et qui peuvent être ramenées à des algorithmes en quelque sorte. (B3.18)

> Dans les maths, il y a certaines preuves qui ne ressemblent pas du tout à des algorithmes... C'est pas du tout des preuves algorithmiques. Les preuves par l'absurde par exemple c'est pas du tout des preuves algorithmiques. Par contre, les preuves par induction, souvent il y a un algorithme caché derrière. (B4.20)

Comme c'est le cas ci-dessus, la référence aux preuves constructives est souvent apportée par la question de donner des exemples de choses non-algorithmiques. La valeur des preuves algorithmiques est aussi discutée par certains chercheurs :

> Pour moi, lorsqu'on dispose d'un algorithme, on a une meilleure preuve d'un résultat, le résultat est devenu effectif. Non seulement on a démontré le résultat, mais on a des procédures calculables qui permettent éventuellement de le quantifier, d'avoir des résultats quantitatifs plus précis. (B10.14)

Enfin, on retrouve aussi la notion de preuve formelle qui lie algorithme et preuve d'une autre façon encore, avec, par exemple, des références à Hales[8] (B20.18, B21.12).

Aspect COMPLEXITÉ

La COMPLEXITÉ est présente dans de nombreux entretiens (12 ch.). Elle est très majoritairement évoquée par la question de la complexité en temps (ou en nombre d'étapes) :

> Parce que chercher un maximum, ordonner, comme ça c'est relativement facile au niveau concept mais là apparaissent, effectivement, des notions nouvelles qui sont les notions de complexité. Et donc ça c'est important. Je veux dire que dans la notion d'algorithme, [...] maintenant on doit faire face à des notions d'efficacité, de complexité d'algorithmes. (B19.10)

La notion de complexité est liée au comportement de l'algorithme. Ainsi, la complexité n'a de sens que si le comportement de l'algorithme varie selon les instances. Ce point, sans être une caractérisation de l'algorithme, est questionné par plusieurs interviewés (en particulier en lien avec la question du discriminant) :

> [Le discriminant] c'est un peu limite parce que c'est vraiment décomposer une formule en trois quoi. Alors qu'on peut l'appliquer directement sans problème. Un algorithme c'est quand même, pour moi, quelque chose qui demande plusieurs étapes qu'on ne peut pas résumer en une formule. (B12.12)

8. Avec la preuve formelle de la conjecture de Kepler.

> Pour moi dans l'idée d'algorithme il y a un peu le côté taille variable [...] Il y a une théorie de la complexité derrière, sur le fait que si c'est des problèmes en taille fixe tout peut être résolu par du... Ouais donc c'est pas très clair. Pour moi un algorithme, c'est un truc qui permet de résoudre des problèmes dont la taille est pas nécessairement fixe mais... Bon là le fait que le problème de départ ait une taille fixe. Là c'est juste une porte logique, je sais pas, j'irais pas jusqu'à dire que c'est un algorithme. (B1.28-30)

Apparaissent aussi, les notions de classes de complexité, et d'approximations de solutions dans les cas de problèmes trop complexes (7 ch.) :

> Donc le but c'est d'arriver à [...] percevoir la complexité d'un algorithme et d'être capable de les classer. Alors, il y a le classement, le gros classement P-NP. Mais par rapport à ça, il y a toute une série de classes d'algorithmes, avec ensuite leur complexité explicite. On va dire tel algorithme est en $O(n)$, tel algorithme est en $O(n^2)$, etc. Où n est la taille des entrées. (B9.64)

> Alors soit on est confrontés à des problèmes faciles, auxquels cas il faut arriver encore à trouver un algorithme polynomial. Et puis si on est dans les problèmes plutôt difficiles, à ce moment-là, genre NP, NP-complétude ou NP-difficile, à ce moment-là, est-ce qu'on est quand même capable de trouver des solutions approchées mais avec des garanties performance par exemple. (B16.48)

Les questions de comparaison d'algorithmes et de recherche d'un algorithme optimal pour un problème sont aussi représentées dans les entretiens (7 ch.) :

> Il faut des algorithmes qui soient capables de traiter ces données et de dire : ben on fait mieux que les autres, heu... Et dans un temps qui est meilleur ou moins bon. Enfin bon, on fait mieux ou moins bien et on se positionne. (B22.36)

> Ce problème on sait bien y répondre avec un ordinateur, on peut le faire de façon efficace. Et on peut dire si la connaissance actuelle permet d'y répondre de la manière la plus efficace possible, ça sera un algorithme optimal. (B20.38)

Aspect MODÈLES THÉORIQUES

L'aspect MODÈLES THÉORIQUES est présent dans les entretiens mais c'est celui qui est le moins souvent évoqué (7 chercheurs en tout). On retrouve, cependant tous les aspects que nous avions soulignés. Cet aspect est le plus souvent évoqué suite aux questions a et b. Certains l'évoquent concernant leur travail de recherche.

On retrouve en particulier le modèle de machine de Turing, mais aussi la notion générale de modèle de calculabilité :

> La définition formelle, ça doit être avec des machines de Turing. (B14.2)

> Des gens l'ont fait plusieurs fois dans l'histoire, de théoriser sur papier quelle serait la mécanisation du calcul et de s'arrêter là. (B18.22)

On retrouve aussi, comme nous l'avons vu concernant l'aspect COMPLEXITÉ, les notions de problèmes P, NP ou encore NP-difficiles. Ces notions sont en effet très liées aux questions de complexité d'un point de vue théorique. Les notions de preuves formelles relèvent aussi, en quelque sorte, des MODÈLES THÉORIQUES.

Bilan

Validation du modèle L'étude documentaire nous avait permis de relever cinq ASPECTS FONDAMENTAUX. Nous les avons tous retrouvés dans les entretiens avec les chercheurs. Cette confrontation nous permet donc de valider le modèle.

Des conceptions incomplètes Tous les aspects ne sont (évidemment) pas évoqués par tous les chercheurs et différents points de vue pourraient être relevés. Nous reviendrons sur ces questions dans le chapitre suivant, en essayant de voir quelles conceptions sous-tendent ces variations de points de vue. Nous essaierons de voir en quoi les différents domaines de recherche peuvent être reliés à ces conceptions.

2.3 μ-conceptions dans les entretiens

Repérer les μ-conceptions dans les entretiens est plus difficile que pour les aspects. En particulier, certaines structures de contrôle sont difficilement visibles dans de tels entretiens. Nous allons nous concentrer essentiellement sur ce qui permet de délimiter les conceptions que nous avons décrites :

- les notions de problème et d'instance (en particulier $\mathscr{P}_{\text{A}}$ et $\mathbb{P}$ qui déterminent la dualité outil-objet),
- les systèmes de représentation qui déterminent fortement les paradigmes,
- les opérateurs et structures de contrôle lorsque cela est possible.

Le modèle de μ-conceptions reprend et précise plusieurs des éléments du modèle par ASPECTS. Une partie des points étudiés plus haut apporte déjà des réponses à la validation du modèle des μ-conceptions. Nous passerons plus rapidement sur ces points.

Problèmes, instances, $\mathscr{P}_{\text{A}}$, $\mathbb{P}$

Nous avons déjà vu que l'aspect PROBLÈME était présent dans les discours de nombreux chercheurs. La notion d'instance est évoquée et souligne la nécessité qu'un algorithme résolve un problème pour tout un ensemble de cas. La notion d'entrée-sortie va aussi dans ce sens.
Ce point évoque l'algorithme en tant qu'outil. Un outil identique pour résoudre plusieurs instances du problème. Comme souligné ici :

> Un algorithme simple dans la vie pour monter un escalier c'est : tant qu'il y a une marche, monter la marche. Quand il n'y a plus de marche, c'est terminé. C'est un algo simple mais ça permet de monter n'importe quel escalier de n'importe quelle taille. (B1.18)

Les exemples proposés par les chercheurs sont très majoritairement [9] des problèmes de $\mathscr{P}_{\text{A}}$ ou des algorithmes les résolvant. Les plus cités sont :

- Les tris
- L'algorithme d'Euclide
- La méthode du pivot de Gauss
- Les opérations posées (addition, division, etc.)
- Les tests de primalité

9. À l'exception, peut-être, des méthodes et schémas numériques, qui ne sont pas toujours des algorithmes au sens de notre définition.

– Des algorithmes sur les graphes (parcours, coloration, etc.)
– Des méthodes numériques (Newton, Cauchy-Lipschitz...)

Les exemples choisis par les chercheurs (à l'exception de ceux tirés de la vie courante) mettent tous en jeu des problèmes avec une infinité d'instances. Même si leurs définitions sont larges, il semble que les chercheurs conçoivent cet aspect comme représentatif. Ces exemples illustrent parfaitement bien des problèmes de $\mathscr{P}_{\text{A}}$.

L'algorithme en tant qu'objet est aussi représenté. Tous les chercheurs ne l'évoquent pas et tous n'abordent pas les mêmes aspects, mais les problèmes de $\mathbb{P}$ se dégagent tout de même assez nettement. Cela passe, par exemple, par la reconnaissance de l'étude des algorithmes comme un champ d'étude :

> C'est un objet d'étude, à la fois pour les gens qui font des schémas numériques et qui montrent [...] qu'utiliser l'algorithme qu'ils proposent va donner une approximation de la véritable solution mathématique. Et puis c'est un objet d'étude en informatique théorique aussi de comprendre la complexité des algorithmes, etc. C'est à la fois un outil, un objet d'étude et quelque chose qu'on aime avoir dans ses résultats. (B8.20)

Nous avons vu aussi que de nombreux chercheurs évoquent l'aspect COMPLEXITÉ qui correspond à l'une des problématiques majeures de l'algorithme en tant qu'objet. Les MODÈLES THÉORIQUES font aussi référence à $\mathbb{P}$.

Enfin, la notion de preuve d'algorithme, évoquée dans de nombreux entretiens (12 ch.), fait référence au problème $\mathfrak{p}_0$. Il s'agit de montrer que l'algorithme résout bien le problème étudié. Comme nous l'avons précisé dans notre modèle, ce problème est à la charnière entre $\mathscr{P}_{\text{A}}$ et $\mathbb{P}$. Il se réfère à la fois à la structure de contrôle dans les conceptions-outil et d'opérateur dans les conceptions-objet. Cette validation de l'algorithme est clairement reconnue comme étant d'ordre mathématique :

> On est beaucoup dans la discipline informatique théorique, à considérer qu'on fait un travail semblable à celui des mathématiciens. Probablement, l'objet n'est pas toujours exactement le même mais les méthodes sont les mêmes. (B20.82)

Systèmes de représentation et opérateurs de AI

Nous n'avons pas évoqué du tout la question des systèmes de représentation. On trouve des points de vue très divers sur ces systèmes de représentation des algorithmes. Comme il s'agit des représentations des opérateurs, il est parfois difficile de les distinguer dans les discours et l'on remarque souvent que les chercheurs ont des difficultés à détacher le fond de la forme.

On trouve bien évidemment de très nombreuses références aux langages de programmation et aux opérations dites élémentaires de ces langages (affectations, boucles...) :

> Donc ce serait essayer de traduire le théorème dans le langage de l'ordinateur. Que ce soit du Fortran, du machin, du Pascal, du Mathlab. (B15.18)

> Enfin, pour moi un programme informatique c'est un algorithme. Après c'est pas une définition précise de l'algorithme. Donc pour moi un algorithme c'est équivalent à un programme informatique (B3.16)

Certains considèrent l'algorithme comme quelque chose qui se situe au dessus des langages de programmation mais qui leur est fortement lié :

> Au bout du compte l'objectif c'est de faire exécuter de façon automatique par une machine un certain type, je dirais, d'actions. Donc c'est une couche, avant, qui se veut indépendante des langages, mais qui est quand même essentiellement basée sur les principes d'itérations, de boucles... Mais c'est aussi profondément lié à la structure des machines, je pense. (B19.2)

On reconnaît clairement ici, le paradigme ALGORITHME INFORMATIQUE (AI) ainsi qu'une référence à un pseudo-code qui serait plus abstrait mais qui serait très proche des langages de programmation. On a ici une vision du langage de l'algorithme comme une abstraction du langage de programmation. On glisse vers le paradigme AM.

Systèmes de représentation et opérateurs de AM

D'autres soulignent aussi que les algorithmes peuvent être exprimés dans un langage moins rigoureux :

> La difficulté de l'algorithmique c'est : à quel niveau on place les descriptions ? Si on est plus proche de la machine, c'est un langage formel où il n'y a pas d'ambiguïtés. C'est vrai que nous on fait de l'algorithmique à un plus haut niveau où c'est plus des étapes et il reste une part d'interprétation. (B11.2)

Il s'agit du langage du paradigme de l'ALGORITHME MATHÉMATIQUE (AM). On retrouve aussi ce paradigme dans le fait de pouvoir exprimer un algorithme dans un langage très peu formalisé :

> Une séquence d'instructions bien définies, pas nécessairement en pseudo-code. [...] En français. Par exemple pour faire l'addition de deux nombres. (B4.6-8)

On retrouve donc différents niveaux de formalisme dans l'expression des algorithmes. On retrouve aussi la nécessité de fixer des opérations de base pour traiter un problème :

> Un algorithme, c'est une suite d'instructions, qui peut être écrite en langage naturel. Donc on entend par instruction une commande de base, la granularité étant à définir par avance. (B13.2)

Le paradigme AM n'est pas uniquement à considérer comme une étape précédent la programmation. Plusieurs chercheurs précisent que leur activité se limite à AM :

> Mais par contre, dans mon travail plus classique, je ne fais que des algorithmes. [...] Je les fais, pas forcément au sens où je les programme après. Justement, on en revient là à la description... À une description... Littérale du travail, où on se convainc en lisant qu'on pourrait le faire. [...] Souvent, je me satisfais d'un article qui va décrire et qui va convaincre le lecteur que ça peut être réalisé. (B20.44-46)

Certains insistent sur les différences de langages et d'opérateurs en fonction de qui exécute l'algorithme (humain, machine) ou du modèle de calcul dans lequel on se place :

> Ce n'est pas pareil d'avoir un algorithme qui doit être implémenté sur une carte à puce qu'un algorithme qui doit être implémenté sur un supercalculateur. (B9.111)

> Cet aspect séquencement c'est important [...]. Alors moi, qui travaille dans le monde du parallélisme, parfois je m'en fous qu'une étape soit avant l'autre. [...] Mais c'est vrai que c'est lié au modèle d'ordinateur. Donc par exemple si on réfléchit au modèle quantique d'ordinateur, c'est encore autre chose. (B16.116)

Cela peut évoquer aussi la représentation ou l'encodage des objets manipulés :

> Le fait de dire qu'un objet a une forme. Est-ce que ça veut dire qu'on va le représenter par un maillage avec des polygones ? Est-ce que ça veut dire qu'on va prendre une fonction de densité dans l'espace ? [...] Il se peut que selon le problème à régler, l'une soit plus riche que l'autre. (B18.34)

> On peut avoir des codages des entiers qui sont redondants et qui permettent d'éliminer les problèmes de retenues, mais en ayant d'autres défauts. On perd sur autre chose, par exemple s'ils sont redondants c'est que ensuite il faut être sûr de retraduire le résultat en langage courant. (B14.12)

Ce point pourrait aussi faire référence à AI. Il est difficile dans ces entretiens de percevoir la nuance que nous avons faite entre structure de données et types de données abstraits.

Systèmes de représentation et opérateurs de PA

Comme nous l'avons déjà évoqué concernant l'aspect PREUVE, la PREUVE ALGORITHMIQUE (PA) est présente dans les entretiens (11 ch.). Ce type de preuve est reconnu comme contenant ou représentant des algorithmes :

> Il y a les choses qui vont être présentées sous forme d'algorithmes, soit dans un langage de programmation, soit dans un simili-langage... et là c'est assez explicite. Soit il y a des résultats qui sont constructifs et que l'on peut rendre algorithmiques ou qui sont déjà présentés, dans la démonstration, algorithmiques. (B8.6)

En particulier les preuves par récurrence sont classées dans cette catégorie :

> Dans certaines des thématiques c'est souvent des incidences de méthodes de preuve, d'une preuve constructive de quelque chose, ou une récurrence pour donner un résultat. [...] Parce que résoudre un problème de manière récursive, tu peux en pratique faire un algo à partir de ça. (B1.44)

Nous ne revenons pas plus sur ce point qui a déjà été développé pour l'aspect PREUVE. On peut simplement conclure que le paradigme PA est bien présent et que c'est l'opposition aux preuves non-constructives qui donne du sens à la preuve algorithmique.

Structures de contrôle de $\mathscr{P}_{\mathbf{A}}$ et opérateurs de $\mathbb{P}$

Nous avons déjà abordé certains éléments dans la partie concernant l'aspect PREUVE. Nous ne reviendrons pas sur les points déjà discutés. En particulier nous avions relevé que la validation des algorithmes et des programmes était bien considérée comme relevant de la validation mathématique. Les entretiens menés ne permettent pas de valider beaucoup plus en détail les spécificités de ces structures de contrôle des opérateurs relatifs aux problèmes de $\mathbb{P}$. On peut tout de même relever quelques remarques faites par des chercheurs.

En particulier, les nuances que nous avions présentées concernant les structures de contrôle de AI-outil se retrouvent dans certains entretiens :

> Il y a aussi l'aspect vérification du programme. [...] On peut avoir un algorithme correct et un programme qui ne l'est pas. (B9.101)

On retrouve aussi des références à des opérateurs précis pour traiter certains problèmes de $\mathbb{P}$, ici dans le cas de AM :

> En gros, mes travaux de recherche il y a deux techniques, [...] c'est des preuves de difficulté, NP-complétude, et des algorithmes. Donc soit des algorithmes

> polynomiaux, soit exponentiels et on essaie d'en faire quand même quelque chose. (B4.50)

Enfin, on peut noter que même si la validation expérimentale existe dans l'étude des algorithmes, la validation mathématique apporte une valeur ajoutée :

> La plupart des gens faisait des heuristiques qui n'étaient absolument pas fondées. Et là, le travail c'est de développer des algorithmes, qui soient efficaces, mais surtout qui soient garantis. [...] Donc de ne pas juste avoir des heuristiques, mais vraiment, en plus, d'associer à l'algorithme des propriétés mathématiques sur la sortie. (B2.32)

Bilan

Validation On retrouve bien les éléments du modèle de μ-conceptions dans les entretiens des chercheurs. En particulier, les classes de problèmes qui permettent de décrire les conceptions outil et objet ($\mathscr{P}_{\text{A}}$ et $\mathbb{P}$) sont clairement identifiées. Les systèmes de représentation qui caractérisent les différents paradigmes sont aussi en jeu. Bien entendu, notre découpage en trois paradigmes ne peut pas réellement être validé, mais on peut tout de même remarquer qu'il s'agit d'une granularité qui semble adaptée à l'analyse.

Diversité On peut remarquer que tous les chercheurs ne considèrent pas tous les paradigmes. La possibilité de coexistence des paradigmes permet d'analyser cela en termes de conceptions des chercheurs, nous reviendrons sur cela dans le chapitre suivant.

Limites Nous avons pu voir certaines limites des entretiens pour repérer les structures de contrôle des conceptions-outil et les opérateurs des conceptions-objet. Cela peut être lié aux choix des questions ou encore aux limites de la modalité d'entretien elle-même. Nous reviendrons sur cette question dans la conclusion.

Conclusion

Validation des modèles

Les entretiens menés nous ont permis de valider les deux modèles proposés. Bien que tous les chercheurs ne fassent pas référence à tous les éléments des modèles, on peut considérer que ces éléments sont suffisamment présents pour confirmer nos choix et nos analyses épistémologiques.

Les variations que nous avons pu constater dans le rapport des chercheurs à l'algorithme, nous poussent à questionner un peu plus ces rapports de manière individuelle. Comprendre les spécificités de chaque conception de chercheur est un bon moyen d'éprouver nos modèles pour en dériver un outil d'analyse. C'est ce que nous ferons dans le chapitre suivant.

Méthodologie - Retour

Nous avons pu constater que les entretiens réalisés n'étaient pas parfaitement adaptés à la validation du modèle de μ-conceptions. D'une part, les entretiens ont eu lieu avant le développement de ce modèle. D'autre part, le modèle de μ-conceptions s'intéresse plutôt à l'activité algorithmique. Valider un tel modèle par des entretiens est difficile car les structures de contrôle ne sont pas aisément repérables.

Une possibilité pour accéder aux structures de contrôle est d'observer l'activité du chercheur ou de s'intéresser à des traces de son activité. Cela n'est pas évident avec des chercheurs, mais représente une piste envisageable pour poursuivre la validation expérimentale du modèle épistémologique.

À travers ces entretiens, nous avons montré qu'une validation expérimentale d'un modèle épistémologique est possible. La confrontation d'un modèle basé sur la littérature du savoir savant aux conceptions de membres des institutions productrices du savoir montre une certaine distance. Il est important que cette distance ne soit pas trop grande et que la conception globale de l'ensemble des chercheurs soit en adéquation avec le modèle théorique proposé. De manière générale, l'analyse des conceptions des chercheurs enrichit l'épistémologie contemporaine d'un concept. En ne se basant que sur le texte du savoir savant, le risque est d'accéder à un objet uniformisé qui ne tient pas compte des spécificités des champs qu'il couvre.

Chapitre 5

Conceptions des chercheurs - Construction d'un outil

Sommaire

1 Motivations 91
2 Méthodologie 92
2.1 Éléments pour l'analyse des conceptions 92
2.2 ASPECTS et paradigmes en jeu 92
3 Les conceptions des chercheurs 93
3.1 Tableau récapitulatif 93
3.2 Résultats 94
Conclusion - Vers une grille d'analyse 95

1 Motivations

À partir du travail épistémologique développé dans la première partie et validé dans le chapitre précédent, nous souhaitons étudier les transpositions didactiques de l'algorithme au travers de sa conception dans diverses institutions. Une telle étude sera menée dans la partie suivante. Avant cela, et dans ce but, il nous faut construire un outil d'analyse à partir de ces modèles.

Une première étape dans la construction d'un tel outil est de voir quelles nuances notre modèle épistémologique permet de distinguer. Développer une grille d'analyse en étudiant, tout d'abord, les conceptions des chercheurs interrogés peut être un premier pas dans cette direction. Les variations que nous avons repérées précédemment dans les entretiens doivent pouvoir être interprétées en termes de conception.

Les nuances que les conceptions doivent mettre en avant devront être mises en lien avec les différences entre les chercheurs, en particulier leur domaine de recherche. Cela peut apporter des informations pour comprendre les différentes transpositions qui peuvent être mises en œuvre dans différentes institutions enseignantes (par exemple, entre l'objet algorithme dans un enseignement d'informatique ou dans un enseignement de mathématiques).

Pour finir, l'analyse des conceptions des chercheurs à travers les entretiens peut aussi nous apporter des éléments concernant le rapport des enseignants au concept algorithme.

D'abord parce que l'on peut s'attendre à retrouver certaines conceptions de chercheurs chez les enseignants du secondaire. Ensuite parce que la grille d'entretien pourrait être ré-utilisée et adaptée pour étudier les conceptions des enseignants.

2 Méthodologie

2.1 Éléments pour l'analyse des conceptions

La distinction des différentes conceptions de l'algorithme doit s'appuyer sur les points suivants :

- définition de l'algorithme,
- forme de l'algorithme,
- rôle de l'algorithme,
- position sur l'exemple du discriminant,
- ASPECTS présents,
- présence des paradigmes AI, AM et PA, et des problèmes $\mathscr{P}_{\text{A}}$ et $\mathbb{P}$.

Dans le cas des entretiens, nous avons posé directement ces questions aux chercheurs mais des éléments de réponse peuvent se retrouver tout au long de la discussion. Nous choisissons de ne pas traiter différemment les réponses à ces questions directes et le reste du discours. Cela n'aurait pas de sens étant donné notre choix de faire des entretiens semi-ouverts.

Précisons que ce que nous allons décrire n'est qu'un ensemble de tendances. Il ne s'agit pas de caractériser les conceptions des chercheurs selon leur domaine mais de repérer des indicateurs susceptibles de guider la suite du travail, de proposer des éléments pour interpréter certaines conceptions et de donner des pistes pour poursuivre ces recherches.

Nous allons présenter ici les résultats, chercheur par chercheur, de l'analyse précédente de validation des modèles. Précisons d'abord la méthode d'étude des conceptions.

2.2 ASPECTS et paradigmes en jeu

Nous avons repéré chaque ASPECT selon les points mis en avant au chapitre 1. Pour chacun, nous précisons quels sont les points que nous considérons pour dire que l'aspect est fortement présent (••), présent (•) ou absent. Nous soulignerons lorsqu'un aspect n'est pas directement évoqué mais apparaît en filigrane dans un discours (·).

EFFECTIVITÉ Les points qui nous paraissent les plus importants sont la transmissibilité à un opérateur quelconque, la finitude et la non-ambiguïté. Ces points correspondent à une notion d'effectivité plus théorisée que, par exemple, la notion de programme informatique.

PROBLÈME Nous avons considéré que l'aspect est plus fortement présent lorsque la notion d'instance ou la notion d'entrée-sortie sont évoquées. La notion de résolution de problème (qui n'a pas nécessairement le sens que nous lui donnons ici) ou celle de tâche confiée à un algorithme sont trop floues pour être considérées comme suffisantes.

COMPLEXITÉ Nous considérons que la complexité est fortement présente si elle n'est pas simplement mentionnée dans le discours mais discutée un peu plus longuement. Par exemple, la présence de notions plus précises ou détaillées telles que la recherche d'algorithmes optimaux ou des critères d'étude de la complexité correspondent à cela.

PREUVE Nous avons considéré que la preuve était plus fortement impliquée quand l'algorithme est évoqué en tant qu'outil de preuve ou quand les preuves algorithmiques sont évoquées (en particulier la récurrence). D'autres points liés à la preuve comme la correction d'algorithmes nous ont paru moins forts.

MODÈLES THÉORIQUES Concernant les modèles théoriques, nous considérons qu'ils sont fortement présents s'ils ne sont pas simplement évoqués mais s'ils sont discutés ou si un rôle important leur est attribué (par exemple s'ils sont évoqués en réponse à la demande de définition de l'algorithme).

Paradigmes Nous utiliserons un codage plus simple pour la présence des paradigmes : soit le paradigme est évoqué (•), soit il est sous-entendu ou ce qui est évoqué est à la limite du paradigme (·), soit il est absent.

3 Les conceptions des chercheurs

3.1 Tableau récapitulatif

Le tableau suivant résume les aspects et paradigmes évoqués par les chercheurs ainsi que leur domaine de recherche [1]. Nous considèrerons que les conceptions-objet sont présentes si les aspects-objet sont présents.

Chercheur	Discipline	Section	EFFECTIVITÉ	PROBLÈME	COMPLEXITÉ	PREUVE	MOD. THÉO.	AI	AM	PA
1	MF, IA	27	••	••	••	••		•	•	•
2	MF, IF	27	••	••	••	•		•	•	·
3	MF	25	••	••		••		•	•	•
4	IF	27	••	••	••	••	•	•	•	•
5	IF	27	••	••	••	••		•	•	•
6	MF	26	••	•		••		•	·	•
7	MA, MF	26	•	••				•	•	
8	MF	25	••	••	•	••		•	•	•
9	MF, IA	25	••	•	••	•	••	•	•	
10	MF	25	••	••	••	••	••	•	•	•
11	MA	27	••	••	·	••	•	•	•	
12	MA	26	•	•	·			•	·	
13	MA	26	••	••	·	••		•	•	•
14	MF	25	••	••	••	••	••	•	•	•
15	MA	26	•	•				•	•	
16	IA	27	••	••	••	·	••	•	•	
17	MA	25	•	••		••		•	•	•
18	IF, MA	27	••	••	•	•	••	•	•	•
19	IA	26	•	•	•	•		•	•	•

1. M : Mathématiques, I : Informatique, F : Fondamentale(s), A : Appliquée(s). Nous utiliserons ces abréviations dans toute la suite de la section.

Ch.	Disc.	Sec.	EFF.	PROB.	COMP.	PR.	M. T.	AI	AM	PA
20	MF, IF	27	●●	●●	●●	●●	●●	●	●	●
21	MF	25	●●	●		●●		●	●	●
22	IA	27	●●	●●	●	●		●	●	

3.2 Résultats

Preuve

Il semble qu'il y ait un lien entre la présence de l'aspect PREUVE et le domaine de recherche. Sur les 11 chercheurs se déclarant en M.F., 10 abordent la preuve dont 8 de manière forte. Chez les chercheurs déclarés en I.F., tous abordent la preuve et 4 sur 5 de manière forte. Chez les chercheurs en M.A., seulement 4 sur 7 abordent la preuve dont seulement 3 de manière forte. Chez les chercheurs en I.A., 4 sur 5 abordent la preuve, mais parmi eux 3 n'abordent que la correction d'algorithme.

La preuve a tendance à être plus abordée (surtout de manière forte) par les chercheurs qui déclarent une partie de leur recherche comme théorique. Cela n'est pas étonnant mais se voit assez nettement dans nos résultats.

complexité

Deux phénomènes sont à remarquer concernant la complexité.
Le premier est que la complexité est abordée par tous les chercheurs déclarant mener des recherches en informatique. Ce constat n'a rien de surprenant mais confirme notre modèle dans lequel les questions de complexité sont plus faciles à exprimer dans les paradigmes AM et AI (en particulier avec les structures de données et les types de données abstraites).

L'autre phénomène est qu'un seul chercheur en M.A. évoque la complexité (c'est aussi un chercheur en I.F.). Pourtant, les chercheurs en M.F. citent tout de même souvent la complexité. Une interprétation possible est que les chercheurs en M.A. se focalisent plus sur la capacité d'approximation des algorithmes qu'ils utilisent que sur leur complexité. Une autre possibilité est que les méthodes en jeu ne sont pas tout à fait des algorithmes au sens où nous l'entendons, par exemple en ne résolvant pas une famille d'instances. Cela peut être soutenu par le fait que 2 de ces chercheurs en M.A. n'évoquent ni la notion d'instance, ni celle d'entrée-sortie.
Enfin, il est possible que la manipulation d'objets continus discrétisés, ne soit pas adaptée à l'étude de la complexité.
Toutes ces hypothèses demandent vérification et doivent être prises avec précautions.

Il faut noter que parmi les 6 chercheurs en M.F. qui citent la complexité, un seul n'est pas chercheur en informatique (et il déclare tout de même une activité de recherche de "loisir" en algorithmique). On peut donc affirmer que la complexité est très liée à l'informatique et bien moins aux mathématiques. Cela rejoint le point de vue de Knuth (1985) concernant l'importance de la complexité dans l'*algorithmic thinking* et son absence dans le *mathematic thinking*.

Modèles théoriques

Les modèles théoriques sont abordés par des chercheurs de tous les domaines. Cependant, si l'on regarde quelles sont les disciplines de ces chercheurs, on trouve, entre autres :

- Recherche opérationnelle et optimisation combinatoire (3 ch.)
- Théorie des nombres et cryptographie (1 ch.)
- Topologie algorithmique (1ch.)

Ces disciplines sont à cheval entre informatique et mathématiques et les modèles théoriques y jouent un rôle important. Les 2 chercheurs en M.F. qui évoquent ces modèles, avouent que cela ne fait pas partie de la culture des mathématiciens. Il semblerait que ces modèles théoriques soient peu connus en dehors des disciplines qui les utilisent.

Paradigmes

Tous les chercheurs évoquent le paradigme AI d'une manière très nette. Une grande majorité citent aussi AM ou y font référence. Il est difficile d'en dire plus concernant ces paradigmes, le questionnaire ne permettant pas facilement d'aborder les spécificités de AM.

Par contre, PA peut être facilement identifié, le questionnaire ayant était conçu en tenant compte de l'aspect PREUVE. Cet aspect coïncide avec les critères que nous avons donnés pour dire que la preuve est fortement présente.
PA est donc principalement évoqué par les chercheurs qui ont une recherche considérée comme théorique.

Concernant l'aspect EFFECTIVITÉ, les points que nous avons considérés comme moins forts relèvent plutôt de AI (programmation, algorithmes à destination des ordinateurs, méthodes automatiques, etc.). On peut considérer que les chercheurs qui n'évoquent que ces points pour l'aspect EFFECTIVITÉ se placent essentiellement dans le paradigme AI. Ce sont les mêmes chercheurs qui ne citent pas ou peu les aspects-objet. Ils sont dans la conception AI-outil.
Ces chercheurs sont essentiellement des chercheurs en M.A.

Lien aux sections

On peut aussi mettre en lien les réponses des chercheurs et leur section CNU. Il ne faut pas pousser trop loin les conclusions sur de tels effectifs, mais les liens précédents apparaissent encore plus clairement :

	COMPLEXITÉ	PREUVE	MOD. THÉO
25e	4/7	7/7	3/7
26e	1/6	3/6	0/6
27e	8/9	8/9	5/9

Conclusion

Résultats et pistes

Même si l'on ne peut pas les décrire en détail, on peut repérer globalement une grande hétérogénéité dans les conceptions des chercheurs. Il est fort probable que cette hétérogénéité soit aussi forte chez les enseignants.

Le domaine de recherche du chercheur semble jouer un rôle dans sa conception de l'algorithme. Ces premiers résultats tendent à valider différentes hypothèses que nous avions faites dans le modèle de μ-conceptions et dont nous avions discuté dans le chapitre 2. Notamment, la relation différente des mathématiques et de l'informatique à l'algorithme se retrouve ici.

Notons, enfin, que beaucoup de chercheurs chez qui l'on retrouve une conception large de l'algorithme (au sens où au plus un ASPECT est oublié), sont pour beaucoup des chercheurs du domaine de l'informatique mathématique et de domaines où les mathématiques discrètes sont très présentes.
Nous avions abordé au chapitre 2, la question de la relation privilégiée du discret à l'algorithme. Les résultats évoqués ici vont dans le sens d'un tel rapport : les mathématiques discrètes constituent un milieu favorable à l'algorithme.

Grille d'analyse

L'analyse des conceptions des chercheurs a montré la pertinence des modèles théoriques d'aspects et de μ-conceptions pour distinguer des nuances. Le choix des questions pour les entretiens à joué un rôle important pour repérer ces conceptions.

Les questions posées aux chercheurs peuvent donc servir de guide pour construire une grille d'analyse de conceptions. En particulier les questions sur la définition de l'algorithme, sur son rôle, les exemples choisis et leurs champs doivent être posées lors d'une telle analyse.

Les outils d'analyse que nous avons utilisés pour repérer les conceptions doivent aussi être intégrés dans une telle grille d'analyse. Notamment, les ASPECTS doivent être repérés, même si l'objectif est de décrire les conceptions : ils permettent de connaître les types de problèmes abordés et de repérer l'algorithme-outil et l'algorithme-objet.

Nous avons vu que les paradigmes peuvent être assez facilement repérés sur des critères de systèmes de représentation (la question de la reconnaissance des algorithmes apporte souvent des éléments). Il paraît utile de s'appuyer sur ces systèmes de représentation pour rechercher des conceptions.

Enfin, selon les ressources analysées ou les personnes interrogées, il faudra se poser la question des problèmes en jeu ($\mathscr{P}_{\text{A}}$? $\mathbb{P}$?), des opérateurs et des structures de contrôle.

Cela nous amène à proposer la grille d'analyse des conceptions suivante, qu'il faudra adapter à la situation et au sujet d'étude.

Grille générale d'analyse des conceptions

- Discours général sur l'algorithme et l'algorithmique :
  - Comment sont définis algorithme et algorithmique ?
  - Quel rôle est attribué à l'algorithme dans les mathématiques ?
  - Quelles tâches sont attendues de l'élève ?
- Algorithmes en jeu :
  - Quels problèmes sont proposés ?
  - Sont-ils des problèmes au sens de notre définition ? (instanciés ?)
  - Quels sont les algorithmes présents ?
  - Dans quels domaines sont ces algorithmes ?
  - Quels rôles jouent ces algorithmes dans l'activité ?
- Questionnement de l'algorithme :
  - Quelles questions sont à étudier concernant les algorithmes ?
  - Quelles discussions ont lieu autour de l'algorithmique ?
- Aspects :
  - Quels aspects sont évoqués ?
  - Relèvent-ils de l'outil ou de l'objet ?
- Conceptions
  - Dans quel paradigme se trouve-t-on ?
  - Quelles conceptions de l'algorithme sont présentes ? Et plus particulièrement : quels types de problèmes ? quels types d'opérateurs ? quels systèmes de représentation ? et quels types de structures de contrôle (si elles sont présentes) ?
  - Concernant les structures de contrôle, à qui revient la responsabilité de ces structures ?
  - Ces conceptions relèvent-elles de l'outil ou de l'objet ?

Des entretiens directifs

Les résultats obtenus sur les conceptions des chercheurs peuvent aider à mieux comprendre le rapport à l'algorithme selon les disciplines. Pour approfondir cette question, une étude plus systématique pourrait être menée. La validation des modèles étant réalisée, on pourrait maintenant proposer des entretiens directifs pour l'étude des conceptions de chercheurs, ou des questionnaires.

L'analyse menée ici, peut servir de pré-expérimentation, permettant de dire que la grille de questions est adaptée à l'étude des conceptions. Les questions directes sur l'algorithme du point de vue scientifique apportent des réponses riches pour l'étude des conceptions. Cependant, il ne faut pas négliger les questions basées sur l'enseignement, qui bien que soulevant régulièrement des échanges totalement en dehors du cadre de l'étude, ont permis de faire apparaître chez certains chercheurs des conceptions et des aspects qui n'avaient pas été évoqués.

Une méthodologie d'entretiens directifs peut permettre d'éviter l'écueil de discussions hors sujet à propos de l'enseignement. Un de nos objectifs est aussi de mieux connaître les conceptions des enseignants vis-à-vis de l'algorithmique. Une entrée par l'enseignement peut être un moyen adapté. Les questions sur l'algorithme du point de vue scientifique pouvant s'avérer trop directes pour interroger les enseignants.

Conclusion, retour sur les questions de recherche

Les entretiens menés nous ont permis de valider expérimentalement les modèles de la première partie. Cela constitue une réponse aux questions $Q4_{\text{aspects}}$ et $Q4_{\text{conceptions}}$.

Les entretiens ont aussi permis de proposer une analyse des conceptions des chercheurs en fonction de leur discipline de recherche. Cette étude nous a donné l'occasion de tester nos modèles comme outils pour l'analyse de conceptions. Les variations dans les conceptions des chercheurs confirment aussi nos analyses sur les rapports différents à l'algorithme en mathématiques et en informatique.
La connaissance partielle de l'algorithme chez les chercheurs interrogés soulève des questions concernant le rapport qu'entretiennent les enseignants du secondaire à l'algorithme.

Finalement, ce travail nous a amené à proposer une grille d'analyse adaptable à différentes situations, pour l'analyse des conceptions et celle de la transposition didactique. Nous mettrons cette grille à l'épreuve dans la partie suivante, en l'utilisant pour analyser les instructions officielles pour le lycée, des ressources proposées en ligne et des collections de manuels.

Troisième partie

L'algorithmique au lycée en France

Questions et problématiques pour cette partie

Nous nous intéressons ici à la transposition de l'algorithmique dans l'institution "Enseignement secondaire en France". Nous étudierons cette question à travers trois types de documents :

- les programmes et documents-ressources,
- des manuels,
- des ressources en ligne.

Les programmes nationaux donnent des indications sur la transposition du savoir savant vers le savoir à enseigner (par la noosphère décrite par Chevallard). Les consignes des programmes sont à voir comme une contrainte sur le savoir enseigné. Les documents-ressources (ou documents d'accompagnements) constituent une précision des attentes des programmes. En ce sens, ces documents sont conçus pour guider les enseignants dans la mise en place des instructions officielles. Étudier ces programmes et documents-ressources est indispensable à la compréhension de la transposition en jeu dans la classe [2].

La traduction de ces consignes dans les manuels constitue une deuxième étape importante de transposition, qui influence le savoir enseigné en classe. Les manuels constituent des ressources courantes dans l'usage des enseignants mais ce sont aussi des ressources au contact des élèves. Nous proposerons d'étudier plusieurs collections de manuels.

La question des ressources et de leurs usages est une problématique très vive en didactique des mathématiques (Gueudet & Trouche, 2010). Il nous semble pertinent de nous intéresser à d'autres types de ressources que les manuels, en particulier celles disponibles en ligne. Celles-ci se libèrent parfois des contraintes des programmes. Nous proposerons un corpus de telles ressources dont nous étudierons le rapport au concept d'algorithme [3].

Rappelons les questions sur la transposition au lycée que nous avions formulées dans la première partie :

Question $Q_{\text{lycée}}$. *Quelles conceptions sont représentées ? Les conceptions présentes sont-elles complètes ? Sont-elles modifiées par rapport à celles du savoir savant ? Relèvent-elles de l'outil ou de l'objet ? Quelle place est donnée aux structures de contrôle ?*

2. Il nous semble bon de préciser cela car les instructions officielles pourraient ne pas être pertinentes pour étudier le savoir enseigné dans d'autres contextes : autres pays, autres institutions d'enseignement, etc.

3. Nous ne nous intéresserons pas ici à la question (cruciale) de savoir comment de telles ressources sont utilisées et adaptées par les enseignants. Cependant, l'usage des ressources ne peut pas être étudié sans comprendre la transposition qui est en jeu.

Nous nous poserons ces questions dans les cas des questions de recherche que nous avions nommées :

$Q5_{conceptions}$ concernant les programmes de mathématiques du lycée, documents d'accompagnement et programmes de la spécialité ISN.

$Q6_{conceptions}$ concernant les manuels de mathématiques du lycée.

$Q7_{conceptions}$ concernant les ressources en ligne pour la formation et celles pour la classe.

Une autre question, transversale à cette partie, sera de valider les outils proposés et d'en faire des outils plus systématiques et facilement transférables. C'est cette question qui oriente l'organisation de cette partie.
Nous commencerons par l'étude des instructions officielles (chapitre 6).
Ensuite nous étudierons un corpus de ressources en lignes (chapitre 7) en essayant de systématiser les outils.
Pour terminer, nous développerons une analyse des manuels (chapitre 8) d'une manière plus systématique, permettant une approche quantitative.

Chapitre 6

Programmes et documents-ressources

Sommaire

1 Organisation des programmes 105
1.1 Le lycée en France 105
1.2 En mathématiques : des objectifs communs pour le lycée 106
1.3 Document ressource 107
2 Grille d'analyse 107
3 Analyse des programmes 108
3.1 Algorithmique dans l'en-tête des programmes 108
3.2 Objectifs pour le lycée 109
3.3 Les algorithmes présents dans les programmes 112
3.4 Conclusion pour les programmes de mathématiques 118
4 Algorithmique de la spécialité ISN 120
4.1 Discours 121
4.2 Contenus 122
5 Documents-ressources pour la seconde 124
5.1 Discussion sur l'algorithmique 125
♢ programme-papier 129
5.2 Activités proposées pour la classe 135
♢ algorithme-instancié 139
♢ programme de modélisation-simulation 142
5.3 Bilan 145
Conclusion 145

1 Organisation des programmes

1.1 Le lycée en France

Au sein des lycées, en France, ont lieu deux types d'enseignements. L'enseignement général et l'enseignement technologique. Nous ne nous intéresserons ici qu'aux mathématiques dans l'enseignement général.

La scolarité au lycée se décompose en trois années : la seconde (commune aux voies générale et technologique), les deux années suivantes (première et terminale) composent le cycle terminal pour lequel plusieurs filières d'enseignement général sont proposées :

– la série *économique et sociale* (ES),

– la série *littéraire* (L),
– la série *scientifique* (S).

En seconde, les mathématiques font partie des enseignements obligatoires (dits *communs*). En première et terminale ES, les mathématiques constituent un enseignement obligatoire (dit *spécifique* en terminale). En terminale ES un enseignement de spécialité en mathématique peut être choisi en supplément de l'enseignement spécifique.
En première L, les mathématiques sont un enseignement obligatoire dont le programme est le même que celui de la première ES. En terminale L, les mathématiques n'existent que dans un enseignement de spécialité (optionnel) dont le programme est le même que celui de l'enseignement spécifique de terminale ES.
En première et terminale S, les mathématiques constituent un enseignement obligatoire (dit *spécifique* en terminale). En terminale S, les élèves doivent choisir un enseignement dit de spécialité parmi plusieurs choix dont un enseignement de mathématiques. Depuis la rentrée 2012, certains établissements proposent aussi un enseignement de spécialité intitulé *Informatique et Sciences du Numérique* (ISN) qui contient une partie d'algorithmique.

Nous allons étudier les six documents officiels ci-dessous qui déterminent ces programmes :

– Programme de mathématiques de seconde (B.O. spécial n°30 du 23 juillet 2009)
– Programme de mathématiques de première ES et L (B.O. spécial n° 9 du 30 septembre 2010)
– Programme de mathématiques de première S (B.O. spécial n° 9 du 30 septembre 2010)
– Programme de mathématiques de terminale ES (spécifique et de spécialité) et L (spécialité) (B.O. spécial n°8 du 13 octobre 2011)
– Programme de mathématiques de terminale S (spécifique et de spécialité) (B.O. spécial n°8 du 13 octobre 2011)
– Programme de l'enseignement de spécialité Informatique et Sciences du Numérique (B.O. spécial n°8 du 13 octobre 2011)

1.2 En mathématiques : des objectifs communs pour le lycée

Il faut noter que les programmes de mathématiques du lycée ont une spécificité concernant l'algorithmique. Les objectifs des programmes de mathématiques sont donnés pour l'ensemble du lycée et sont donc les mêmes dans tous les programmes. Chaque programme précise ensuite quels algorithmes sont attendus en fonction des contenus. Nous reproduisons ici ce paragraphe commun que nous étudierons en détail par la suite. Seule l'introduction change entre la seconde et l'ensemble des séries du cycle terminal.

Le paragraphe *Algorithmique* dans les programmes de mathématiques du lycée :
Introduction dans le programme de seconde :

La démarche algorithmique est, depuis les origines, une composante essentielle de l'activité mathématique. Au collège, les élèves ont rencontré des algorithmes (algorithmes opératoires, algorithme des différences, algorithme d'Euclide, algorithmes de construction en géométrie). Ce qui est proposé dans le programme est une formalisation en langage naturel propre à donner lieu à traduction sur une calculatrice ou à l'aide d'un logiciel. Il s'agit de familiariser les élèves avec les grands principes d'organisation d'un algorithme : gestion des entrées-sorties, affectation d'une valeur et mise en forme d'un calcul.

Introduction dans les programmes du cycle terminal :

En seconde, les élèves ont conçu et mis en œuvre quelques algorithmes. Cette formation se poursuit tout au long du cycle terminal.

Contenu commun à l'ensemble du lycée :

Dans le cadre de cette activité algorithmique, les élèves sont entraînés à :

- décrire certains algorithmes en langage naturel ou dans un langage symbolique ;
- en réaliser quelques-uns à l'aide d'un tableur ou d'un programme sur calculatrice ou avec un logiciel adapté ;
- interpréter des algorithmes plus complexes.

Aucun langage, aucun logiciel n'est imposé.

L'algorithmique a une place naturelle dans tous les champs des mathématiques et les problèmes posés doivent être en relation avec les autres parties du programme (analyse, géométrie, statistiques et probabilités, logique) mais aussi avec les autres disciplines ou le traitement de problèmes concrets.
À l'occasion de l'écriture d'algorithmes et de programmes, il convient de donner aux élèves de bonnes habitudes de rigueur et de les entraîner aux pratiques systématiques de vérification et de contrôle.

Instructions élémentaires (affectation, calcul, entrée, sortie)
Les élèves, dans le cadre d'une résolution de problèmes, doivent être capables : • d'écrire une formule permettant un calcul ; • d'écrire un programme calculant et donnant la valeur d'une fonction, ainsi que les instructions d'entrées et sorties nécessaires au traitement.
Boucle et itérateur, instruction conditionnelle
Les élèves, dans le cadre d'une résolution de problèmes, doivent être capables de : • programmer un calcul itératif, le nombre d'itérations étant donné ; • programmer une instruction conditionnelle, un calcul itératif, avec une fin de boucle conditionnelle.

1.3 Document ressource

Lors de l'entrée dans les programmes de seconde, en 2009, de l'algorithmique, un document d'accompagnement intitulé *Ressources pour la classe de seconde* spécifique à l'algorithmique en seconde a été mis à disposition des enseignants. Ce document complète les nouveaux programmes de seconde et développe de nombreux exemples plus explicitement. Il est disponible à l'adresse `eduscol.education.fr/D0015`. Nous étudierons aussi son contenu.

2 Grille d'analyse

Nous analyserons les différents programmes selon la grille suivante [1], qui se base sur les résultats de la première partie :

- Discours général sur l'algorithme et l'algorithmique :
  - Comment sont définis algorithme et algorithmique ?
  - Quel rôle est attribué à l'algorithme dans les mathématiques ?
  - Quelles tâches sont attendues de l'élève ?
- Algorithmes en jeu :
  - Quels problèmes sont proposés ?
  - Sont-ils des problèmes au sens de notre définition ? (instanciés ?)
  - Quels sont les algorithmes présents ?
  - Dans quels domaines sont ces algorithmes ?
  - Quels rôles jouent ces algorithmes dans l'activité ?
- Questionnement de l'algorithme :

1. Plusieurs des questions de cette grille se recoupent ou sont directement liées. Cela ne gêne en aucun cas son utilisation. Au contraire cela permet une certaine souplesse pour réaliser l'analyse. Nous ne détaillerons pas les liens qui existent entre ces questions qui découlent directement de nos discussions de la partie 1.

  - Quelles questions sont à étudier concernant les algorithmes ?
  - Quelles discussions ont lieu autour de l'algorithmique ?
- Aspects :
  - Quels aspects sont évoqués ?
  - Relèvent-ils de l'outil ou de l'objet ?
- Conceptions
  - Dans quel paradigme se trouve-t-on ?
  - Quelles conceptions de l'algorithme sont présentes ? Et plus particulièrement : quels types de problèmes ? quels types d'opérateurs ? quels systèmes de représentation ? et quels types de structures de contrôle (si elles sont présentes) ?
  - Concernant les structures de contrôle, à qui revient la responsabilité de ces structures ?
  - Ces conceptions relèvent-elles de l'outil ou de l'objet ?

3 Analyse des programmes

Pour plus de lisibilité, dans cette section, nous utiliserons l'italique pour les citations extraites du programme (et uniquement pour cela).

3.1 Algorithmique dans l'en-tête des programmes

En seconde, l'objectif général du programme est *de former les élèves à la démarche scientifique sous toutes ses formes*. Cela pour les rendre capables de plusieurs types d'activités dont l'un est de *pratiquer une activité expérimentale ou algorithmique*. Le choix d'associer algorithmique et expérimental montre que l'algorithme est vu plutôt comme un élément lié à la démarche expérimentale. On se place clairement du côté outil de l'algorithmique.
Quant à la *diversité des activités de l'élève*, elle comprend une activité *appliquer des techniques et mettre en œuvre des algorithmes*. Cette mise au même niveau de l'application de techniques et de la mise en œuvre d'algorithmes donne un sens particulier au terme *mise en œuvre*. Il prend le sens d'appliquer un algorithme dans un cas particulier, de résoudre une instance particulière d'un problème. C'est la conception outil.
Dans l'ensemble des programmes de cycle terminal, la mise en œuvre d'algorithmes n'est plus associée à l'application de techniques mais le sens de mise en œuvre doit surement être compris de la même façon.

Les programmes de toutes les sections indiquent que les capacités attendues dans le domaine de l'algorithmique sont présentées dans un paragraphe en fin de programme (c'est ce paragraphe que nous étudions après) mais *doivent être exercées à l'intérieur de chaque champ du programme*. Le programme de seconde spécifie même qu'*elles sont transversales*.

Les introductions des programmes présentent donc l'algorithmique, non comme un champ des mathématiques, mais comme un type d'activités ou une démarche qui doit être mise en œuvre dans tous les chapitres. Les algorithmes, eux, ont pour rôle de permettre leur application sur des instances spécifiques. On a pour l'instant une vision restreinte de l'algorithmique qui est axée sur l'aspect outil : outil pour la démarche expérimentale, outil pour l'application pratique sur des cas particuliers.

3.2 Objectifs pour le lycée

Concernant les objectifs communs à l'ensemble des programmes de mathématiques du lycée général, nous suivrons une analyse linéaire du contenu. Nous relèverons au fur et à mesure les éléments concernant la grille d'analyse.

Activité de l'élève

Les élèves doivent être *entrainés*[2] *à décrire certains algorithmes en langage naturel ou symbolique*. Ce que sont ces deux langages n'est pas très clair. On comprend que le langage *naturel* se réfère à une description moins formalisée d'un algorithme, mais on peut l'interpréter comme étant la langue française ou, si l'on considère que l'on est en classe de mathématique, le langage mathématique (ou un mélange des deux). Le langage *symbolique* fait référence à un langage plus spécifique, plus formalisé – qui peut être compris comme le langage mathématique, un pseudo-code ou encore un langage de programmation – mais dont la nature reste ambigüe. Cette phrase fait néanmoins référence à une activité de l'ordre de la gestion de plusieurs niveaux de langage, qui correspondraient aux systèmes de représentation des conceptions de AM-outil et AI-outil.

Les élèves doivent aussi être *entraînés à réaliser quelques* algorithmes *à l'aide d'un tableur ou d'un programme*. Le choix du terme *réaliser* nous paraît lui aussi ambigu, ce n'est ni un terme usuel de l'algorithmique ni de la programmation. Cependant le contexte nous laisse penser qu'il s'agit de traduire un algorithme dans un langage de programmation, ou dans un tableur, dans le but de son exécution. Cela fait référence à un travail de l'ordre du passage d'un langage à un autre. C'est une activité d'implémentation (et d'exécution) qui se réfère à l'aspect outil.

Enfin, les élèves sont *entraînés à interpréter des algorithmes plus complexes*. Le terme *interpréter* fait référence à une activité qui porte sur le langage. Il s'agit apparemment de donner du sens à un algorithme écrit dans un certain langage. Cependant, le programme n'explicite pas l'activité précise que cela représente. S'agit-il de comprendre et d'être capable d'appliquer chaque étape de l'algorithme (en particulier sur une instance donnée) ? Dans ce cas on se place complètement du côté outil : il s'agit de savoir utiliser un algorithme qu'on nous donne. C'est un point de l'aspect effectif. S'agit-il de pouvoir dire quel est le problème ou la question auquel répond l'algorithme ? Et dans ce cas, quelle validation est attendue de cette "interprétation" ? Aucun opérateur ni structure de contrôle ne sont évoqués pour ce type de questions portant sur les algorithmes. Nous considérons donc qu'il ne s'agit pas vraiment d'un problème où l'algorithme est objet.

Nous ne commenterons pas le fait qu'aucun langage ni logiciel ne soit imposé. Bien que certains logiciels et langages soient surement plus adaptés que d'autres à l'apprentissage de l'algorithmique (au sens que nous lui avons donné en partie 1). Cela dépasse le cadre de notre analyse.

Place dans les mathématiques

Le choix de placer l'algorithmique comme une section du programme à part[3] et transversale aux autres parties est expliqué ici. *L'algorithmique a une place naturelle dans tous*

2. Le choix du terme est étonnant
3. Tout comme le raisonnement.

les champs des mathématiques et les problèmes posés doivent être en relation avec les autres parties du programme. Le fait que l'algorithme soit une méthode de résolution de problèmes qui s'applique dans divers champs des mathématiques est importante mais l'affirmation faite ici demanderait un peu plus de précisions et en particulier sur ce que signifie *naturel*. Les parties 1 et 2 ont montré que la place de l'algorithme varie selon les domaines des mathématiques et laisse penser que certains sont plus propices à une activité algorithmique que d'autres (en particulier pour l'algorithme-objet). Il nous faudra voir si tous les chapitres font l'objet d'activités algorithmiques aussi naturellement les uns que les autres dans les programmes spécifiques à chaque classe.

Les problèmes posés doivent être en relation *aussi avec les autres disciplines ou le traitement de problèmes concrets.* On voit ici que l'aspect problème est mis au centre des activités autour de l'algorithmique. Cependant l'interaction entre l'algorithmique et *les autres disciplines* ou l'algorithmique et *les problèmes concrets* n'est pas plus détaillée. Il nous faudra regarder comment ces demandes sont interprétées dans les documents d'accompagnement et les manuels.

Rôle de l'algorithmique pour l'apprentissage

Un des objectifs affichés de l'activité algorithmique dans les programmes est l'apprentissage de la *rigueur* : *Au cours de l'écriture d'algorithmes et de programmes, il convient de donner aux élèves de bonnes habitudes de rigueur et aux pratiques systématiques de vérification et de contrôle.*
Tout d'abord notons que la référence à l'algorithmique est encore une fois faite par le biais de l'*écriture*, c'est un travail sur le langage qui est évoqué. Ce travail doit mener à une certaine rigueur, ce que l'on peut interpréter comme un effort de produire des algorithmes non-ambigus (c'est l'aspect effectif) et des programmes qui soient "compris" par la machine. Associé aux *pratiques systématiques de vérification et de contrôle*, le sens devient plus flou. Le terme *vérification et contrôle* fait-il référence à la preuve, à la justification de l'algorithme ou au respect des contraintes d'un langage donné (plus ou moins proche d'un langage de programmation) ? Étant donné le contexte du début de la phrase, il semble que ce soit une rigueur vis-à-vis du respect du langage. Les *pratiques systématiques de vérification et de contrôle* feraient alors référence à une bonne organisation dans ce travail d'écriture ou encore une vérification que l'algorithme ou le programme soit conforme aux exigences du problème traité.
Quoi qu'il en soit, on se situe dans les conceptions AM-outil et AI-outil et principalement sur les questions de système de représentation et de passage de l'un à l'autre. Des structures de contrôle sont évoquées mais il est impossible ici d'en dire la nature. Notre hypothèse est qu'il ne s'agit pas de preuve mais plus de respect de contraintes des langages.

Capacités attendues

La suite est présentée sous forme d'un tableau qui présente les notions à aborder et les capacités attendues des élèves. On peut penser que ce seront ces capacités qui seront évaluées et qui seront principalement travaillées. Dans les autres parties du programme, les notions à enseigner et les compétences exigibles sont celles qui sont décrites dans les tableaux.

Ces capacités se découpent en deux : Les *instructions élémentaires (affectation, calcul, entrée, sortie)* et les *boucles*, *itérateurs* et *instructions conditionnelles*. Dans les deux cas, cela fait référence à des notions liées aux outils de programmation. C'est l'apprentissage

des principales structures de programmation qui est en jeu. Ces structures ne sont pas uniquement présentes dans les langages de programmation mais sont des outils utilisés pour formuler des algorithmes de manière très proche des langages de programmation (souvent dans le but de parler de programmation sans entrer dans un langage trop spécifique). On peut considérer que l'on se situe surtout du côté de l'Algorithme Informatique de notre modèle.

Dans les deux parties du tableau, les *capacités attendues* de l'élève se situent *dans le cadre d'une résolution de problèmes*. L'aspect problème est encore mis en avant ici.

Les deux *capacités attendues* concernant les instructions élémentaires sont : - *écrire une formule permettant un calcul*. Il s'agit encore d'une activité d'écriture dont l'objectif est le passage d'une formule à un programme (sur machine ou sur papier). On peut penser qu'il s'agit de passer de la formule à son découpage en instructions élémentaires issues de la programmation.
- *écrire un programme calculant et donnant la valeur d'une fonction, ainsi que les instructions d'entrée et sortie nécessaires au traitement.* Il s'agit encore plus clairement ici d'une activité d'implémentation dans un certain langage de programmation. Le but semble aussi de faire entrer une fonction dans le cadre des instructions élémentaires en spécifiant *entrée*, *sortie* et *traitement* (qui fait référence à l'ensemble des instructions menant de l'entrée au calcul de la sortie).
Les deux *capacités attendues* concernant boucle, itérateur et instruction conditionnelle sont : - *Programmer un calcul itératif, le nombre d'étapes étant donné.*
Il s'agit ici de programmer (donc d'utiliser un langage de programmation, on est dans AI) la boucle "FOR".
- *Programmer une instruction conditionnelle, un calcul itératif avec une fin de boucle conditionnelle.*
Toujours dans AI, il s'agit de programmer avec les instructions "IF THEN ELSE" et "WHILE".
Toutes ces *capacités attendues* font référence à des tâches de programmation. On se situe complétement dans AI et les attentes sont de l'ordre de la capacité à exprimer formules, fonctions et résolutions de problèmes, dans les contraintes des langages de programmation. C'est l'aspect effectif qui est mis en avant.

En résumé

Pour résumer les points précédents concernant le programme d'algorithmique au lycée :

- Il nous semble important, tout d'abord, de mettre en avant l'ambiguïté des termes utilisés (qui ne sont ni définis, ni ne font référence à des termes canoniques ou usuels de l'algorithmique). Il ne nous semble pas que ce soit le cas dans les autres parties des programmes. En particulier, les termes algorithme et programme ne sont pas clairement distingués et semblent même utilisés comme des équivalents.
- Les capacités attendues et les activités proposées sont axées sur le langage et sont centrées autour de la programmation et de l'implémentation de méthode de résolution.
- Les aspects présents sont seulement la résolution de problème et l'effectivité. On se situe uniquement du côté outil. Les questions de preuve ou de complexité ne sont jamais évoquées.

- La conception globale des programmes semble donc axée sur AM et surtout sur AI. L'importance est mise sur les systèmes de représentation et la reformulation dans AI. La conception PM n'est jamais évoquée. Aucune structure de contrôle n'est décrite, mise à part le respect des contraintes des langages utilisés.

Regardons maintenant les spécificités (et notamment les algorithmes) présents dans les programmes de chaque classe.

3.3 Les algorithmes présents dans les programmes

Dans les chapitres du programme de chaque classe, les contenus sont organisés selon un tableau à trois colonnes : *contenus*, *capacités attendues* et *commentaires*. Nous reprenons ces termes dans nos analyses sans les commenter. De plus, nous préciserons à chaque fois dans quelle colonne on se trouve. En effet, une *capacité attendue* autour de l'algorithme n'a pas le même statut qu'un *commentaire* évoquant une activité algorithmique possible. Nous suivrons le découpage des chapitres de chaque programme. Nous commenterons et analyserons chaque référence faite à l'algorithme.

Seconde

L'analyse des algorithmes et activités algorithmiques présents dans les différents chapitres de seconde est présentée dans le tableau suivant :

Extrait du programme	Analyse et commentaires
Fonctions	
Étude qualitative des fonctions *Commentaire : Même si les logiciels traceurs de courbes permettent d'obtenir rapidement la représentation graphique d'une fonction définie par une formule algébrique, il est intéressant, notamment pour les fonctions définies par morceaux, de faire écrire aux élèves un algorithme de tracé de courbe.*	Il est clair qu'ici l'objectif de l'algorithme est de pouvoir tracer la courbe, il s'agit donc d'écrire l'algorithme dans un environnement de programmation. On ne sait pas vraiment s'il s'agit de travailler sur le calcul des valeurs point par point de la fonction pour en faire tracer la courbe avec une bonne résolution ou si c'est simplement la disjonction de cas liée à la fonction définie par morceaux (et qui implique l'utilisation d'instructions conditionnelles) qui doit être soulevée. Faire tracer une courbe étant donnée sa formule n'est pas un algorithme très riche pour en faire un objet d'étude étant donné qu'il répond à une problématique pratique (faire tracer une courbe sur un écran) plutôt qu'à un problème mathématique. C'est l'aspect outil qui est en jeu.
Équations, résolution graphique et algébrique	

Capacités attendues : *Encadrer une racine d'une équation grâce à un algorithme de dichotomie*	On ne sait pas bien si la capacité concerne la conception de l'algorithme ou la résolution de l'équation. Cependant, un problème précis et une méthode algorithmique sont évoqués. Aucune capacité ou question n'est en jeu concernant cet algorithme lui-même alors que la dichotomie a un potentiel pour être objet de questions (efficacité, optimalité, etc).
Géométrie	
Configurations du plan Commentaire : *Le cadre de la géométrie repérée offre la possibilité de traduire numériquement des propriétés géométriques et permet de résoudre certains problèmes par la mise en œuvre d'algorithmes simples.*	Il va surement s'agir d'algorithmes pour tester certaines propriétés étant données les coordonnées de points, ou déterminer les coordonnées de points spécifiques (milieu, intersection de deux droites), etc. Tous ces éléments se rapportent à la capacité de traduction de formules en algorithmes ou en programmes. Comme nous l'avons dit précédemment, cela s'apparente à l'aspect outil et à la conception AI.
Statistiques et probabilités	
Réalisation d'une simulation Commentaire : *À l'occasion de la mise en place d'une simulation on peut [...] mettre en place des instructions conditionnelles dans un algorithme.*	La simulation est présentée comme une activité permettant d'aborder un certain type d'instruction. On se situe dans AI, du côté outil. Il s'agit de faire apprendre la programmation.
Parmi les objectifs visés en statistiques et probabilités, dans le cadre de résolutions de problèmes : *La répétition d'expériences aléatoires peut donner lieu à l'écriture d'algorithmes (marches aléatoires).*	Il s'agit d'une activité d'écriture d'algorithmes qui est motivée par la répétition un grand nombre de fois d'une même expérience. C'est l'aspect effectif, la possibilité de réaliser ce grand nombre d'expériences, qui motive l'algorithme ici (qui sera certainement exprimé dans AI). On ne sait pas trop quel rôle joue l'algorithme dans la démarche.

Première ES-L

L'analyse des algorithmes et activités algorithmiques présents dans les différents chapitres de première ES et L est présentée dans le tableau suivant :

Extrait du programme	Analyse et commentaires
Algèbre et analyse	
Second degré (polynômes)	

Commentaires : *Des activités algorithmiques sont réalisées dans ce cadre.*	L'algorithmique doit être abordée dans ce cadre mais aucune précision n'est donnée. Vue les attentes concernant la manipulation de polynômes du second degré, il va surement s'agir d'implémenter, de rendre effectives certaines formules et opérations formelles. On peut penser que l'objectif ici est de rendre plus accessibles les objets en les programmant, peu importe l'algorithme ou le problème traité.
Suites Capacité attendue : *Mettre en œuvre un algorithme permettant de calculer un terme de rang donné.*	Selon le mode de génération de la suite étudiée il va s'agir de programmer une formule ou de répéter (avec une boucle "FOR") la relation de récurrence définissant la suite. Il s'agit donc essentiellement d'un travail de traduction en langage de programmation. La motivation semble être le calcul effectif de termes de la suite ou la compréhension des différents modes de génération de suites (comme le commentaire suivant le laisse penser).
Suites Commentaire : *L'utilisation du tableur et la mise en œuvre d'algorithmes sont l'occasion d'étudier en particulier des suites générées par une relation de récurrence.*	L'algorithme est ici présenté comme un outil (au même titre que le tableur) pour étudier le comportement des suites, un outil d'étude mais aussi un outil pour la compréhension de ce type de suites. Il s'agit encore clairement d'une activité de programmation, donc de AI-outil.
Suites Commentaire : *On peut utiliser un algorithme ou un tableur pour traiter des problèmes de comparaison d'évolutions, de seuils et de taux moyen.*	Il s'agit d'utiliser l'algorithme (sous forme de programme) ou le tableur comme outil pour une démarche expérimentale avec les suites, en implémentant certaines formules. On est du côté outil et l'aspect effectif laisse penser que l'on se place dans AI.
Statistiques et probabilités	
Probabilités Commentaire : *On peut simuler la loi binomiale avec un algorithme.*	Il s'agit d'implémenter (AI) une répétition d'expériences aléatoires identiques. L'algorithme est ici outil.
Échantillonnage Commentaire : *L'intervalle de fluctuation peut être déterminé à l'aide d'un tableur ou d'un algorithme.*	Il s'agit simplement de mettre en application la formule de l'intervalle de fluctuation, puisque le tableur fait aussi bien l'affaire que l'algorithme.

Terminale ES-L

L'analyse des algorithmes et activités algorithmiques présents dans les différents chapitres de terminale ES et L est présentée dans le tableau suivant :

Extrait du programme	Analyse et commentaires
Analyse	
Suites (géométriques) Capacité attendue : *Étant donné une suite* (q^n) *avec* $0 < q < 1$, *mettre en œuvre un algorithme permettant de déterminer un seuil à partir duquel* q^n *est inférieur à un réel a positif donné.*	Une capacité précise est attendue. Malgré l'ambiguïté du terme *mettre en œuvre*, un problème est évoqué clairement avec ses instances et sa question. Cependant, on peut penser qu'il s'agira essentiellement de programmer une boucle "WHILE" en implémentant le calcul effectif des termes de la suite. Des questions sur l'algorithme lui-même (et ses variantes) auraient pu être posées : Combien d'étapes sont nécessaires (en fonction de q) ? Peut-on trouver un algorithme plus efficace que d'autres ?
Enseignement de spécialité, série ES	
Les thèmes abordés sont particulièrement propices à l'utilisation des outils informatiques (logiciels de calcul, tableur) et à la mise en œuvre d'algorithmes.	Le programme n'est pas très explicite sur les algorithmes pouvant être en jeu alors que les graphes et les matrices sont des objets privilégiés pour l'algorithmique et notamment l'algorithme en tant qu'objet.

Première S

L'analyse des algorithmes et activités algorithmiques présents dans les différents chapitres de première S est présentée dans le tableau suivant :

Extrait du programme	Analyse et commentaires
Analyse	
Second degré (polynômes) Commentaire : *Des activités algorithmiques doivent être réalisées dans ce cadre.*	Cf première ES-L.
Suites Capacité attendue : *Mettre en œuvre des algorithmes permettant : - d'obtenir une liste de termes d'une suite ; - de calculer un terme de rang donné.*	Cf première ES-L.
Suites Commentaire : *L'utilisation du tableur et la mise en œuvre d'algorithmes sont l'occasion d'étudier en particulier des suites générées par une relation de récurrence.*	Cf première ES-L.
Suites	

Commentaire : *On peut utiliser un algorithme ou un tableur pour traiter des problèmes de comparaison, d'évolution et de seuils.*	Cf première ES-L.
Statistiques et probabilités	
Probabilités Commentaire : *On peut simuler la loi géométrique tronquée avec un algorithme.*	Il s'agit de répéter une expérience aléatoire un grand nombre de fois. On se place dans le cadre de l'algorithme outil pour une démarche expérimentale ou pour rendre effectif un objet. Il s'agit donc de programmer, on est dans AI.
Probabilités Commentaire : *On peut simuler la loi binomiale avec un algorithme.*	Cf première ES-L.
Échantillonnage Commentaire : *L'intervalle de fluctuation peut être déterminé à l'aide d'un tableur ou d'un algorithme.*	Cf première ES-L.

Terminale S

L'analyse des algorithmes et activités algorithmiques présents dans les différents chapitres de terminale S est présentée dans le tableau suivant :

Extrait du programme	Analyse et commentaires
Analyse	
Suites Capacité attendue : *Dans le cas d'une limite infinie, étant donnés une suite croissante (u_n) et un nombre réel A, déterminer à l'aide d'un algorithme un rang à partir duquel u_n est supérieur à A.*	On est dans le même cas qu'en terminale ES-L pour les suites (q^n). Le problème est clairement défini mais l'algorithme n'est qu'outil alors qu'il pourrait être objet.
Suites Commentaire : *Des activités algorithmiques sont menées dans ce cadre.*	Ces activités ne sont pas détaillées mais sont de l'ordre de toutes les activités sur les suites des autres programmes.
Continuité sur un intervalle, théorème des valeurs intermédiaires	

Commentaire : *Des activités algorithmiques sont réalisées dans le cadre de la recherche de solutions de l'équation $f(x) = k$.*	Le programme ne dit pas quelles activités. On peut penser aux approches balayage et dichotomie qui peuvent permettre de traiter l'algorithme comme objet mais ni la preuve ni la complexité ne sont évoquées. On restera très certainement dans l'outil et du côté AI.
Intégration Capacité attendue : *Pour une fonction monotone positive, mettre en œuvre un algorithme pour déterminer un encadrement d'une intégrale.*	Le programme ne précise pas quelles sont ces activités algorithmiques mais un problème précis est présenté. La capacité attendue est encore de *mettre en œuvre*, l'activité sera centrée sur la description et l'implémentation de la méthode. On reste sur l'aspect effectif et outil de l'algorithme alors que des questions sur l'algorithme pourraient être proposées : complexité, erreur, comparaison d'algorithmes, validité, etc.
Probabilités et statistique	
Conditionnement, indépendance Commentaire : *Des activités algorithmiques sont menées dans ce cadre, notamment pour simuler une marche aléatoire.*	On peut s'attendre à des activités de simulation et d'utilisation de programme pour une démarche expérimentale comme c'est le cas en probabilités et statistique dans les autres programmes.
Enseignement de spécialité	
Les thèmes abordés sont particulièrement propices à l'utilisation des outils informatiques (logiciels de calcul, tableur) et à la mise en œuvre d'algorithmes.	Tout comme pour la spécialité de terminale ES, on peut regretter le peu de précisions données. D'autant plus que les domaines au programme (arithmétique et matrices) sont riches pour l'algorithmique et notamment l'algorithme-objet.

Bilan

Résumons nos constats pour les contenus spécifiques à chaque classe :

- Seulement 6 *capacités attendues* concernent l'algorithmique sur l'ensemble des programmes. Pour ces capacités seulement le type de problème à savoir résoudre avec un algorithme est précisé.
- Les algorithmes ne semblent pas trouver une place si *naturelle* dans tous les champs comme l'annonçait le paragraphe commun à l'ensemble du lycée. Ce sont toujours les mêmes types de problématiques qui sont présents (simulation en probabilités et statistiques, résolution d'équations en analyse, etc.) et plusieurs parties du programme n'ont aucune remarque concernant l'algorithmique (géométrie dans l'espace, fonctions en première ES-L, probabilités et statistiques ainsi qu'une grande partie de l'analyse en terminale ES-L, géométrie en première S, géométrie ainsi qu'une grande partie des probabilités et statistique de terminale S). Pour beaucoup d'autres parties du programme il s'agit souvent de préciser qu'une notion est propice à une activité algorithmique sans plus de précisions.
- On recense seulement trois algorithmes "riches" (au sens où ils ne sont pas des formules à implémenter ou des traductions directes du problème et pourraient être

objets d'étude) : l'algorithme de dichotomie pour trouver la racine d'une équation (en seconde), la recherche de solution à une équation $f(x) = k$ et l'encadrement d'intégrales (tous les deux en Terminale S). Ces algorithmes appartiennent tous au domaine de l'approximation en analyse.

- La mise en œuvre effective semble être une des priorités, via l'implémentation de tous les algorithmes en jeu dont très peu sont plus qu'une traduction en langage de programmation de formules (polynômes, suites, fonctions) ou une répétition d'un calcul (suites, simulations).
- Les algorithmes en jeu sont majoritairement des algorithmes d'approximation ou de simulation (si l'on exclut les implémentations de formules).
- Le paradigme AI est très majoritairement présent.
- L'algorithme n'est l'objet d'aucune question ou étude et semble ne pas être un objet mathématique mais un moyen, par le biais de la programmation de :
  - rendre effectif les mathématiques par l'obtention de résultats concrets,
  - aider à comprendre les objets par leur implémentation,
  - comprendre ce que fait la calculatrice ou certains logiciels ,
  - permettre une activité expérimentale.
- Certains domaines sont pointés comme permettant de faire découvrir l'algorithmique par la possibilité de rencontrer les structures de programmation usuelles (configurations du plan, simulation, polynômes de degré 2, etc.)

3.4 Conclusion pour les programmes de mathématiques

Nous présentons nos conclusions en suivant les grands points de la grille d'analyse.

Présentation générale de l'algorithme et de l'algorithmique

Tout d'abord il faut noter que l'algorithme n'est pas défini mais seulement illustré et expliqué au travers d'exemples. Cela ne contribue pas à donner à l'algorithme le statut d'objet mathématique. Cela peut aussi contribuer à un amalgame entre programme et algorithme car la différence entre les deux n'est pas explicitée et l'utilisation des deux termes ne semble pas non plus liée à une différentiation.
L'algorithmique est présentée comme un type d'activité apparenté à la démarche expérimentale. Cette activité est souvent mise sur le même plan que l'utilisation de logiciels. Elle est souvent évoquée comme relevant d'une activité au niveau du langage.

Dans l'activité, les algorithmes ont pour rôle de permettre la mise en œuvre pratique des résultats mathématiques et l'implémentation de certains objets d'une part, et d'autre part de contribuer à l'expérimental en permettant notamment la simulation de processus.

Pour l'apprentissage des mathématiques, il semble que ce soit la possibilité d'implémenter les objets mathématiques qui soit au centre.

Les compétences attendues de l'élève sont de l'ordre de la programmation (implémentation) et de la maîtrise d'un langage de programmation (à travers la maîtrise des structures de programmation usuelles) et d'un langage dit naturel sur lequel on n'a pas beaucoup de précisions.

Algorithmes en jeu

Sur l'ensemble du cursus, peu d'algorithmes sont présentés au final. Nous pouvons résumer les types d'algorithmes ou les thèmes d'activités algorithmiques proposés ici :

- application de formules géométriques,
- simulation et répétition d'expériences aléatoires,
- calcul de termes de suites, recherche d'un terme vérifiant une propriété et étude expérimentale de suites,
- calcul de valeurs particulières d'une série statistique,
- résolution d'équations et approximation en analyse,
- opérations sur les polynômes du second degré.

Parmi ces algorithmes, peu sont assez riches pour permettre d'en faire des objets de questionnement (et cela n'est jamais évoqué). Les algorithmes concernés sont des algorithmes d'approximation en analyse (l'un en seconde et les deux autres en terminale S).

Les algorithmes sont tout d'abord annoncés comme ayant une place naturelle dans tous les champs puis on constate que peu d'exemples sont finalement donnés dans les contenus (et l'on parle plus souvent de thématiques propices à des activités algorithmiques). Certains chapitres ne contiennent aucun algorithme et l'on retrouve dans les autres toujours le même type d'algorithmes.
Notre hypothèse est que les algorithmes ne jouent pas le même rôle partout, dans certains champs ils sont plus fondamentaux ou présentent une plus grande richesse pour leur étude alors que dans d'autres ils sont plus souvent des outils.

Les algorithmes en jeu sont pour beaucoup des reformulations ou des traductions de processus effectifs sous forme d'algorithmes ou de programmes. Par exemple : les formules explicites et les fonctions définies par une formule algébrique, les suites (les deux modes de génération) ou encore les modèles de lois par des répétitions d'expériences aléatoires.

Questionnement de l'algorithme

Concernant les algorithmes, aucune problématique n'est précisément posée. Seules les tâches de programmer, d'écrire, de comprendre, de lire un algorithme sont en jeu. Cependant, malgré la pauvreté des algorithmes en jeu, des questions pourraient être soulevées sur certains des algorithmes du programme : les questions d'approximation et d'erreurs, les questions de complexité, d'efficacité, les questions de comparaison d'algorithmes et d'optimalité.

Aspects

Le seul aspect présent est l'aspect effectif qui semble motiver la grande majorité de l'algorithmique, notamment au travers de la programmation et de l'exécution d'algorithmes.
L'aspect résolution de problèmes est aussi présent mais le fait qu'un même algorithme résolve toute une famille d'instances d'un problème n'est pas explicité.

La question de la preuve n'est jamais abordée ni même sous une forme atténuée de validation d'un algorithme. On ne trouve aucune référence à la complexité d'un algorithme ou à son efficacité.
L'aspect modèles théoriques ne nous semble pas nécessairement devoir apparaître dans les

programmes (contrairement à la preuve et la complexité qui nous semblent contribuer à ce qui donne du sens au concept) mais la définition d'un algorithme constitue en quelque sorte un modèle très simple que l'on ne retrouve même pas dans ces programmes.

On peut donc affirmer que seul l'aspect outil est en jeu dans ces programmes.

Conceptions

Le discours sur les formes d'algorithmes évoque AI et AM. Cependant les activités algorithmiques proposées relèvent fortement de AI et même simplement d'une activité d'implémentation comme traduction dans un langage de programmation et dans un objectif d'exécuter les algorithmes.

Les problèmes en jeu sont parfois des problèmes au sens de notre définition mais plus souvent des problèmes de simulation ou d'application d'une technique. Il ne s'agit donc pas de résoudre un problème mathématique mais de réaliser une tâche de calcul effectif non réalisable à la main.

Les opérateurs sont des algorithmes et plus particulièrement des programmes. Ils sont précisés comme constitués des *structures de contrôle*, qui sont les boucles, tests et manipulation des mémoires issus des langages de programmation classiques.

Les systèmes de représentation sont les *langages symboliques et naturels*. En fait la plupart du temps ce sont les langages de programmation qui sont évoqués.

Les structures de contrôle sont très peu présentes. On peut seulement citer les pratiques *de vérification et de contrôle* et la *rigueur*. On peut donc supposer que la structure de contrôle est l'ensemble des règles du langage, et que cela peut ensuite être testé via l'implémentation et la compilation/exécution des programmes.

La conception dominante nous semble donc AI-outil avec une absence de structure de contrôle forte. La conception AM est évoquée mais il semble qu'elle ne soit présente que très faiblement (notamment à cause de la confusion programme-algorithme qui peut être faite).
L'aspect preuve étant totalement absent, le paradigme PM ne peut être présent.

4 Algorithmique de la spécialité ISN

Le programme de la spécialité ISN se découpe en quatre branches, considérées parfois comme les quatre piliers de la science informatique : *représentation de l'information*, *algorithmique*, *langages et programmation* et *architectures matérielles*. Notons que ce découpage met directement en avant une distinction entre algorithmique et programmation.

Étant donnée notre problématique et étant donnés les outils que nous avons développés, nous n'étudierons que la partie algorithmique de ce programme. Tout comme dans la section précédente, nous utiliserons l'italique pour citer les programmes sans alourdir la lecture. De même aussi nous étudierons d'abord le discours sur l'algorithmique et ensuite les contenus spécifiques. Voici l'extrait du programme concerné :

4.2 Algorithmique

Un algorithme se définit comme une méthode opérationnelle permettant de résoudre, en un nombre fini d'étapes clairement spécifiées, toutes les instances d'un problème donné. Cette méthode peut être exécutée par une machine ou par une personne.

Les élèves ont été confrontés aux algorithmes très tôt dans leur parcours scolaire (avec les quatre opérations arithmétiques) et régulièrement de nouvelles situations de nature algorithmique leur ont été proposées ; ainsi, la construction de figures en géométrie euclidienne, la transcription des « formules » moléculaires en chimie, le code génétique ou encore l'analyse fonctionnelle en technologie sont autant de situations évoquant des algorithmes. Les programmes de mathématiques des classes de seconde et première contiennent une initiation à l'algorithmique sur laquelle il convient de s'appuyer.

À travers l'étude de quelques algorithmes, on développe la faculté de lire et comprendre un algorithme conçu par d'autres, puis d'en concevoir de nouveaux. Ces algorithmes sont exprimés dans un langage de programmation et exécutés sur une machine ou bien définis de manière informelle.

Savoirs	Capacités	Observations
Algorithmes simples - rechercher un élément dans un tableau trié par une méthode dichotomique ; - trier un tableau par sélection ; - ajouter deux entiers exprimés en binaire.	**Comprendre** un algorithme et **expliquer** ce qu'il fait. **Modifier** un algorithme existant pour obtenir un résultat différent. **Concevoir** un algorithme. **Programmer** un algorithme. ♦ **S'interroger** sur l'efficacité d'un algorithme.	On présente simultanément les notions d'algorithme et de programme, puis on les distingue. L'objectif est une compréhension de ces algorithmes et la capacité à les mettre en œuvre. Les situations produisant une erreur (division par zéro, dépassement de capacité) sont mises en évidence.
Algorithmes plus avancés ♦ tri par fusion ; ♦ recherche d'un chemin dans un graphe par un parcours en profondeur (DFS) ; ♦ recherche d'un plus court chemin par un parcours en largeur (BFS).	**Comprendre et expliquer** (oralement ou par écrit) ce que fait un algorithme. ♦ **S'interroger** sur l'efficacité d'un algorithme.	L'objectif se limite à une compréhension des principes fondamentaux sans exiger leur programmation.

4.1 Discours

Définition

Ce programme d'algorithmique commence par une définition. Celle-ci est très proche de celle que nous avons proposée : *Un algorithme se définit comme une méthode opérationnelle permettant de résoudre, en un nombre fini d'étapes clairement spécifiées, toutes les instances d'un problème donné. Cette méthode peut être exécutée par une machine ou par une personne.*

On retrouve dans cette définition l'aspect EFFECTIF : aspect systématique de l'algorithme (*méthode opérationnelle*), finitude (*nombre fini d'étapes*), non-ambiguïté (*clairement spécifiées*), possibilité d'exécution par un opérateur (*peut être exécutée par une machine ou une personne*).

On retrouve aussi l'aspect PROBLÈME (*permettant de résoudre toutes les instances d'un problème donné*). Le fait qu'un algorithme doivent traiter une famille d'instances et pas une seule est ici mis en avant dès la définition. On peut comprendre que la notion de problème est très proche de celle que nous avons utilisée dans la partie 1. Cette notion d'instance évoque aussi l'entrée de l'algorithme, le terme *résoudre* fait, dans ce contexte, implicitement référence à une sortie adaptée à chaque instance traitée.

Le commentaire concernant l'"exécuteur", *une machine ou une personne* montre que la

notion d'algorithme n'existe pas uniquement dans un but technologique ou de programmation, mais est une façon de répondre de manière effective à des problèmes dans différents contextes.

Positionnement dans le cursus scolaire

Le programme rappelle que les élèves ne découvrent pas l'algorithmique et les algorithmes ici. Ils ont été en contact très tôt avec des algorithmes. L'exemple (classique et pertinent) des quatre opérations qui était donné dans les programmes de mathématiques se retrouve à nouveau ici.
Les autres exemples de *situations de nature algorithmique* sont moins clairs. Les *constructions géométriques* peuvent s'énoncer clairement comme des algorithmes, même si les constructions ayant un nombre constant d'étapes présentent peu d'intérêt en tant qu'algorithmes-objets [4].
Les trois autres exemples issus de la chimie, les SVT et la technologie ne sont pas des références à des algorithmes au sens de notre définition. Il s'agit plutôt de situations où une certaine action systématique est en jeu (des *situations évoquant des algorithmes*). Il nous semble qu'il s'agit surtout du côté systématique, automatique, guidé par certaines règles ou étapes élémentaires. C'est donc l'aspect effectif qui est mis en avant.
L'algorithmique vue en mathématiques en seconde et première est aussi reconnue comme expérience sur laquelle doit se baser l'enseignement et aucune référence n'est faite à une possible différence avec l'approche du cours de mathématiques.

Facultés en jeu

Deux *facultés* sont attendues des élèves dans cette option : celle de *lire et comprendre un algorithme conçu par d'autres* qui se situe du côté outil de l'activité et celle d'en *concevoir de nouveaux*. C'est l'aspect effectif (notamment en ce qui concerne la transmissibilité des algorithmes (lire et comprendre ceux des autres). Le type de situations où des algorithmes doivent être conçus n'est pas précisé. Il n'y a aucune référence à la validation d'un algorithme que l'on a conçu.

Deux formes d'algorithmes sont évoquées. Ils peuvent être *exprimés dans un langage de programmation puis exécuté sur une machine* ou bien *définis de manière informelle*. Les premiers font référence à AI et à l'aspect outil (effectivité et résolution de problèmes), puisque l'exécution est mise en avant. La deuxième fait référence à AM, à l'expression d'algorithmes indépendamment des langages de programmation.

Ces facultés doivent être développées par l'étude de quelques algorithmes, dont on imagine qu'il s'agit de ceux spécifiés dans les contenus.

4.2 Contenus

Les contenus se découpent suivant deux points : Les *algorithmes simples* pour lesquels certaines *capacités* sont attendues et les *algorithmes plus avancés* pour lesquels les *capacités* attendues sont moindres et pour lesquels *l'objectif se limite à une compréhension des principes fondamentaux sans exiger leur programmation.*

4. Tout comme la résolution des équations de degré 2 évoquées précédemment.

Tout d'abord relevons l'observation suivante : *On présente simultanément les notions d'algorithme et de programme, puis on les distingue.* Ici, pas d'amalgame entre les deux, la différence doit être explicitement faite par l'enseignant (suivant la définition du programme).
Trois algorithmes simples sont au programme : la recherche dichotomique dans un tableau, le tri d'un tableau par sélection et la somme d'entiers en binaire. Ces trois exemples sont issus du traitement des données et de la gestion des nombres en machine. Ce sont des algorithmes "de base" qui répondent à des besoins de gestion des données et des entiers dans les machines courantes.
Concernant ces algorithmes les capacités attendues sont de comprendre et expliquer les algorithmes, de les modifier pour obtenir d'autres résultats, de concevoir des algorithmes et de les programmer. Toutes ces attentes concernent l'algorithme-outil. Même *comprendre un algorithme et expliquer ce qu'il fait*, car l'objectif affiché est seulement *une compréhension de ces algorithmes et la capacité à les mettre en œuvre* : aucune nécessité de preuve ou de validation n'est avancée. On ne sait même pas s'il s'agit de déterminer quel problème est résolu ou d'expliquer en quoi l'algorithme résout tel problème ou s'il s'agit simplement de comprendre les différentes étapes et instructions de l'algorithme.

Toujours concernant ces *algorithmes simples*, l'enseignant doit mettre en évidence *les situations produisant une erreur (division par zéro, dépassement de capacité)*. Ces points font bien sûr principalement référence à la programmation et aux limites liées au matériel. Ce sont donc ici des éléments liés à la capacité de programmation des algorithmes qui sont en jeu. C'est le paradigme AI qui semble dominer.

La seule apparition de problèmes où l'algorithme est objet est la capacité *S'interroger sur l'efficacité d'un algorithme* qui est présent pour tous les contenus (algorithmes simples et plus avancés). Cependant, cette unique capacité pour l'algorithme-objet fait partie de points *optionnels et [qui] seront traités en fonction des équipements disponibles ainsi que des orientations pédagogiques choisies par les enseignants.* L'algorithme en tant qu'objet risque de ne pas beaucoup apparaître.

Concernant les *algorithmes plus avancés*, les algorithmes à aborder sont le tri par fusion, le parcours de graphe en profondeur et le parcours en largeur (pour un plus court chemin). Ces algorithmes sont aussi issus du traitement des données et abordent des structures de données ou des types de données abstraits classiques.
Ces algorithmes plus avancés ne doivent pas être programmés et on peut affirmer sans risque que cela est lié aux difficultés d'implémentation de ces algorithmes mais aussi des structures en jeu (graphes, arbres...) alors que certaines opérations "non-élémentaires" peuvent rester telles quelles dans un algorithme décrit *de manière informelle.* Le point où nous voulons venir est qu'il s'agit très clairement ici du paradigme AM. Contrairement à ce que nous avons pu voir dans les programmes, les algorithmes présentés ici ne sont pas simplement une écriture des programmes associés sur papier mais bien l'expression d'une méthode générique, indépendante de contraintes de programmation et utilisant certaines opérations de bases qui demanderaient un travail de programmation complexe.

Tous les problèmes du programme nécessitent réellement le concept d'algorithme pour être résolus. C'est-à-dire qu'il ne s'agit pas ici de mettre sous forme d'algorithmes (ou de programmes sur papier) des procédures que l'on peut exprimer autrement (constructions, fonctions, suites, formules...) mais bien de problèmes pour lesquels l'algorithme répond à un besoin concernant leur résolution.

Bilan

Nous résumons les points important en suivants la grille d'analyse :

- L'algorithme est clairement défini. En particulier il est nettement distingué de la notion de programme. Son rôle dans la résolution de problème est bien précisé en appuyant sur la notion d'instance notamment.
- Les algorithmes en jeu répondent à des problèmes de traitement informatiques des données, mais sont bien des éléments de $\mathscr{P}_{\text{A}}$. Les problèmes choisis mettent bien en évidence le statut de la résolution algorithmique.
- Certains algorithmes sont l'objet d'un questionnement, en particulier leur complexité est à discuter en classe.
- On retrouve les aspects EFFECTIVITÉ et PROBLÈMES, mais aussi l'aspect COMPLEXITÉ. L'algorithme n'est pas qu'outil, l'aspect objet apparaît.
- On retrouve deux paradigmes : AI et AM, clairement délimités. L'efficacité des algorithmes est questionnée dans ces deux cadres, on peut donc affirmer que les conceptions AI-outil, AM-outil, AI-objet et AM-objet sont présentes.

La transposition en place diffère fortement de celle repérée dans les programmes de mathématiques. L'étude du document ressource pour la seconde devrait permettre de préciser cela.

5 Documents d'accompagnement "Ressources pour la classe de seconde"

Tout comme dans les sections précédentes nous réserverons l'italique pour les termes et formules extraits du document. Les citations plus longues ne sont pas en italique mais se distinguent du corps du texte par un retrait à gauche.

Le document "Ressources pour la classe de seconde - Algorithmique" que nous analysons se décompose suivant six sections et une présentation des logiciels utilisés. Les deux premières sections intitulées *Présentation générale* et *Une initiation à l'algorithmique* relèvent d'un discours général sur l'algorithmique et ce qu'elle doit être dans les programmes. L'analyse de ces sections peut se faire de manière similaire à l'analyse menée précédemment pour les programmes. Pour éviter une lecture fastidieuse d'une nouvelle analyse de ce type, nous supposerons que le lecteur comprend maintenant l'utilisation concrète de la grille et nous nous focaliserons sur les résultats, tout en illustrant au maximum nos propos par des extraits du document.

Les parties suivantes intitulées *Exemples de dispositifs en classe*, *Algorithmes et géométrie*, *Algorithmes et fonctions* et *Algorithmes et probabilités* sont des propositions d'activités pour la classe. Nous préciserons la grille que nous utilisons pour analyser ce type de ressources, qui diffère légèrement de celle utilisée précédemment. Nous entrerons donc ici plus dans le détail de l'analyse.

Enfin, il ne nous semble pas utile pour l'analyse de s'attarder sur la section *Présentation rapide des logiciels*. Nous relevons simplement cela comme une indication d'un fort ancrage des activités algorithmiques dans l'utilisation d'outils informatiques et de l'importance de la programmation que nous avons déjà repérée dans les programmes.

5.1 Discussion sur l'algorithmique

Comme dit plus haut, nous présentons ici l'analyse du discours des sections *Présentation générale* et *Une initiation à l'algorithmique*. Ces sections confirment la conception que nous avons relevée dans les programmes de mathématiques, en particulier une forte présence de l'algorithme en tant qu'outil ainsi que du paradigme AI.

L'activité algorithme et le rôle des algorithmes

Le document présente l'algorithmique comme un type d'activité qui trouve son sens dans l'apprentissage de la démarche scientifique. Tout au long du document, l'algorithmique est donc associée à une démarche expérimentale ou à une démarche de résolution de problème. On peut percevoir à travers cela qu'il s'agit déjà de présenter l'algorithme comme un outil pour les mathématiques.
Deux grands points justifient l'introduction de cette activité dans l'enseignement des mathématiques :

- La présence universelle des algorithmes, liée aux progrès de la science informatique : les algorithmes et l'activité algorithmique sont présents partout et les mathématiques ne peuvent y échapper. On peut percevoir une motivation sociale (de démystification des machines) derrière cela, en montrant les algorithmes qui se cachent derrière les machines :

  > Leur présence cependant, ne se traduit pas par un contact direct avec l'utilisateur qui assimile volontiers « la machine » à son mode de fonctionnement. (p.3)

- Aborder différemment les objets mathématiques. Les enjeux présentés semblent immenses :

  > Le développement du calcul automatisé a permis (au niveau de la recherche) de développer de nouveaux objets (fractales...), de nouvelles méthodes de démonstrations et a profondément modifié la pratique des chercheurs. (p.3)

  Mais cependant, au niveau de la classe de seconde, il s'agit simplement d'*étudier certaines notions sous un angle différent*, c'est-à-dire, comme nous le verrons plus loin, de rendre ces notions effectives en les programmant.

Le document insiste fortement sur ce qui est demandé dans le programme : l'activité algorithmique ne doit pas faire l'objet de cours spécifiques, doit être mise en place tout au long de l'année et de manière transversale à tous les chapitres et lors de la résolution de problèmes. Il est intéressant de noter quelles motivations doivent mener à l'introduction d'éléments d'algorithmique :

> L'introduction de chaque nouvel élément (variable, boucle, itération, etc.) devrait apparaître lors de la résolution de problèmes pour lesquels les démarches habituelles sont malcommodes ou peu performantes : par exemple dans le cas de répétition d'une tâche, ou dans le cas d'un traitement trop long pour être envisagé « à la main ». Ces situations peuvent être rencontrées lors de l'extension à des cas plus généraux de situations déjà rencontrées : recherche du pgcd de nombres très grands, tri d'un très grand nombre de valeurs numériques, simulations sur des échantillons de grande taille...(p.4)

On voit ici une motivation pour l'algorithmique qui est de l'ordre de la capacité d'un ordinateur ou d'une calculatrice à traiter des grandes quantités d'opérations de manière répétitive. Il s'agit, selon nous, plus d'une motivation de la programmation, qui implique de décrire un procédé systématique, alors que l'on pourrait aussi attendre, en mathématiques, une motivation de l'algorithmique relative à la résolution générale d'un problème (c'est-à-dire de toutes ses instances).

D'ailleurs, la prépondérance de la programmation dans le document montre bien cet appui très fort sur la machine. Il est pourtant précisé :

> La pratique de l'algorithmique ne se résume pas à l'écriture de programmes ; il serait même judicieux de ne pas commencer par là. Il convient donc de proposer aux élèves des situations, activités et organisations pédagogiques variées. (p.4)

Mais, cependant, il semble que ces situations sans programmation aient pour rôle de préparer à l'introduction des outils pour la programmation, un travail préparatoire introduisant un formalisme pour faciliter l'implémentation. Nous reviendrons sur ce point dans les paragraphes suivants.

Définition de l'algorithme et formalisme pour l'écriture

Contrairement au programme, le document ressource propose une clarification, voire une définition de la notion d'algorithme. On trouve aussi une description de la structure d'un algorithme et du formalisme nécessaire à son expression.

Tout d'abord, est évoqué, comme dans les programmes, le fait que les élèves ont déjà rencontré l'algorithme dans leur cursus. La présentation de la notion d'algorithme dans la classe semble devoir se baser sur cette expérience :

> Dans le cours de Mathématiques, les algorithmes apparaissent très tôt dans la scolarité. Opérations posées, réduction au même dénominateur, résolution d'une équation du second degré, factorisation d'une expression polynomiale, identification d'une situation de proportionnalité, tous ces algorithmes supplantent parfois chez les élèves les objets qu'ils manipulent.
> En travaillant dans un univers plus restreint, dans lequel les règles en vigueur et la symbolique utilisées sont en nombre limité, on essaiera de préciser et de formaliser partiellement la notion d'algorithme. (p.3)

On sent ici l'importance de présenter l'algorithme comme l'expression d'une méthode dans un langage symbolique limité. Il s'agit presque ici de définir l'algorithme comme le langage permettant son expression.

Plus loin, on trouve une explicitation de la notion d'algorithme et une définition (issue de l'Enciclopaedia Universalis) qui ne semble pas devoir être nécessairement choisie ou présentée aux élèves :

> Plus généralement le mot « algorithme » désigne tout procédé de calcul systématique voire automatique. S'ajoute à cela la notion de « finitude ».
> On définit parfois les algorithmes de la manière suivante : « un algorithme est une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver, en un nombre fini d'étapes, à un certain résultat et cela indépendamment des données. » Le résultat doit donc s'obtenir en un temps fini.
> À propos des « règles à appliquer », il faut entendre un traitement fait sur des

données imposé par une suite « d'instructions » visant à transformer ces données pour arriver au résultat visé.
Ces « instructions » sont de natures diverses selon le type de données au départ.
C'est ce que nous allons préciser.(p.6)

Il semble que les points importants ici soient l'aspect systématique de l'algorithme et sa finitude. Cependant le document présente ensuite une structuration de l'algorithme qui montre clairement que dans la classe de seconde, c'est la forme et le langage qui caractérisent un algorithme.

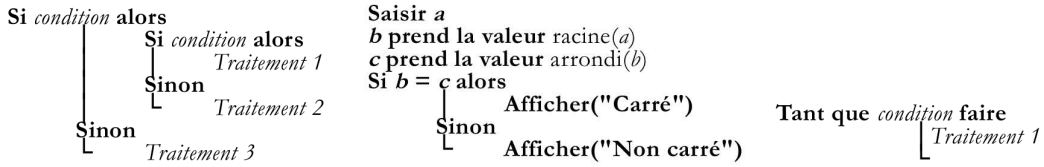
La suite présente de manière extrêmement précise, les éléments constitutifs d'un algorithme et le langage pour les exprimer. Tout d'abord, l'algorithme est découpé en trois parties : la préparation du traitement, le traitement, et la sortie des résultats. Il s'agit d'une très forte structuration de l'expression de l'algorithme qui nous semble être inspirée de certains langages de programmation. La ressemblance avec la description théorique de la syntaxe d'un langage informatique [5] est frappante.

La préparation du traitement est décrite comme le repérage des données nécessaires à la résolution et leur structuration.

Le traitement consiste en la description des instructions *à donner pour une exécution automatique*. Ces instructions sont ensuite présentées autour d'un exemple. Elles sont formulées dans ce qui est appelé un *pseudo langage « en français »* et doublées d'illustrations dans le cadre de logiciels précis. Elles sont réparties en trois groupes :

- Les *instructions pour traiter les données* : la lecture de données, l'affectation de variable et l'écriture de données. il s'agit des manipulations de base, qui sont ensuite organisées et contrôlées par les autres instructions.
- Les *séquences d'instructions*, c'est-à-dire la mise en séquence, les unes à la suite des autres de plusieurs instructions
- Les *instructions (ou structures) de contrôle* : la structure alternative et les structures répétitives.

Chacune étant décrite en donnant la formulation à utiliser pour l'écrire.
Cela conduit à des algorithmes devant respecter une structuration très poussée, comme les exemples suivants le montrent :



Enfin, comme on peut le voir ci-dessus, la sortie des résultats consiste à faire apparaître le résultat de l'algorithme d'une façon ou d'une autre. En fait il s'agit de faire produire le résultat via une interface informatique. On constate ici que l'algorithme, tel qu'il est décrit, fait explicitement référence aux machines sur lesquelles il peut être implémenté. Cela est caractéristique d'un amalgame entre algorithme et programme que nous avions entrevu dans les programmes et sur lequel il faut s'attarder.

5. Un langage de programmation est défini à l'aide d'une grammaire formelle qui se décrit d'une manière très proche de ce qui est fait dans ce document ressource.

Amalgame programme-algorithme

Nous venons de voir que l'algorithme dans les documents-ressources est décrit suivant une structure extrêmement proche de certains langages de programmation. Le mélange entre ce qui relève de l'implémentation sur machine et ce qui relève d'une méthode générique (l'algorithme, au sens où nous l'avons défini) va encore plus loin.

Les objets manipulés par les algorithmes mélangent objets mathématiques, codage de ces objets et structures de données. Par exemple :

> Ces données peuvent être numériques, ou sous forme de textes (on dit souvent des chaînes de caractères), ou de type logique (à deux valeurs possibles, vrai ou faux), ou enfin de type graphique (des points).
> Souvent les données pertinentes doivent être agencées sous une forme plus vaste, comme par exemple des tableaux ou listes où on peut par exemple ranger commodément les valeurs prises par une fonction sur un grand nombre de points. (p.6)

La question des entrées et sorties montre elle-aussi un tel amalgame. L'entrée de l'algorithme, au sens où nous l'entendons, représente une instance (ou son codage) et la sortie représente la réponse au problème (ou son codage) pour l'instance donnée. Ces notions d'entrée et de sortie font référence à l'entrée et la sortie de l'algorithme lui-même. Or ici cela est confondu avec les entrées et sorties dans un programme via une interface quelconque. Or, lorsque l'on s'intéresse à l'algorithme comme méthode de résolution d'un problème, il n'est d'aucune utilité de savoir si l'information est saisie au clavier, affichée à l'écran ou signalée par un bip.

On parlera d'amalgame entre saisie et entrée (respectivement entre sortie et affichage) pour désigner cette confusion entre les composantes d'un algorithme (son entrée, sa sortie) et des moyens techniques qui peuvent être utilisés pour mettre en œuvre ces éléments dans un programme. Cette confusion nous semble problématique. D'une part cet amalgame entre entrée-sortie d'un algorithme et saisie-affichage de données induit une confusion entre la machine et la méthode que l'on implémente, l'algorithme. D'autre part, cela empêche la présentation des programmes comme des fonctions, ce qui ne permet plus de faire appel à un programme comme fonction intermédiaire, ni de programmer des algorithmes récursifs.

Par exemple :

> Dans cette phase peut aussi figurer ce qu'on appelle l'entrée des données, qui peut se manifester par la saisie de caractères ou de nombres sur le clavier, ou la lecture de la position du pointeur de la souris, ou encore par la lecture d'un fichier contenant ces nombres ou caractères. (p.6)

Ou encore :

> Les résultats obtenus peuvent être affichés sur l'écran, ou imprimés sur papier, ou bien encore conservés dans un fichier. Si on n'en fait rien, ils « restent » en mémoire jusqu'à la prochaine exécution ou sont perdus. À l'occasion, la sortie pourrait être graphique (afficher ou déplacer le pointeur de la souris ou des objets sur l'écran) ou sonore ... voire sur Internet. (p.6)

Ce point nous amène à parler aussi du statut des variables. Tout d'abord, notons que même dans sa version pseudo-langage, l'algorithme tel que le propose le document nécessite une

déclaration des variables en jeu (comme dans les langages Pascal ou C) dans la préparation du traitement. Que ce soient les variables en entrées autant que les variables intermédiaires :

> Il s'agit aussi de repérer les résultats intermédiaires qu'il est bon de mémoriser pour la suite car indispensables au traitement. Il est parfois utile d'utiliser des variables auxiliaires pour ne pas perturber les données initiales. (p.6)

D'autre part le statut de la variable informatique dans la description d'algorithme et dans les programmes est le même. Nous avons décrit la notion de variable informatique dans l'algorithme mathématique (AM) comme un modèle de la gestion de la mémoire et de la variable dans les programmes informatiques.

> Cette action nécessite de créer une « mémoire » ou **variable** destinée à cet usage, zone de mémoire à laquelle il est commode de donner un nom ou **identificateur**. [...] Les identificateurs sont des suites de lettres et chiffres (sans espaces) qui doivent être choisies judicieusement pour que l'algorithme soit immédiatement lisible et interprétable [...]. (p.7)

Et une note de bas de page précise :

> Il ne faut surtout pas confondre la variable et son identificateur ; en effet, la variable, outre son identificateur qui est juste un nom, possède une « valeur » qui est son contenu et une « adresse » qui est l'endroit dans la mémoire où on va ranger la valeur. (p.7)

Il s'agit bien ici de la définition informatique d'une variable. Les questions d'adresse en mémoire ne jouant aucun rôle lorsque l'on développe un algorithme. C'est lors de son implémentation, pour des questions de programmation que cela peut avoir du sens.

Enfin l'ensemble des instructions constitutives d'un algorithme dans ce document (Si alors, Pour, Tant que, Répète jusqu'à) sont directement les instructions disponibles dans les langages de programmation les plus utilisés, traduites en français. Elles sont les seules présentées comme description d'un algorithme et toute autre représentation est rejetée, à l'instar des organigrammes :

> Nous n'utiliserons pas les « organigrammes » qui eurent leurs heures de gloire dans l'enseignement de l'informatique mais se sont avérés inadéquats face aux progrès de cette science (en particulier,dans le cadre des programmations fonctionnelle ou orientée objets). (p.7)

Un concept pour l'analyse : le programme-papier

Le fait que le document décrive l'algorithme comme l'expression d'une méthode dans un langage très proche d'un langage de programmation et le fait que les éléments de la programmation et ceux de l'algorithme soient toujours mêlés, dans un amalgame entre algorithme et programme, nous amène à introduire la notion de **programme-papier**.

Nous appellerons programme-papier, le fait de présenter les algorithmes comme des programmes (au niveau de la forme) et dans lesquels, bien que l'on ne soit pas sur une machine, on se soucie de points liés à l'implémentation et à la machine. Ces programmes-papier sont en quelque sorte des programmes génériques que l'on peut écrire en dehors d'un contexte de programmation mais qui relèvent bien de celui-ci.

Nous considérons qu'ils sont caractéristiques d'un ancrage dans AI et dans une activité centrée sur la programmation. Bien qu'ils ressemblent à des algorithmes dans AM, les

programmes-papier ont pour objectif central l'exécution en machine et ils ne se focalisent pas sur la méthode de résolution. La présence d'un langage très formalisé, mais surtout d'instructions relatives à la machine (calculatrice, ordinateur, etc...) est un critère efficace pour les repérer. La suite du document ressource en donne des exemples très nets.
Par exemple :

> L'« écriture des données » permet d'afficher pour l'utilisateur les valeurs des variables après traitement (ou en cours de traitement dans le cas où l'on veut contrôler l'exécution). On a choisi de la traduire par l'instruction : **Afficher** *identificateur*. On pourra peaufiner la présentation des résultats pour avoir un affichage lisible et compréhensible. Une variante consiste à « sortir » directement des informations non contenues dans une variable, nous le traduirons par : **Afficher** « message ». (p.7)

On reconnaît ici le programme-papier au fait que l'on a un algorithme qui n'est pas implémenté mais qui est exprimé dans un langage très précis et qui, sur papier, "dialogue" avec l'utilisateur au travers d'affichage à l'écran (et dans d'autres cas de saisie au clavier).

Algorithmes en jeu

Les algorithmes, ou les problèmes qui font appel à l'algorithme, présents dans cette partie du document sont :

- Les algorithmes anciens que les élèves ont déjà rencontré : algorithmes opératoires, algorithme des différences, algorithme d'Euclide, algorithmes de construction en géométrie.
- Les exemples de problèmes mettant en jeu des objets mathématiques sous un jour nouveau : organisation de la recherche du maximum d'une fonction, représentation d'une droite avec des pixels, tri des données nécessaire au calcul de la médiane.
- Les exemples *souvent repris* pour illustrer la notion d'algorithme : rendu de monnaie, recettes de cuisine, s'habiller, conduite d'un train, consultation d'un catalogue de bibliothèque.
- Les exemples pour présenter les *éléments de base d'un algorithme simple* : simulation d'un lancé de deux dés avec calcul d'un gain, répétition 10 fois de cette expérience, répétition de cette expérience jusqu'à obtenir un certain gain.

Ces algorithmes sont présentés pèle-mêle. Pourtant, selon nos critères, certains ne relèvent pas complètement du concept d'algorithme :

- Les algorithmes de tous les jours : rendu de monnaie, recettes, s'habiller, conduire un train et chercher dans un catalogue. La description de ces activités peut relever d'une définition de l'algorithme qui n'est pas celle des mathématiques et la notion de méthode ou de suite de règles nous paraît suffisante pour les qualifier. Ces activités font référence essentiellement à l'effectivité et l'on peut reprocher à certaines de ne pas résoudre une famille d'instances d'un problème.
- Simulation d'expérience : les trois variantes de simulation ne nous semblent pas relever réellement de l'algorithmique : il s'agit de mettre sous forme d'un programme ou d'un programme-papier, de traduire, une expérience avec des outils numériques (générateur de nombres aléatoires). On ne traite pas un problème au sens où nous l'avons défini.

– Algorithmes de construction en géométrie : la notion de construction peut être exprimée sous forme d'un algorithme mais l'algorithmique n'apporte rien de plus dans la résolution du problème. C'est, nous semble-t-il, la programmation qui motive ici le choix de ces algorithmes ainsi que l'aspect effectif et systématique qui est déjà dans la notion de construction. De plus, nous avons pu voir dans les programmes qu'il s'agit essentiellement de faire des traductions des formules sur les coordonnées des points.

Les autres algorithmes et problèmes généraux correspondent à notre définition. Parmi eux, seuls six algorithmes sont "riches" et pourraient être objets d'étude : la recherche d'extremum d'une fonction sur un intervalle (par balayage et par tirage aléatoire), le test de la monotonie d'une fonction (par balayage), la recherche par dichotomie ("nombre à deviner" et recherche du zéro d'une fonction) et la recherche du maximum d'une fonction croissante puis décroissante. Ce sont tous des algorithmes d'approximation en analyse (à l'exception de l'algorithme du "nombre à deviner", mais son rôle est d'introduire la dichotomie pour rechercher des zéros de fonctions).

Pour terminer, notons que pour illustrer l'introduction des diverses instructions (affectations, boucles, etc.), le document s'appuie sur les simulations de répétition d'expériences aléatoires que nous venons d'évoquer. On voit bien ici, par le choix de ces exemples non représentatifs de la notion d'algorithme, que ce n'est pas la résolution d'une famille d'instances d'un problème qui est l'enjeu mais bien la mise en œuvre concrète, via les outils de programmation, de constructions issues des mathématiques.

Questions sur les algorithmes et tâches de l'élève

Le document met bien en évidence le travail attendu de l'élève concernant cet enseignement d'algorithmique, à travers les *compétences qui doivent être identifiées et travaillées*, les *pratiques de l'élève* ou les éléments de *l'évaluation par compétence.*

On peut distinguer tout d'abord que les tâches proposées sont pour beaucoup relatives à la programmation. Certaines le sont directement comme les *compétences* : *adapter l'algorithme aux contraintes du langage de programmation*, *valider un programme simple* ou les *support d'activité* : *transpositions d'algorithmes dans différents types de langages*, progressivité dans le choix et l'utilisation des outils de programmation.
D'autres y font référence moins directement mais on peut comprendre que le cadre est celui de la programmation :

– certaines parce que la tâche proposée se fait plus couramment dans un langage de programmation : *modifier un algorithme pour obtenir un résultat particulier* ou *complexification progressive*,
– d'autres parce que l'on retrouve les caractéristiques du programme-papier comme *mettre au point une solution algorithmique : comment écrire un algorithme en « langage courant » en respectant un code, identifier les boucles, les tests, des opérations d'écriture, d'affichage...* ou *valider la solution algorithmique par des traces d'exécution et des jeux d'essais simples.*

Les autres tâches sont reliées à la compréhension et l'écriture d'algorithmes : *comprendre et analyser un algorithme existant*, *relecture d'algorithmes* ou *créer un algorithme en réponse*

à un problème donné.

Deux fois seulement sont évoquées des tâches qui pourraient relever de l'algorithme comme objet [6] :

- *valider la solution algorithmique par des traces d'exécution et des jeux d'essai simples* qui réfère à une validation qui n'est pas de l'ordre de la preuve mais plutôt expérimentale. Il s'agit plus de trouver des bugs que de valider la méthode pour toute entrée.
- *valider un programme simple.* La validation d'un programme peut passer par la preuve de l'algorithme sous-jacent mais aucune référence précise à cela n'est faite et nulle part on ne parle de preuves d'algorithmes.

Les *pratiques* à évaluer sont uniquement tournées vers la compréhension d'un algorithme (au sens de la compréhension de ses étapes et de l'explicitation de son but) sa modification et la création d'un algorithme. C'est-à-dire uniquement des tâches où l'algorithme est outil.

Aspects

Nous l'avons déjà dit, la grille d'analyse soulève des questions qui se recoupent en partie. Plus précisément, elles sont dans un ordre où les premières questions jouent un rôle important pour répondre aux suivantes. Ainsi, les questions auxquelles nous avons répondu dans les points précédents nous permettent de répondre précisément à la question des aspects et le lecteur devrait déjà avoir une idée des aspects qui sont présents ou non.

L'aspect EFFECTIF est extrêmement présent au sein de cette partie du document. En particulier on retrouve la notion de non-ambiguïté, de méthode systématique, de transfert à un opérateur ou une machine et de finitude.
Le poids important qui est est mis sur la programmation et que nous avons soulevé plus haut contribue de manière très forte à cet aspect. En particulier le choix de proposer d'exprimer les algorithmes dans un langage très strict et l'objectif de programmer et d'exécuter ces algorithmes mettent souvent la non-ambiguïté et l'automatisation des méthodes au centre des préoccupations pour l'algorithmique.
On le retrouve aussi dans les caractéristiques évoquées pour la notion d'algorithme. Par exemple, la motivation de l'algorithme *dans le cas de la répétition d'une tâche, ou dans le cas d'un traitement trop long pour être envisagé « à la main »* relève complétement de la possibilité de rendre automatiques certaines tâches si elles sont explicitées clairement. La référence à Al-Khuwarizmi et ses *procédés de calcul à suivre pas à pas*, la définition choisie de l'Encyclopaedia Universalis, ou l'insistance sur la notion de *« finitude »* sont aussi des références caractéristiques à l'aspect effectif.
La *présence universelle des algorithmes* dans le quotidien et les autres disciplines, que soulève à plusieurs reprises le document, relève aussi de cet aspect.

L'aspect PROBLÈME se dégage nettement aussi. En particulier les notions d'entrée et de sortie sont très présentes (malgré l'amalgame avec les interfaces d'entrée et de sortie de la machine). La volonté de placer l'utilisation des algorithmes pour la résolution de problèmes peut faire référence à cet aspect problème, cependant, nous avons vu précédemment que

6. Nous reviendrons dans les paragraphes suivants sur la distinction outil et objet. Rappelons simplement pour l'instant, que pour être objet, l'algorithme ne doit pas seulement être questionné mais certains opérateurs et une certaine structure de contrôle doivent être évoqués.

l'amalgame algorithme-programme peut laisser penser que cela fait seulement référence à l'utilisation de logiciels de programmation dans une activité de résolution de problèmes. En effet, il s'agit de *faire en sorte que les mathématiques et l'algorithmique soient au service d'activités de résolution de problèmes pour les sciences.* Ces problèmes pour les sciences pouvant surement être d'un genre très éloigné de notre définition de problème.
Notons tout de même qu'une première définition de l'algorithme met en avant clairement cet aspect problème :

> Dans un premier temps rédiger un algorithme consiste à décrire les différentes étapes de calcul pour résoudre un problème algébrique, numérique ou décisionnel. (p.6)

L'aspect preuve est légèrement évoqué et ne fait pas, nous l'avons vu, l'objet de tâches particulières. Son évocation relève d'un discours général sur la place de l'algorithmique et de l'informatique vis-à-vis des mathématiques :

> Le développement du calcul automatisé a permis (au niveau de la recherche) de développer de nouveaux objets (fractales...), de nouvelles méthodes de démonstrations et a profondément modifié la pratique des chercheurs. (p.3)

L'autre référence à la preuve à lieu au moment de l'introduction de la *boucle conditionnelle*, concernant les problèmes de terminaison [7] :

> Pour que l'algorithme soit correct, il est nécessaire que la condition cesse d'être vérifiée au bout d'un nombre fini de répétitions. Un raisonnement mathématique permet parfois de s'en assurer, mais il est parfois difficile d'avoir l'assurance du fait que le programme ne bouclera jamais indéfiniment. (p.9)

Quelques lignes plus loin, la répétition d'une instruction jusqu'à obtention d'un gain fixé soulève la note de bas de page suivante : *Situation fort intéressante tant au plan mathématique qu'algorithmique : l'algorithme se termine-t-il ? Rien ne l'assure !* On peut considérer que l'aspect preuve est reconnu mais sa présence reste tout de même extrêmement légère et aucune attente ou exigence n'est énoncée concernant cet aspect. Étant quasiment absent des programmes, et très peu évoqué ici, on peut faire l'hypothèse que cet aspect n'existera surement pas dans les classes.

L'aspect COMPLEXITÉ est moins présent encore. On parle de *temps de calcul* à une reprise, mais il s'agit seulement de dire que cela, avec d'autres critères, *détermine de manière plus ou moins ouverte le choix de l'instrument*, c'est-à-dire que cela soulève des questions de choix des *supports de programmation.*

L'aspect MODÈLES THÉORIQUES n'est pas attendu dans un tel document. Cependant, tout comme la preuve est évoquée sans être vraiment attendue en classe, on pourrait imaginer trouver, à destination des enseignants, une référence à ces notions théoriques, tout comme aux questions liées à la complexité.

En résumé, seuls les aspects EFFECTIVITÉ et PROBLÈME sont mis en avant dans cette partie du document, comme les programmes le laissaient attendre. Ajoutons que l'aspect EFFECTIVITÉ semble bien plus dominant que l'aspect PROBLÈME, comme l'exemple choisi pour illustrer les différentes instructions, une expérience aléatoire, le montre. C'est l'utilité de la notion d'algorithme pour exprimer une méthode ou une construction de manière

7. Notons d'ailleurs qu'à ce sujet l'amalgame programme-algorithme apparaît clairement : le document ne parle pas de terminaison mais explique que l'instruction sera *exécuté[e] indéfiniment*, que *l'utilisateur sera obligé d'arrêter (brutalement) le programme* et qu'il s'agit d'*une erreur majeure de programmation.*

non-ambigüe et implémentable et exécutable par un ordinateur qui prime sur la question de résoudre un problème particulier, quelle que soit l'instance choisie.
C'est donc très largement l'aspect outil qui domine cette partie des documents-ressources.

Conceptions

Ce que nous avons dit pour l'étude des aspects vaut aussi pour l'étude des conceptions : c'est pour beaucoup les questions précédentes de la grille qui donnent les moyens de repérer la ou les conceptions en jeu. En particulier, nous avons déjà abordé la question des problèmes proposés, de la forme dans laquelle sont exprimés les algorithmes et des structures de contrôle qui sont en jeu (où plutôt qui ne sont pas en jeu).

Tout comme nous l'avions remarqué pour les programmes de mathématiques, le discours général sur l'algorithme évoque les paradigmes AI et AM. Cependant, il ressort que les algorithmes, lorsqu'ils ne sont pas sous forme de programmes, n'existent que sous la forme de programmes-papier. Dans ce cas on ne peut pas vraiment dire que le paradigme AM soit en jeu.

Les problèmes en jeu, nous l'avons vu, sont soit des problèmes de $\mathscr{P}_{\text{A}}$, pour lesquels on cherche une solution commune à toutes les instances, soit des questions ou des activités qui ne sont pas des problèmes au sens algorithmique. Par exemple, ce sont des questions de simulation ou d'écriture sous la forme d'un algorithme ou dans un langage spécifique de méthodes ou constructions automatiques. D'autres problèmes évoqués sont des problèmes de la vie courante (*conduire*, *s'habiller*, etc.).

Les opérateurs ne sont que des algorithmes ou des programmes. Cela est complètement lié au fait que seulement des problèmes de $\mathscr{P}_{\text{A}}$ sont posés et que nous sommes donc du côté outil des conceptions. Certains opérateurs sont des programmes qui ne sont pas des algorithmes (dans le cas des simulations). On peut déjà dire que par rapport aux μ-conceptions, la conception de l'algorithme, ici, s'étend à tout ce qui est programmable, y compris les méthodes que nous n'avons pas considérées être des algorithmes.

Il est important de s'attarder sur les systèmes de représentations qui sont proposés. En effet, ce sont eux qui, en grande partie, permettent de distinguer le paradigme dans lequel on se trouve.
Certaines remarques laissent penser que cohabitent deux paradigmes dans le document : AM et AI. Notamment, il est dit :

> La pratique de l'algorithmique ne se résume pas à l'écriture de programmes ; il serait même judicieux de ne pas commencer par là. (p.4)

De même, une partie des *compétences* consiste à traduire ou adapter un algorithme à un langage de programmation. Et une partie des *algorithmes* de la vie courante (prenons comme exemple la *consultation d'un catalogue de bibliothèque*) s'adressant à des humains, les exprimer dans un langage de programmation ne semblerait pas pertinent. Ajoutons la remarque suivante :

> On pourrait très bien commencer par exécuter des algorithmes sans ordinateur à la main sur papier, avec les mains, avec les pieds ou avec des objets, etc. (p.4)

Ce qui peut laisser penser que les algorithmes présentés ne sont pas donnés dans un langage de programmation. Pourtant, deux langages d'expression des algorithmes sont présentés : le programme et le programme-papier donné par le *pseudo-code « en français »* dont nous

avons déjà longuement parlé. De ce point de vue on se situe dans AI seulement. Pour être dans AM, il faudrait une volonté claire de s'abstraire des contraintes de la machine et de se concentrer sur la méthode de résolution du problème elle-même, ce qui n'apparaît pas ici.

Les structures de contrôle ne sont pas beaucoup mises en avant ici. C'est souvent le cas dans les discussions générales sur l'algorithmique. Elles seront plus facilement repérables dans des activités détaillées et seront surement plus explicites dans la suite du document ressource. Étant donné la quasi-absence de l'aspect PREUVE [8], on peut déjà noter que ces structures de contrôle ne sont pas de l'ordre de la preuve mathématique.
Bien sûr, des structures de contrôle existent mais elles relèvent du respect du langage (*écrire un algorithme en langage courant en respectant un code*) et de la validation par l'exécution (*valider la solution algorithmique par des traces d'exécutions et des jeux d'essais simples*).

Nous devons préciser pourquoi nous considérons que les problèmes de $\mathbb{P}$ sont absents et que les conceptions-outils n'apparaissent donc pas. Quelques questions soulevées peuvent être vues comme s'adressant à l'algorithme en tant qu'objet. Celle de la terminaison d'algorithmes dont nous venons de parler ou celle de la précision ou de la fiabilité d'un algorithme d'étude de fonctions réelles. Aucun opérateur n'est donné pour répondre à ces questions et encore moins de structure de contrôle.
On retrouve aussi des questions quant au codage des objets ou au temps de calcul mais il s'agit uniquement de savoir adapter le choix du support de programmation selon ces critères.

En résumé, la conception présente ici est AI, avec une structure de contrôle très faible et un amalgame fort entre algorithme et programme.

5.2 Activités proposées pour la classe

Suivant le découpage proposé, nous étudions maintenant les sections *Exemples de dispositifs en classe*, *Algorithmes et géométrie*, *Algorithmes et fonctions* et *Algorithmes et probabilités*. Pour étudier ces propositions d'activités nous re-préciserons la grille d'analyse utilisée et nous montrerons comment les activités peuvent être analysées individuellement à l'aide de la notion de conception et du modèle épistémologique des μ-conceptions pour l'algorithme.

Une grille pour analyser les activités

Pour l'analyse de cette autre partie du document, qui concerne des propositions d'activités en classe, nous utiliserons la même grille que précédemment mais nous allons préciser son utilisation, en particulier en ce qui concerne l'étude des conceptions. En effet, nous avons pu voir que les discours généraux sur l'algorithmique sont très adaptés à la recherche des aspects mais pas forcément à celle de conceptions détaillées. Dans le cas d'activités précises proposées, c'est l'inverse. Repérer les aspects n'est pas forcément facile mais la description de la conception dans chaque activité devient plus aisée.
C'est cela que nous souhaitions préciser ici : pour chaque activité proposée, on peut chercher à repérer la conception en jeu, c'est-à-dire le quadruplet (P, R, L, Σ). Dans les cas précédents, nous ne pouvions que décrire de manière générale, le type de problème, le type d'opérateur, ou le type de structure de contrôle en jeu. Par exemple, il s'agissait uniquement de dire quelle famille de problèmes était en jeu : $\mathscr{P}_{\text{A}}$ ou $\mathbb{P}$? Quel genre de structure

8. La seule évocation de la preuve comme structure de contrôle étant qu'elle peut parfois assurer la terminaison. Nous l'avons évoqué au paragraphe précédent.

de contrôle : preuve mathématique ? implémentation ? exécution ?
Nous allons maintenant pouvoir préciser à chaque fois quel problème est traité, avec quels opérateurs, sous quelle forme et guidé par quelles structures de contrôle.

Décrire ces conceptions pour chaque activité est long. Nous n'allons pas détailler l'ensemble de ce travail, mais il nous semble nécessaire de montrer à travers quelques exemples représentatifs, comment se met en œuvre concrètement cet outil et jusqu'à quel niveau de détail il permet d'aller.

Dans un deuxième temps nous présenterons un bilan des analyses. Nous ne suivrons pas tout à fait le plan de la grille d'analyse pour alléger la lecture. Nous présenterons l'analyse en terme de conceptions (et nous passerons plus vite sur les outils déjà utilisés plus haut ou les éléments déjà repérés et discutés dans les programmes et le début du document). Le lecteur pourra retrouver, dans cette présentation, les éléments de réponse aux questions de la grille d'analyse, même s'ils ne sont pas formulés comme tels.

Conceptions détaillées et exemples choisis

• Le *jeu du cartable* (p.10) (voir tableau ci-dessous) : Il semble que l'objectif soit l'expres-

P	Préparer son sac pour le lendemain, étant donnés les éléments déjà dans le sac et l'emploi du temps du lendemain. Nous pourrions exprimer ce problème sous la forme (Instances, Question) si la forme des informations et les opérations élémentaires étaient précisées.
R	Ce sont les différentes stratégies possibles. Les opérations autorisées ne sont pas précisées.
L	Les opérateurs sont exprimés en français, dans un cadre déjà structuré par le découpage en préparation du traitement, traitement, édition des résultats et rédigés sur une fiche : – préparation du traitement : liste des disciplines du lendemain et liste du matériel déjà présent dans le cartable ; liste du matériel par discipline ; – traitement : comparaison des documents se trouvant dans le cartable et ceux à y mettre ; – édition des résultats : liste des fournitures placées dans le cartable. Une fois réalisée, cette fiche est fournie à un autre groupe qui peut ainsi la tester.
Σ	Le test par un autre groupe permet la validation de la stratégie. Aucune autre structure de contrôle n'est proposée.

sion rigoureuse d'une méthode systématique pour résoudre la tâche. Ce problème, proposé pour introduire la notion d'algorithme, met surtout en avant l'aspect effectif et ne présente pas un problème très riche algorithmiquement. Il s'agit surtout d'exprimer sans ambiguïté une stratégie. On reconnaît cependant la conception AM-outil.

• *Un peu d'épargne* (p.12) :
Cette activité est proposée dans un cadre de *lecture d'algorithme*. On retrouve la conception AI-outil dans une version très faible (tableau ci-dessous). On retrouve ici un programme-papier. De plus le problème est traité dans une version instanciée, on ne peut pas considérer que ce qui est décrit soit un algorithme, même si l'on reconnaît l'algorithme générique qui se trouve derrière le calcul décrit.
Pour cette même raison, on ne peut pas considérer que les questions posées sur ce "calcul" mettent l'algorithme en jeu en tant qu'objet. Ces questions (*Écrire les affichages successifs qui apparaîtront à l'écran* et *Donner un problème dont la solution est donnée par cet algorithme*[9]) portent sur un algorithme ou une méthode décrite pour un cas particulier, on ne peut donc pas considérer que AI-objet est présent. D'autant plus qu'aucun opérateur n'est proposé pour répondre à ces questions. Il s'agit donc ici simplement de comprendre

9. D'autant plus que l'énoncé précise que *S désigne la somme détenue par Paul à la banque et N désigne le nombre de semestres de dépôt.*

P	Le problème est instancié. Il s'agit de calculer le nombre de semestres nécessaires pour obtenir 8000 euros en plaçant 5000 euros à 2% par semestre. Il peut être vu comme une instanciation du problème général suivant (ou d'un sous-problème de celui-ci) : Données : une somme de départ S_d, un taux semestriel t et une somme à atteindre S_a. Question : Au bout de combien de semestres atteint-on S_a ?
R	Les opérateurs sont les instructions définies dans le document (affectation, boucles, affichage, etc.). On reconnaît des opérateurs liés à la gestion de l'affichage.
L	Le langage est le pseudo-code évoqué dans la première partie du document ressource : Mettre 5000 dans S **Mettre 0 dans N** **Effacer l'écran** **Tant que S est strictement inférieur à 8000** **Remplacer S par S*1,02** **Augmenter N de 1** **Afficher N et S** **Fin du Tant Que**
Σ	Aucune structure n'est évoquée. Surement, d'une part, parce qu'il s'agit de comprendre les instructions et d'en déduire le *problème* (plutôt une question instanciée) qu'elles permettent de résoudre et d'autre part, parce qu'il s'agit ici de simuler le bénéfice effectué chaque année.

des instructions données dans un certain langage.

• La *distance entre deux points* (p.16-17) : Deux activités sont proposées autour de la distance entre 2 points. La première s'intéresse au problème du calcul de la distance, elle met en jeu la conception AI-outil :

	Conception
P	Instance : les coordonnées de A et B, deux points du plan Question : Quelle est la distance AB ? Le but n'est pas la résolution du problème, il s'agit *d'automatiser le calcul de la distance.*
R	Les opérateurs sont les instructions du *pseudo-code* ou les commandes des langages de programmation proposés pour traduire ce *pseudo-code*. On trouve à la fois des instructions sur les objets manipulés et des instructions d'affichage et de saisie.

L	L'algorithme est décrit avec le *pseudo-code* du document puis exprimé dans les langages de programmation Casio et Python :
	Variables
	x_A , y_A // Coordonnées de A
	x_B , y_B // Coordonnées de B
	D // Carré de la distance de A à B
	Entrées
	Saisir x_A , y_A , x_B , y_B
	Traitement
	D **prend la valeur** $(x_B - x_A)^2+(y_B - y_A)^2$
	Sortie
	Afficher « AB²= »
	Afficher D
	Afficher « AB= »
	Afficher $\sqrt{D}$
	Traduction CASIO :
	`"A(X,Y)"`
	`"X="?→X`
	`"Y="?→Y`
	`"B(X,Y)"`
	`"X="?→Z`
	`"Y="?→T`
	`(Z-X)²+(T-Y)²→D`
	`"AB²="`
	`D◀`
	`"AB="`
	`√(D)◀`
	`End`
	Traduction PYTHON :
	`# Calcul de la distance entre deux points`
	`from math import *  # permet d'utiliser sqrt (racine)`
	`xa=input("Saisissez l'abscisse du point A :")  #`
	`ya=input("Saisissez l'ordonnée du point A :")  # saisie des`
	`xb=input("Saisissez l'abscisse du point B :")  # coordonnées des points`
	`yb=input("Saisissez l'ordonnée du point B :")  #`
	`D=(xb-xa)**2+(yb-ya)**2  # La puissance est notée **`
	`print 'AB^2 = ',D  # Calcul exact (grand entier)`
	`print 'AB =',sqrt(D)  # Calcul approché`

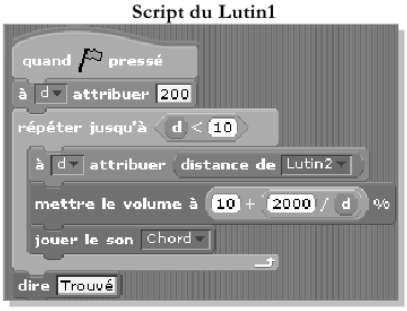
Σ	La structure de contrôle est la formule, donnée, de la distance : $d = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$. On peut penser que, dans le cas de l'utilisation d'un langage de programmation, la compilation et l'exécution joueront un rôle de contrôle.

Le programme-papier se réfère au paradigme AI, on n'a donc que la conception AI-outil. De plus, l'algorithme de la distance n'est qu'une traduction d'une formule de géométrie, l'enjeu se concentre donc uniquement sur le travail de reformulation dans un autre langage. Ce type d'algorithme ne permet pas une approche du point de vue objet.

L'autre activité proposée autour de la distance se place dans le logiciel Scratch dans lequel la fonction distance est déjà implémentée. L'activité propose un *script* pour jouer à chercher un objet caché sur l'écran en déplaçant la souris, en étant guidé par un son dont le volume augmente au fur et à mesure que l'on s'approche de l'objet :

L'exemple suivant utilise ces techniques pour moduler le volume sonore en fonction de la distance à un objet caché (le « trésor »). L'objet caché (ici nommé Lutin2) est une simple croix matérialisant un emplacement (un point, en somme) et le script associé est simplissime :

Script du Lutin1



Script du Lutin2 (trésor)

Ce programme n'est pas un algorithme au sens où nous l'avons décrit : il ne résout pas un problème et sa terminaison n'est pas claire. Il s'agit plutôt d'une activité de programmation d'un jeu ou de la simulation d'un détecteur de métaux. On voit bien ici l'amalgame entre programme et algorithme.
L'activité de l'élève concernant ce programme consiste à le modifier et le comprendre. Nous considérons que l'on est dans une activité de programmation qui est bien au-delà du paradigme AI, étant donné que la notion d'algorithme n'est plus présente.

• La *dichotomie* (p.25) : La dichotomie est présentée dans le cadre de la *recherche d'un zéro* d'une fonction. Une première activité, le *jeu du nombre à deviner* est proposée pour introduire la dichotomie.
Dans ce *jeu du nombre à deviner*, il s'agit de trouver un nombre compris entre 10 et 100, l'ordinateur répondant "c'est plus" ou "c'est moins" aux propositions du joueur.
Un premier programme-papier est proposé, il simule le joueur qui choisit le nombre. La programmation du jeu est proposée comme une première activité algorithmique. Ensuite l'élève doit jouer contre le programme et essayer de gagner à tous les coups, c'est-à-dire en moins de 6 essais. À ce niveau, l'algorithme en jeu est la stratégie de dichotomie qui doit émerger du côté du joueur qui devine.

P	Le problème peut être reformulé ainsi. Instances : le choix d'un nombre gagnant entre 10 et 100 Question : Trouver le nombre en moins de 6 essais.
R	La stratégie du joueur qui cherche le nombre (l'élève) : une règle qui dit quel nombre tester selon les réponses précédentes du programme.
L	Cette stratégie est surement attendue en langage courant, étant donnée qu'elle est formulée (et justifiée) de manière très informelle : Une bonne stratégie conduit à l'algorithme utilisant la dichotomie. Cette méthode consiste, en choisissant à chaque fois la valeur située au milieu de l'intervalle en cours, à réduire de moitié à chaque fois l'amplitude de l'intervalle dans lequel se trouve le nombre et comme 2^6 est égal à 64, le dernier intervalle, sur cet exemple, est d'amplitude 1.
Σ	La citation ci-dessus montre aussi la justification de la méthode. La réussite systématique face à l'ordinateur constitue un autre contrôle de l'algorithme. Notons que la preuve de l'optimalité de la dichotomie n'est pas faite, ni en général, ni dans ce cas particulier de l'intervalle de 10 à 100 car il faudrait aussi prouver que 5 coups ne suffisent pas toujours. Notons aussi que l'argument de validité de l'algorithme est très peu détaillé.

On trouve ici la conception AM-outil avec une structure de contrôle de l'ordre de la preuve mathématique. Est-ce que la conception AM-objet est aussi présente ? Peu de questions sur l'algorithme lui-même sont posées, il est seulement dit :

Le choix des valeurs 10 et 100 qui encadrent le nombre à trouver, ainsi que le nombre d'essais est à mettre en débat dans la classe. En effet, selon le choix de ces valeurs, il sera ou non possible de déterminer à coup sûr la solution avec une bonne stratégie, ou on pourra seulement optimiser les chances de gagner.

Nous dirons que le potentiel de l'algorithme de dichotomie pour vivre en tant qu'objet est soulevé, mais aucun opérateur n'est mis en avant pour étudier cet algorithme. On ne peut pas vraiment dire que la conception AM-objet est présente.
Relevons un autre point : le programme de jeu n'a pas d'utilité réelle dans cette activité, étant donné que des élèves pourraient jouer l'un contre l'autre. Programmer un adversaire virtuel est aussi considéré ici comme une activité algorithmique (cela relève encore de l'amalgame algorithme-programme).

La deuxième activité sur la dichotomie est la recherche d'un zéro d'une fonction par dichotomie (voir tableau ci-dessous). On est toujours dans la conception AI-outil et la validation

P	Instance : une fonction f qui change de signe entre a et b Question : résoudre $f(x) = 0$
R	Les opérateurs sont encore les instructions définies par le *pseudo-code*.
L	On retrouve le *pseudo-code* : **Variables** m , valeur milieu de l'intervalle « courant » **Initialisation** a et b, les bornes de l'intervalle [a ; b] *f*, la fonction (rappel : *f* change de signe entre *a* et *b*) **Traitement** **Pour** i **variant de** 1 **à** 50 m **prend la valeur** (a+b)/2 **Si** *f*(m) **et** *f*(a) **sont de même signe alors** a **prend la valeur** m **sinon** b **prend la valeur** m **Sortie** **Affiche** a et b
Σ	La validité de la méthode n'est pas questionnée. D'ailleurs il n'est même pas précisé que la fonction s'annule au moins une fois (parce qu'elle est continue ou qu'on a d'autres hypothèses). La structure de contrôle qui semble valider l'efficacité de la méthode pour trouver le zéro est l'activité précédente, faisant émerger la dichotomie : *On transpose cette méthode ici.* Notons que cette méthode n'a été validée comme optimale que pour une recherche parmi un intervalle de 91 entiers.

de l'algorithme est absente.
La conception objet, qui pourrait apparaître ici, étant donné la richesse de la situation, n'est touchée qu'au travers d'une remarque sur la précision de la méthode en fonction du nombre d'itérations.

• La *coïncidence des dates d'anniversaire* (p.30) :
Il s'agit d'étudier la probabilité que deux personnes aient leur anniversaire le même jour parmi un groupe de 30 personnes. Cette activité ne relève pas de l'algorithmique. On peut considérer, au mieux, que la conception AI-outil est en jeu (voir tableau ci-dessous). Nous reviendrons sur ce point un peu plus loin : les programmes de simulations ne relèvent pas, selon nous de l'algorithmique.

Un concept pour l'analyse : l'algorithme-instancié

Nous avons évoqué plus haut, l'exemple d'un algorithme qui était décrit dans un cas particulier. Il nous semble utile pour la suite de donner un nom à ce genre d'objet. Nous

P	L'enjeu est de simuler un grand nombre de fois une situation aléatoire. Il ne s'agit pas d'un problème de $\mathscr{P}_{\text{A}}$.
R	les opérateurs du programme-papier ou de langages de programmation : instructions décrites précédemment mais aussi générateur aléatoire de nombres.

L	Pseudo-langage du document et langage du logiciel Scilab :
	Variables
	dates : tableau des trente jours d'anniversaire
	trouvé : un booléen qui indique si deux dates coïncident.
	k, p: deux compteurs de boucles.
	Initialisation
	Pour k **de** 0 **à** 29
	dates[k] **prend une valeur** entière aléatoire comprise entre 1 et 365 inclus
	trouvé **prend la valeur** faux
	Traitement
	Pour k **de** 0 **à** 28
	Pour p **de** k+1 **à** 29
	Si dates[k] = dates[p] **alors**
	trouvé **prend la valeur vrai**
	Sortie
	Affiche trouvé
	Si le tirage au sort des dates se fait aisément sur tableur, il n'en va pas de même de la recherche de dates identiques (du fait des deux boucles).
	Traduction SCILAB :
	`dates=floor(rand(30,1)*365+1);`
	`trouve=%F;`
	`for k=1:29`
	`  for p=k+1:30`
	`    if (dates(p)==dates(k))`
	`      trouve=%T;`
	`    end;`
	`  end;`
	`end;`
	`if (trouve)`
	`  disp("Deux personnes ont même anniversaire");`
	`else`
	`  disp("Pas deux anniversaires communs");`
	`end;`
	Traduction SCILAB (1000 expériences) :
	`N=0;`
	`for i=1:1000`
	`  dates=floor(rand(30,1)*365+1);`
	`  trouve=%F;`
	`  for k=1:29`
	`    for p=k+1:30`
	`      if (dates(p)==dates(k))`
	`        trouve=%T;`
	`      end;`
	`    end;`
	`  end;`
	`  if (trouve)`
	`    N=N+1;`
	`  end;`
	`end;`
	`disp("Il y a "+string(N)+" coïncidences");`

Σ	Aucune structure de contrôle n'est présentée. Comme il s'agit d'une simulation, la seule activité est en fait une activité de programmation et les structures de contrôle ne peuvent pas être de celles que nous avons repérées pour l'algorithmique.

proposons le terme d'**algorithme-instancié**.

Nous appellerons algorithme-instancié, l'expression d'un algorithme dont les variables représentant l'entrée ont été remplacées par des valeurs précises. Un algorithme-instancié est à mi-chemin entre un algorithme et son exécution sur une instance. Le langage utilisé pour l'exprimer n'a pas d'importance.

Un algorithme-instancié n'est pas un algorithme, au sens où il ne répond pas à une question pour une famille d'instances. Un expert, c'est-à-dire une personne connaissant l'algorithme en jeu, reconnaîtra facilement l'algorithme derrière le cas particulier traité. Cependant, un novice ne verra pas la différence avec un algorithme erroné (au sens où il ne donne pas toujours la bonne réponse à la question), instancié sur une des instances où il est valide. Par exemple, pour le problème d'épargne (p.12) évoqué juste au-dessus [10] :

```
Mettre 5000 dans S
Mettre 0 dans N
Tant que S est strictement inférieur à 8000
    Remplacer S par S×1,02
    Augmenter N de 1
Fin du Tant Que
Afficher N
```

```
Mettre 5000 dans S
Mettre 0 dans N
Tant que S est strictement inférieur à 8000
    Remplacer S par S×(1,04)
    Augmenter N de 2
Fin du Tant Que
Afficher N
```

10. Nous reprenons volontairement le pseudo-code de l'exemple issu du document.

Comment différencier l'algorithme-instancié "valide" de gauche de l'algorithme-instancié "erroné"[11] de droite ? Si l'on cherche à prouver qu'ils répondent au problème posé (qui n'a qu'une instance), ils sont tous les deux corrects : ils donnent la solution de 24 semestres[12].

C'est donc bien le fait de répondre correctement à toutes les instances du problème qui doit être au centre du concept d'algorithme. D'un point de vue didactique, il est donc important de repérer ces algorithmes-instanciés. Si l'on rencontre des algorithmes-instanciés dont la généralisation ne fait pas référence à un algorithme correct, nous utiliserons le terme d'algorithme-erroné-instancié.

Problèmes et algorithmes

Cette partie du document présente une trentaine d'activités. Toutes ne mettent pas en jeu des problèmes de $\mathbb{P}$ ou de $\mathscr{P}_{\text{A}}$. Notamment, une grande partie repose sur la simulation de jeux, de systèmes automatiques ou d'expériences aléatoires. Bien que ces simulations puissent inclure de réels problèmes algorithmiques, ils ne sont qu'évoqués dans les activités.

Globalement, les activités de simulation relèvent de la programmation ou de l'automatisme (modéliser un digicode, simulation du déplacement d'un curseur avec le clavier, simulation d'un grand nombre de parties d'un jeu de hasard, etc.). Leur présence sans distinction au cœur d'autres problèmes d'ordre algorithmique révèle encore un amalgame programme-algorithme.
Par exemple, concernant l'activité de reproduire le fonctionnement d'un distributeur de billets ou du débit d'une carte de self (qui ne relève que de la simulation par la programmation), on retrouve pêle-mêle des questions d'interface (*choix multiples ou ouvert de la somme à retirer*, *impression ou non de la facturette*), des actions très simples (*vérification de la date de validité, du solde du compte, du retrait maximal autorisé*) et des problèmes de $\mathbb{P}$, au potentiel algorithmique et mathématique non exploités : *comment fournir cette somme en fonction du nombre de billets encore disponibles dans le distributeur*[13].
Plusieurs autres activités ne mettent pas en jeu l'algorithme mais uniquement la notion de programme (en particulier celles utilisant le logiciel Scratch, comme celle détaillée plus haut sur la notion de distance).

Même les problèmes de $\mathscr{P}_{\text{A}}$ et les algorithmes associés sont motivés par une question de modélisation. Par exemple, la recherche du maximum d'une fonction sur un intervalle ou du point le plus haut dans un plan, sont introduits à travers la question de trouver un fenêtrage adapté à une représentation graphique. Cela cache le problème. Il devient difficile de savoir quel est le problème traité (cela change aussi la question de la structure de contrôle et de la validité de la solution).

D'autres problèmes, bien qu'appartenant à $\mathbb{P}$, ne consistent qu'en la traduction d'une méthode ou d'une formule dans un langage de programmation pour rendre systématique leur utilisation. C'est surtout le cas des problèmes posés en géométrie (recherche du milieu d'un segment, du quatrième sommet d'un parallélogramme, calcul de la distance entre deux points, test pour déterminer si un triangle est isocèle).

11. Basé sur l'erreur classique : deux semestres à 2% correspondent à une année à 4%.
12. Tout comme l'algorithme : Afficher 24.
13. Le problème de former une somme en utilisant le moins de billets ou de savoir si une somme peut être formée avec un système monétaire donné est une question extrêmement riche et encore vive.

Une partie des activités est mise en avant pour d'autres raisons que l'algorithme lui-même. Les algorithmes ou les programmes proposés visent souvent à faire découvrir une notion (le ppcm, ou la décomposition en facteurs premiers, le cercle comme ensemble des points à distance constante du centre, la médiatrice, etc.), à permettre l'étude expérimentale d'objets mathématiques (comportement asymptotique et sens de variation d'une fonction, probabilité d'un événement) ou encore de motiver l'introduction de nouvelles instructions (les boucles par la nécessité de répéter un grand nombre de fois une tâche, etc.).

Modélisation-simulation

À ce point de la discussion, il nous semble utile d'introduire une nouvelle définition, les **programmes de modélisation-simulation**, étant donnée leur grande présence dans le document.

Nous appellerons programme de modélisation-simulation, toute expression d'une méthode systématique visant à simuler un phénomène ou un modèle donné de ce phénomène. Il peut s'agir de simuler un jeu, une machine, une expérience, un modèle de comportement d'un appareil, etc. Le problème des anniversaires étudié plus haut en donne une bonne illustration.

Les programmes de modélisation-simulation, ne sont pas des algorithmes selon notre définition. Ils ne résolvent pas un problème, ils n'ont parfois même aucune entrée. Leur rôle est souvent de pouvoir simuler avec des outils informatiques, dans le cadre d'une démarche expérimentale. Nous les appelons programmes plutôt que méthodes pour insister sur le fait qu'ils ont pour but une programmation effective, mais ils ne sont pas nécessairement exprimés dans un langage de programmation.

Questions et discussions

La grande majorité des questions posées dans cette partie du document relève de l'écriture d'algorithmes, de programmes-papiers ou de programmes (*rédaction d'une méthode*, *comment programmer une machine pour que [...] ?*, *programmation effective de l'algorithme*, *Modifier le programme pour que [...]*, etc.) ou de leur interprétation et de la compréhension des instructions (*écrire les affichages successifs*, *quelle est la position initiale de l'objet ?*, *analyser l'algorithme a priori et prévoir la figure qui sera obtenue*, etc.). Assez souvent la question est simplement celle à laquelle répond l'algorithme.
On est donc dans une conception de l'algorithme comme outil, lorsqu'il y a vraiment un algorithme.

D'autres questions, plus rares, sont posées sur les algorithmes eux-mêmes. Quelques questions et remarques concernent l'exactitude des calculs effectués (par exemple lorsqu'on utilise la fonction racine carrée) mais dans ce cas ce n'est pas l'algorithme lui-même qui est questionné mais le logiciel ou programme utilisé. Les questions d'approximation dans les algorithmes sur les fonctions (recherche de zéro, recherche d'un maximum, etc.) pourraient relever d'une conception objet de l'algorithme.
On trouve aussi une question de comparaison d'algorithmes entre la recherche d'un maximum par balayage et la recherche aléatoire :

> La comparaison des résultats des deux algorithmes avec un même nombre d'itérations pourra donner lieu à des échanges autour de la notion de hasard et de son efficacité selon les cas. (p.23)

Opérateurs et système de représentation

On trouve dans les activités proposées trois types de systèmes de représentation :

- Le *pseudo-code* décrit dans le document : la grande majorité des algorithmes et programmes qui sont présentés sont d'abord donnés avec ce pseudo-code (tous sont sous la forme de programmes-papiers).
- Des traductions des méthodes dans un ou plusieurs langages de programmation (Casio, TI, Xcas, Linotte, Python, Scilab) sont ensuite données. Parfois, on a directement le langage de programmation.
- Un *langage naturel* (rare) qui sert à exprimer les méthodes de manière *informelle*. Ce langage naturel reste très proche du langage de programmation puisqu'il prend aussi en charge affichage et saisie d'information :

  Algorithme en langage naturel

  Saisir les coordonnées des points.
  Calculer les longueurs AB et AC.
  Si $AB^2=AC^2$ alors afficher que le triangle est isocèle en A,
  sinon afficher qu'il ne l'est pas.

Les opérateurs sont essentiellement les instructions de ces langages, c'est pourquoi nous avons choisi d'associer opérateurs et systèmes de représentation dans un même paragraphe. L'algorithmique est considérée ici comme une activité essentiellement langagière (ce qui peut expliquer que programmes et algorithmes ne soient pas distingués). D'ailleurs on comprend clairement que le *langage naturel* n'est qu'une étape qui doit laisser la place aux *pseudo-code* et langages de programmation :

> Dans ces premières activités, la formulation sera probablement lourde en comportant des structures de la forme : « si tu es dans telle situation alors fais cela, sinon » ou « tant que ceci se produit on continue à faire, etc. ». Cette lourdeur pourra alors fournir une motivation pour la mise en place d'une syntaxe commune sur des éléments d'algorithmique récurrents : boucles, tests... Cette syntaxe commune peut être un premier pas vers de premières rencontres avec des langages dédiés. Au cours de ce processus on pourra mettre en évidence le besoin de formalisation et de rigueur. (p.10)

Structures de contrôle

La validation par la preuve d'algorithme est quasiment absente de cette partie du document. Comme annoncé :

> Toutes ces activités ont pour objectif la production ou l'interprétation d'algorithmes et leur vérification obtenue par l'action immédiate et pas nécessairement issue directement du champ mathématique. (p.10)

Pourtant, dans les critères d'évaluation :

> Se limiter à un simple critère de « fonctionnement » n'est pas satisfaisant : certains algorithmes de piètre qualité « fonctionnent » tandis que d'autres ne « fonctionnent » pas mais pour des causes tout à fait mineures. (p.13)

En réalité, le contrôle de l'action s'effectue majoritairement par l'exécution des algorithmes ou des programmes et par le respect des règles du langage.

D'autres fois, c'est la validité de la méthode présentée sous forme d'algorithme qui valide l'algorithme lui-même (c'est le cas pour l'écriture de formules ou de constructions géométriques sous forme d'algorithmes). Nous avions noté, dans le cas de la dichotomie, quelque

chose d'un peu équivalent. On valide une méthode dans un cas particulier (chercher un nombre entre 10 et 100 en moins de 6 coups) et on peut ensuite utiliser cette méthode dans un cas plus général.
On peut noter cependant quelques structures de contrôle d'ordre mathématique. Par exemple, un invariant est donné en indication pour surveiller la bonne exécution d'un programme ou une discussion est développée sur le fait que le test de monotonie (par balayage) permet de prouver qu'une fonction n'est pas monotone mais pas qu'elle l'est.
La comparaison des méthodes de recherche du maximum d'une fonction, question de $\mathscr{P}_{\text{A}}$, met en jeu une structure de contrôle par des exemples :

> La comparaison des résultats des deux algorithmes avec un même nombre d'itérations pourra donner lieu à des échanges autour de la notion de hasard et de son efficacité selon les cas.
>
> Par exemple, avec la fonction carré sur l'intervalle $[-1;1]$, on obtient une valeur approchée du minimum à 10^{-2} près dès que le nombre tiré au hasard appartient à l'intervalle $[-0{,}1;0{,}1]$, ce qui se produit avec une probabilité de 0,1. On devine qu'avec 50 itérations, il y a une grande probabilité que cela se réalise au moins une fois. Plus précisément, cette probabilité vaut $1-0{,}9^{50}$ qui est supérieur à 99%.
>
> L'algorithme déterministe à pas constant donnera 10^{-4} comme minimum de la fonction carré sur l'intervalle $[-1;2]$ avec 100 itérations. Un raisonnement identique montre que l'algorithme aléatoire, donnera quant à lui une meilleure valeur approchée du minimum plus d'une fois sur deux.

Amalgame programmation-algorithmique

L'amalgame entre programmation et algorithmique est toujours très prononcé dans cette partie du document. Nous avons relevé quelques exemples plus-haut. Cependant nous souhaitons ajouter quelques indicateurs forts de cela.

Dans le jeu des multiples :

> Description du jeu : les élèves sont placés en rond. L'un après l'autre, les élèves égrènent les entiers naturels . La difficulté réside dans le fait que si l'entier est par exemple un multiple de 3, l'élève annonce « fizz », si c'est un multiple de 5, il annonce « buzz », etc.
>
> On peut proposer aux élèves d'écrire une règle du jeu à faire tester à leurs camarades : comment programmer une machine pour qu'elle réponde correctement si on l'incluait dans le jeu ?
>
> La question de l'arrêt du jeu se pose : fixer un entier maximal ? Arrêter quand il ne reste plus qu'un seul participant ?

L'activité consiste à proposer un algorithme qui peut répondre à la place du joueur (ce qui soulève des questions d'algorithmique pour l'arithmétique). Puis, se pose la question de l'arrêt du jeu, comme s'il s'agissait d'une question de terminaison. Il semble que se mélangent ici la simulation d'une partie et l'algorithme qui permet de répondre.

Enfin, les critères d'évaluation (grille ci-dessous) des algorithmes sont uniquement des

> La grille ci-dessous peut donner quelques éléments en ce sens, non limitatifs il va de soi, et on adaptera ce questionnement aux situations effectivement rencontrées.

Critère	Excellent	Bon	Moyen	Insuffisant
Respect des bons usages Le but visé par l'algorithme est explicité, et des commentaires précisent le déroulement. Les variables ont des noms bien choisis.	Aucune erreur	De petits détails sont négligés. Le but est difficile à déterminer	Des détails manquent, mais le programme tente quand même d'accomplir ses fonctions essentielles.	Ne répond pas au problème posé. Objectif impossible à déterminer
Correction du code : L'algorithme fonctionne.	Fonctionne correctement dans **tous** les cas.	Fonctionne pour des données (entrées) standard mais échecs mineurs sur des cas particuliers.	Échoue pour des données (entrées) standard, mais pour une raison mineure.	Échoue pour des données (entrées) standard, pour une raison importante.
Interface utilisateur : (entrées, sorties) Elle est claire et commode.	Aucune faute	1-3 fautes mineures	Plus de trois fautes mineures ou une faute majeure	Plus d'une faute majeure

indicateurs concernant la programmation. Cela inclut même une évaluation des bonnes pratiques d'écriture de programmes.

5.3 Bilan

Résumons les points importants relevés dans le document *Ressources pour la classe de seconde* :

- L'activité algorithmique est centrée autour du langage. Elle se concentre sur l'expression rigoureuse d'opérations, avec pour but l'écriture de programmes. Le langage joue alors à la fois le rôle de système de représentation, d'opérateur, et parfois même de structure de contrôle. Si le langage caractérise l'algorithmique, alors on ne peut pas distinguer l'algorithme des programmes-papiers, algorithmes-instanciés et programmes de modélisation-simulation.
- Une conséquence de cela est un amalgame très marqué entre algorithme et programme.
- L'algorithme est uniquement outil et la conception dominante est AI-Outil. La conception AM-Outil ne peut pas vraiment être considérée comme présente (en particulier, par la présence permanente des programmes papiers).
- L'algorithme n'est jamais considéré comme un objet, la majorité des questions posées sur les algorithmes relèvent du langage.

Cette section nous a aussi amené à définir trois objets :

– les programmes-papiers (caractéristique de AI),

– les algorithmes-instanciés (qui ne sont pas des algorithmes),

– les programmes de modélisation-simulation (qui relèvent de l'outil de programmation).

La présence répétées de ces objets nous a poussé à les identifier et il sera nécessaire de savoir si on les retrouve dans les autres ressources étudiées.

Conclusion

Deux transpositions différentes

Ce chapitre montre la transposition à l'œuvre dans deux institutions différentes : l'institution des instructions pour l'enseignement mathématique au lycée et celle de l'informatique et sciences du numérique. Nous avons mis en lumière deux transpositions très différentes du concept d'algorithme.

On peut souligner trois différences fondamentales entre ces deux transpositions :

	Mathématiques	ISN
Relation algorithme-programme	Amalgame entre algorithme et programme.	Algorithme et programme sont distingués et définis.
Relation algorithme-langage	L'algorithmique est une activité centrée sur le langage	L'algorithme est une méthode qui nécessite un langage pour être exprimée.
Statut de l'algorithme	L'algorithme est un outil. Il n'est pas clairement défini.	L'algorithme est à la fois un outil et un objet : il est clairement défini est il peut être l'objet de questions.

Transposition dans les mathématiques

La transposition en jeu dans les programmes de mathématiques et les documents-ressources montre une grande distance avec le savoir savant que nous avions modélisé avec les μ-conceptions. Les conclusions de l'analyse du document ressource confirment celles des programmes et les renforcent. Les hypothèses que nous avions faites sur certains points des programmes se trouvent aussi validées. En particulier en ce qui concerne l'amalgame algorithme-programme et ce que nous avons appelé les programmes-papiers.

Les documents-ressources ne concernent que la seconde et l'on pourrait se demander si nos conclusions vont aussi concerner les classes de premières et de terminales (en particulier scientifiques). Une réponse à cette question peut être donnée par l'analyse des algorithmes présents dans les sujets du baccalauréat 2012, série S. En annexe C, nous avons regroupé les algorithmes présents dans les différents sujets. On y retrouve des programmes papiers, des programmes de modélisation-simulation et un algorithme-instancié.

Retour sur les questions de recherche

Ce chapitre nous a permis de répondre à la question $Q5_{\text{conceptions}}$:
Quelles conceptions sont représentées dans les programmes de mathématiques et documents d'accompagnement du lycée ? Les conceptions présentes sont-elles complètes ? Sont-elles modifiées par rapport à celles du savoir savant ? Relèvent-elles de l'outil ou de l'objet ? Quelle place est donnée aux structures de contrôle ? Qu'en est-il dans les programmes de la spécialité ISN ?

Les outils que nous avons mis en œuvre nous ont permis de répondre à cette question de manière relativement précise. Nous avons montré leur utilisation dans l'étude de différents niveaux de discours (discours généraux sur l'algorithmique, sur les activités à mener en classe ou encore présentation détaillée d'activités spécifiques). Nous avons aussi pu mettre en évidence des différences de transposition grâce à notre outil. Cela était un de nos objectifs et l'analyse des conceptions des chercheurs avait montré ce potentiel. Cependant, la possibilité de le faire dans le cadre d'une transposition didactique constitue une validation de l'outil.

Nous poursuivrons l'étude de la transposition en tentant d'automatiser l'analyse, avec pour but la production d'une grille et d'un outil plus systématiques.

Chapitre 7

Ressources du site des IREM en algorithmique

Sommaire

1 Quelles ressources ? Quels outils ? 147
1.1 Les ressources en ligne des IREM 148
1.2 Outils d'analyse 149
2 Analyse 150
2.1 Illustration : différents traitements de la dichotomie 150
2.2 Quels Aspects dans les ressources ? 164
2.3 Quelles conceptions dans les ressources 168
2.4 Quels algorithmes dans les ressources ? 173
2.5 Classification des documents 174
Conclusions 174
2.6 Résumé des résultats 174
2.7 Interprétations 175

1 Quelles ressources ? Quels outils ?

Les motivations à analyser d'autres ressources que les manuels et les programmes et documents d'accompagnement sont nombreuses.

Tout d'abord, les pratiques enseignantes intègrent largement les ressources en ligne en complément des manuels.
Suite à l'introduction rapide de l'algorithmique, les manuels n'ont pas pu proposer tout de suite de contenu d'algorithmique. Les enseignants ont donc dû les chercher ailleurs. Au sein des IREM, de nombreux groupes *algorithmique* se sont créés dès la rentrée 2009 et ont produit des ressources pour les enseignants.
Ensuite, on peut supposer que l'algorithmique ne fait généralement pas partie de la formation des enseignants de mathématiques. Tout du moins, les résultats du chapitre 5 concernant les conceptions de chercheurs nous laissent penser que l'on retrouve aussi une connaissance partielle de l'algorithme chez les enseignants. Il est donc important de s'intéresser aussi aux ressources permettant de se former à l'algorithmique. La formation des enseignants à l'algorithmique est d'ailleurs, dans beaucoup de cas, l'un des objectifs des groupes *algorithmique* des IREM.

Enfin, l'algorithmique n'existait quasiment pas dans l'enseignement avant cette introduction. On peut dire qu'aucune ressource ne pré-existait à cette introduction. Par conséquent, une forte part des ressources reste, pour l'instant, des ressources en ligne.

Notre choix s'est donc porté sur l'étude des ressources en ligne proposées par le site des IREM pour plusieurs raisons :

- Ce sont des ressources en ligne accessibles à tous les enseignants.
- Le réseau des IREM représente une institution universitaire connue des enseignants et disposant, en tant que telle, d'une légitimité et d'un crédit auprès des enseignants (nous faisons en tout cas cette hypothèse).
- Le site propose notamment deux types de documentation : des activités pour la classe et des documents de formation à destination des enseignants.
- L'ensemble des ressources disponibles sur un site donné constitue un corpus bien délimité. Construire un corpus de ressources en ligne de diverses origines demanderait une analyse très poussée des pratiques des enseignants et de leurs sources privilégiées.
- Ces ressources sont conçues pour la grande majorité par des enseignants en collaboration avec des universitaires (mathématiciens et/ou informaticiens). On peut supposer que ce qu'on retrouvera dans ces ressources, du point de vue des conceptions de l'algorithmique et des aspects en jeu, se retrouvera dans une majorité d'autres ressources produites par des enseignants.
- Ces ressources sont validées par le réseau des IREM : il y a une forme de relecture des contenus proposés. Comme le stipule la rubrique "algorithmique" du site des IREM : « Les documents cités de cette rubrique ont fait l'objet d'une lecture par l'un des membres du comité d'édition suivant : [...] ». Quant à la rubrique "évolution au lycée", elle est gérée par un groupe de travail inter-IREM qui sélectionne les documents proposés.
- Une dernière raison est que les travaux des IREM sont directement cités et recommandés dans le document d'accompagnement *Ressources pour la classe de seconde - Algorithmique* :

  > La sensibilisation de l'élève à la « démarche algorithmique » pourra se faire en évitant toute technicité ou exposé systématique. On pourra sur ce thème consulter les publications réalisées dans le cadre des IREM. (p. 3)

1.1 Les ressources en ligne des IREM

Les ressources en algorithmique des IREM se décomposent suivant plusieurs rubriques et sous-rubriques. Les rubriques qui concernent l'algorithmique sont les suivantes :

- La rubrique *algorithmique* et toutes ses sous-rubriques,
- La rubrique *évolution au lycée* qui traite des nouveautés des programmes concernant l'algorithmique et la logique.

L'annexe A montre l'arborescence des documents présents sur le site des IREM, on peut y voir que certains documents sont présents dans plusieurs rubriques. D'autres documents ont été retirés : les liens vers d'autres sites de ressources et les documents officiels (programmes, documents ressources Eduscol, réactions de l'ADIREM, etc.). À un certain niveau d'analyse nous nous autoriserons à mettre de côté d'autres documents : les ressources trop peu détaillées (informations trop partielles) ou les ressources "hors-propos" (savoir que ce genre de ressource est classifié avec l'algorithmique est une information intéressante mais cela

n'a pas de sens de l'étudier avec nos outils).

Notre corpus est finalement constitué d'une trentaine de documents. Ces documents sont de natures et de formats variés : en particulier on y trouve des activités précises proposées pour la classe, des discussions générales sur l'algorithmique au lycée ou encore des documents de formation pouvant contenir à la fois des discours généraux, des exercices et des activités utilisés en formation d'enseignants et des activités pour la classe. Il faudra tenir compte de cette diversité de contenus et de longueur des documents dans l'analyse.

1.2 Outils d'analyse

Nous proposons une analyse de ces documents selon les outils d'analyse développés dans la partie précédente. Il s'agit donc d'étudier quels aspects sont en jeu dans ces documents et s'ils réfèrent à l'outil ou à l'objet algorithme. Nous souhaitons aussi étudier quels sont les paradigmes dans lesquels se placent les ressources : Preuve Algorithme, Algorithme Mathématique, Algorithme Informatique ? Et quelles conceptions de l'algorithme sont en jeu dans ces ressources.
Il faut bien entendu mettre cela en relation avec le type de ressources étudié, les objectifs et le public visé. Bien sûr, il est important de se pencher sur les algorithmes étudiés et les domaines dans lesquels ces algorithmes sont choisis.

Pour chaque ressource, nous avons donc suivi la grille d'analyse suivante :

- Type de ressource :
  - Quelle forme de ressource ?
  - À qui se destine-t-elle ?
  - Quels objectifs sont affichés ?
- Algorithmes en jeu :
  - Quels problèmes mathématiques sont proposés ?
  - Sont-ils des problèmes au sens de notre définition ? (instanciés ?)
  - Quels sont les algorithmes présentés ou attendus ?
  - Dans quel domaine se situe-t-on ?
  - Quels rôles jouent les algorithmes dans l'activité ?
- Questions :
  - Quelles questions sont posées concernant ces algorithmes ?
  - Quelles problématiques sont soulevées ?
- Aspects :
  - Quels aspects sont mis en avant ?
  - Relèvent-ils de l'outil ou de l'objet ?
- Conceptions :
  - Dans quel(s) paradigme(s) se trouve-t-on ? (en lien avec le système de représentation)
  - Quelles conceptions de l'algorithme sont présentes ? Et plus particulièrement : quels types de problèmes ? quels types d'opérateurs ? quels systèmes de représentation ? et quels types de structures de contrôle (si elles sont présentes) ?

– Concernant les structures de contrôle, qui a la responsabilité de ces structures ? (notamment dans les activités pour la classe)

Cette grille est très proche de celle utilisée pour les instructions officielles. Nous avons vu qu'elle était adaptée à la fois au discours sur l'algorithmique, aux descriptions des contenus et aux activités proposées par le document-ressource. Chaque point de la grille s'est avéré plus ou moins adapté et plus ou moins révélateur selon le type de contenus. Cela contribue à la souplesse de l'outil, qui devrait donc s'adapter facilement à la diversité des ressources du site des IREM.

2 Analyse

2.1 Illustration : différents traitements de la dichotomie

Avant de présenter l'analyse complète du corpus, nous présentons ici en détail l'analyse de quelques documents ou extraits de documents. Cela permet de détailler la mise en œuvre des outils d'analyse. Nous avons choisi, pour cette illustration, un même thème que l'on retrouve dans plusieurs documents : la dichotomie. La dichotomie est clairement liée à l'algorithmique et l'on peut voir que son traitement peut être divers dans les ressources.

Le choix de s'intéresser au traitement de la dichotomie n'est pas anodin. La dichotomie est un type d'algorithme qui permet de résoudre des problèmes dans divers domaines et qui peut être un objet d'étude riche :

Problème La dichotomie résout des problèmes qui peuvent être exprimés selon notre modèle (Instances, Question). En particulier, pouvoir résoudre l'ensemble des instances du problème est clairement un enjeu de la dichotomie.

Preuve Un algorithme de dichotomie est souvent facile à décrire, par contre sa validité n'est pas toujours évidente.

Complexité La dichotomie n'est pas simplement un algorithme résolvant certains problèmes, il s'agit en général d'un algorithme rapide et l'étude de sa complexité logarithmique est caractéristique de l'idée de "diviser pour régner".

Optimalité Non seulement la dichotomie fournit une solution très efficace à un problème, mais dans la majorité des cas elle est la solution optimale en termes de complexité. C'est la comparaison de cet algorithme à d'autres (une recherche par balayage par exemple) qui lui donne son sens. Plus précisément, c'est le fait qu'il s'agit du meilleur algorithme en termes de complexité. La dichotomie peut permettre d'aborder ce genre de problématiques qui lui sont intrinsèques.

AM-AI-PA La dichotomie peut exister dans divers paradigmes : dans AI lorsque l'on fait du traitement de données, dans AM, par exemple lorsque l'on cherche des stratégies optimales pour un joueur, ou encore dans PA, par exemple lorsque l'on se pose la question d'un nombre d'étapes minimum pour trouver une solution à un problème (sans chercher nécessairement une stratégie).

La dichotomie constitue donc un sujet riche pour l'algorithmique (en tant qu'outil et qu'objet). Cela devrait nous permettre de voir comment sont transposés des concepts vraiment issus de l'algorithmique et comment nos outils permettent de le faire (deux choses que nous n'avions pas beaucoup pu faire dans l'étude des instructions officielles).

Les documents concernés sont les suivants :

– **A1 :** Méthode pour élaborer des algorithmes itératifs (IREM d'Aix-Marseille), section 1.4 : *La recherche dichotomique* (pp.7-13)

– **A3 :** Diviser pour régner (IREM de Lyon), section 2 : *Recherche dans une liste* (pp.2-3)

– **A8 :** Ouvrage de formation à l'algorithmique et à la logique (IREM de Marseille), section 1.4 : *Un principe fondamental : la dichotomie* (pp.14-17, 33-34)

– **A9 :** Introduction à l'algorithmique en classe de seconde (IREM de Montpellier), section 5.4 : *Itérer tant que : Le juste prix* et section 5.5 : *Itérer tant que : zéros par dichotomie* (pp.46-54)

– **A13 :** Jeu : « C'est plus, c'est moins » (IREM de Brest)

– **A21 :** Document d'accompagnement des stages de formation à l'algorithmique (IREM de Grenoble), section 7 : *Recherche dichotomique* (pp.33-36)

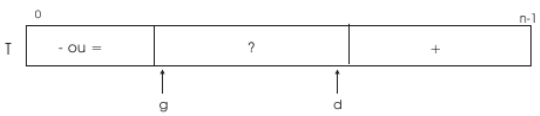
Document A1

Ce document est à destination des enseignants ou formateurs à la recherche d'approfondissements en algorithmique. Il présente une méthode pour concevoir des algorithmes itératifs valides (c'est-à-dire corrects et qui terminent) en liant conception d'un algorithme itératif et construction d'un invariant.
Concernant la dichotomie, le document propose plusieurs algorithmes qui diffèrent notamment en fonction du choix de l'invariant.
Le document met en jeu les conceptions Outil et Objet du paradigme AM, présentées et illustrées dans les deux tableaux suivants :

Conception AM-outil dans A1 (extrait dichotomie)

	Conception	Extraits du document (illustration)
P	Instance : un tableau T trié et un élément x Question : x est-il dans T ? Le problème appartient à $\mathscr{P}_{\text{A}}$	Soit $T[0..n-1]$ un tableau de n éléments classés par ordre croissant. Le problème consiste à établir un algorithme qui permette de dire si un élément x appartient ou pas au tableau T. L'algorithme recherché ne doit pas effectuer une recherche séquentielle, mais doit utiliser le principe de la dichotomie : [...]

R	Opérateurs de manipulation de types de données abstraites (le tableau), Opérations mathématiques, affectations, boucles et conditionnelles	$g \leftarrow 0$
		$d \leftarrow n-1$
		tant que $g \leq d$ **faire**
		  $k \leftarrow (g+d)$ **div** 2
		  si $T[k] \leq x$ **alors** $g \leftarrow k+1$ **sinon** $d \leftarrow k-1$
		si $d \geq 0$ et $T[d] = x$ **alors** " trouvé en d " **sinon** " pas trouvé "

L	Pour R : Langage mathématique mélangé à un langage proche des langages de programmation. Les objets manipulés sont des types de données abstraites et objets mathématiques munis de certaines opérations de base Pour Σ : schéma décrivant l'invariant et légende du schéma en langage naturel	Voir ci-dessus et ci-dessous
Σ	Les invariants et la méthode de construction d'algorithmes itératifs proposés au début du document sont les structures de contrôle qui valident l'algorithme	 où + indique la zone des éléments de T qui sont strictement plus grands que x et - ou = indique la zone des éléments inférieurs ou égaux à x.

Conception AM-objet dans A1 (extrait dichotomie)

	Conception	Extraits du document (illustration)
P	Instance : un problème p de $\mathscr{P}_{\text{A}}$ (instancié pour p : recherche d'un élément dans une liste triée) Question : Construire un algorithme valide (correct et qui termine) pour p ? Le problème appartient à $\mathbb{P}$	La démarche présentée tente, sans introduire un formalisme excessif, d'apporter des éléments de réponses aux questions fondamentales qui se posent en algorithmique : – Comment élaborer un algorithme ? – L'algorithme s'arrête-t-il toujours ? – Comment prouver qu'un algorithme résout effectivement le problème posé ?
R	La méthode de construction de l'algorithme ([1],[2],[3],[4])et l'invariant permettent de construire et valider l'algorithme.	Une fois cette étude conduite l'algorithme aura la structure suivante : [4] tant que non [2] faire [3] Cette façon de procéder montre comment on prouve la validité de l'algorithme au fur et à mesure de son élaboration. En effet la situation générale choisie en [1] est en fait l'invariant qui caractérise la boucle $tantque$. Cette situation est satisfaite au départ à cause de l'étape [4], elle reste vraie à chaque itération (étape [3]). Ainsi lorsque la condition d'arrêt est atteinte cette situation nous permet d'affirmer que le problème est résolu. C'est également en analysant l'étape [3] qu'on peut prouver la terminaison de l'algorithme.
L	Pour R : schéma décrivant l'invariant et légende du schéma en langage naturel Pour Σ : langage courant et langage mathématique	Voir ci-dessus et ci-dessous
Σ	Raisonnements et outils de logique mathématiques valident l'action des opérateurs sur l'algorithme. Par exemple, on remarque des énumérations de tous les cas à envisager pour valider l'algorithme.	L'étape [3] conserve de façon évidente la situation choisie en [1]. D'autre part, à chaque étape, soit g augmente, soit d diminue, la condition d'arrêt sera toujours atteinte. —— Tout semble parfaitement correct, et pourtant il y a une erreur. On n'est pas assuré que la condition d'arrêt soit atteinte dans tout les cas. En effet, dans le cas où d devient égal à $g+1$, $k \leftarrow (g+d)$ **div** 2 est égal à g et si le test $T[k] \leq x$ est satisfait l'instruction $g \leftarrow k$ ne permet pas à g d'augmenter et le programme boucle ! Pour être sûr de la terminaison, il faut écrire $g \geq d-1$ pour condition d'arrêt. Lorsque cette condition d'arrêt est atteinte il faut examiner attentivement toutes les situations finales possibles avant de conclure. —— Lorsque l'étape 3 est exécutée soit $g \leftarrow k$, soit $d \leftarrow k-1$, dans tous les cas soit g augmente strictement, soit k diminue strictement. La condition d'arrêt $g = d$ sera donc toujours atteinte.

Aspects de l'algorithme dans A1 (extrait dichotomie)

OUTIL	
Effectivité	- programmation : implicite, à travers le choix d'un langage très proche d'un langage de programmation et la structure de tableau - ordinateur/machine : idem - finitude : volonté de prouver la terminaison - non-ambiguïté : implicite, par le choix d'un langage proche d'un langage de programmation - agit sur des données finies : implicite par la manipulation de la liste de taille n
Problème	- entrée/sortie : implicite dans les algorithmes décrits. Exemple : « alors "trouvé en $g-1$" sinon "pas trouvé" » - Résout un problème pour toute instance : implicite « un algorithme qui permette de dire si un élément x appartient ou pas au tableau T »

OBJET	
Preuve	- Preuve de terminaison : c'est une des préoccupations centrales du document. - Preuve de correction : c'est une des préoccupations centrales du document. - Invariant : c'est l'outil clé du document pour valider les algorithmes
Complexité	La complexité est, dans cet extrait, à peine évoquée : « Pour rechercher un élément dans un tableau ayant un million d'éléments, au plus vingt comparaisons suffiront ». Cette remarque fait référence à la comparaison avec la recherche séquentielle mais la comparaison des algorithmes n'est pas du tout mise en avant.
Modèles théoriques	×

Le document se place globalement dans le paradigme AM mais il y a une forte proximité de AI avec des algorithmes exprimés dans un langage très proche d'un langage de programmation et des termes relatifs à la programmation ou aux machines comme « dans ce cas, accéder à $T[d]$ déclencherait une erreur. ».

Les aspects présents dans le document sont résumés dans le troisième tableau. On note la présence d'aspects autant du côté outil que du côté objet.

On retrouve bien l'aspect outil et l'aspect objet de l'algorithme. Les aspects d'invariants, de preuve de correction et de terminaison sont mis au centre du document. Cependant l'aspect COMPLEXITÉ est peu présent alors qu'il est l'un des points importants de la dichotomie (l'algorithme de recherche par dichotomie peut être abordé par la recherche de l'algorithme le plus efficace pour le problème proposé). La complexité de la recherche dichotomique n'est pas étudiée, la méthode n'est pas comparée à d'autres (notamment la recherche séquentielle) et la question de l'optimalité de la méthode pour le problème de la recherche dans une liste triée n'est pas du tout posée.

Document A3

Ce document est à destination des enseignants ou formateurs à la recherche d'approfondissements en algorithmique. Il introduit le principe algorithmique "diviser pour régner" au travers de plusieurs exemples.
Concernant la dichotomie, il propose la comparaison de deux algorithmes donnés pour le problème de recherche d'un élément dans une liste strictement croissante.
Le document met en jeu les conceptions Outil et Objet du paradigme AI, présentées et illustrées dans les deux tableaux suivants.

Conception AI-outil dans A3 (extrait dichotomie)

	Conception	Extraits du document (illustration)
P	Instance : une liste triée par ordre strictement croissant et un élément x Question : Quel est le rang de x dans la liste ? Le problème appartient à $\mathscr{P}_{\text{A}}$	On se propose de comparer les deux algorithmes suivants (langage Xcas) qui ont pour objectif de chercher le rang d'un élément dans une liste dont les éléments sont supposés triés dans l'ordre strictement croissant. Remarque 1 : les listes sont numérotées en Xcas de 0 à dim(liste)-1. Remarque 2 : pour alléger le code, on suppose que la valeur cherchée est effectivement dans la liste.

R	Opérateurs du langage Xcas.	Xcas
		`sequentiel(liste,valeur):={`
		`local rang;`
		`rang:=0;`
		`tantque liste[rang]<>valeur faire`
		`rang:=rang+1;`
		`ftantque;`
		`retourne rang;`
		`};;`

L	Langage Xcas	
Σ	$\times$	

Un autre problème est présent : étant donné un programme itératif, le réécrire en un programme récursif (instancié pour les deux algorithmes en Xcas donnés au départ). Cela pourrait entrer dans la conception AI-objet mais on ne peut pas détailler la conception car le document comprend trop peu de détails (on ne sait pas quels opérateurs sont utilisés ni

Conception AI-objet dans A3 (extrait dichotomie)

	Conception	Extraits du document (illustration)
P	Instance : un algorithme (sous forme d'un programme) A (instancié pour deux algorithmes : les recherches séquentielle et dichotomique d'un élément dans une liste triée en Xcas) Question : Combien au plus d'itérations de la boucle ont lieu pour une liste de taille 2^p ? Le problème appartient à $\mathbb{P}$	1. Combien, au plus, d'itérations de la boucle auront lieu lors d'une recherche pour chacun des deux programmes ? Pour simplifier on se placera dans le cas où le nombre d'éléments dans la liste est de la forme $n = 2^p$.
R	Opérateurs d'ordre mathématique (recherche du pire cas, étude de suites...)	(b) Pour la recherche dichotomique. Notons v_n le nombre maximum d'itérations. Pour $n = 1$, on a $v_1 = 1$. Pour $n = 2$, on a $v_2 = 2$. Pour $n = 2^2$, on a $v_4 = 3$ ... Pour $n = 2^p$, les sous-listes contiennent au plus 2^{p-1} éléments. On a donc $v_{2^p} \leqslant 1 + v_{2^{p-1}}$ (croissance de la suite (v_n)). On en déduit $v_{2^p} \leqslant p + v_1$ c'est à dire $v_n \leqslant 1 + \log_2(n)$. Pour n quelconque. On a : $n \leqslant 2^p$ où l'on a posé $p = \lceil \log_2(n) \rceil$. On a donc $v_n \leqslant v_{2^p} \leqslant 1 + p$. Soit $v_n \leqslant 1 + \lceil \log_2(n) \rceil$.
L	Pour R : langage mathématique et langage naturel	
Σ	Raisonnements et outils de logique mathématiques valident l'action des opérateurs sur l'algorithme.	On a donc [...] On en déduit [...] Pour n quelconque [...] On a donc

quelle structures de contrôle interviennent) mais on peut la classifier dans le paradigme AI.

La question de la complexité de la recherche dichotomique pourrait tout aussi bien être soulevée dans AM. La conception AI n'est pas liée ici au type de questions traitées mais peut-être plutôt à une prédominance de cette conception sur AM.

Les aspects présents dans le document sont résumés dans le tableau suivant. On note la présence d'aspects autant du côté outil que du côté objet.

Aspects de l'algorithme dans A3 (extrait dichotomie)

OUTIL	
Effectivité	- programmation : algorithmes exprimés dans le langage de programmation Xcas. - ordinateur/machine : idem - non-ambiguïté : implicite, par le choix d'exprimer les algorithmes dans un langage de programmation - agit sur des données finies : implicite par la manipulation du tableau de taille n
Problème	- entrée/sortie : implicite dans les algorithmes décrits par l'écriture sous forme de fonctions des programmes en Xcas : « séquentiel(liste,valeur) := » « dichotomique(liste,valeur) := », « retourne rang », « retourne k » - instance : implicite par l'étude de la complexité d'un problème.
OBJET	
Preuve	L'aspect PREUVE n'est présent que pour le calcul de la complexité des algorithmes.
Complexité	- Au pire : la complexité au pire en nombre de boucles est évaluée dans les cas où le calcul est simple et des équivalents en temps sont calculés. - Comparaison : les deux algorithmes sont uniquement comparés par le calcul de leurs complexités individuellement.
Modèles théoriques	×

Les aspects outils et objets sont présents. Cependant, dans cet extrait, c'est surtout autour de la complexité que l'algorithme existe en tant qu'objet. Tout comme nous l'avions fait remarquer pour l'extrait du document A1, l'aspect d'optimalité de l'algorithme de recherche dichotomique n'est pas du tout évoqué ici alors que les questions se centrent sur la complexité.

Document A8

Ce document se présente comme destiné à des enseignants qui n'ont aucune base algorithmique et s'appuyant sur des exemples tirés du programme de seconde.
Il propose, dans une section intitulée « Un principe fondamental : la dichotomie », une méthode de recherche de zéro d'une fonction.
Le document met en jeu la conception Outil du paradigme AM.

Conception AM-outil dans A8 (extrait dichotomie)

	Conception	Extraits du document (illustration)
P	Instance : Un intervalle $[a, b]$, une fonction f continue sur $[a, b]$ et telle que $f(a)f(b) < 0$ et un réel ϵ Question : Donner une racine approchée de f sur $[a, b]$ à ϵ près. Le problème appartient à $\mathscr{P}_{\text{A}}$	Soit f une fonction continue sur un intervalle $[a, b]$, qui change de signe entre a et b. Le principe de dichotomie permet en divisant à chaque étape l'intervalle de moitié, en considérant le milieu c, de trouver une valeur approchée d'une racine de f à ϵ près.
R	Affectations, instructions conditionnelles, boucles et tests d'un langage proche d'un langage de programmation et opérations mathématiques.	**Algorithme 9** (Recherche d'une racine par dichotomie, $2^{\text{ème}}$ version)**.** Entrées : a réel, b réel, $a < b$, f fonction continue sur $[a, b]$ telle que $f(a) * f(b) \leq 0$, ϵ un nombre réel arbitrairement petit. Sortie : un nombre réel qui approche une racine de f à ϵ près. Invariant : $f(a) * f(b) \leq 0$ **tant que** $b - a > \epsilon$ **faire** $c \leftarrow \frac{a+b}{2}$ **si** $f(a) * f(c) \leq 0$ **alors** $b \leftarrow c$ **sinon** $a \leftarrow c$ **résultat** a
L	Pour R : langage mathématique et pseudo-code très proche d'un langage de programmation Pour Σ : langage mathématique et langue naturelle	
Σ	Description informelle de l'algorithme à mettre en œuvre Discussions sur la condition d'arrêt et sur l'ensemble des cas à envisager Invariant donné dans le préambule de l'algorithme	de f à ϵ près. Pour atteindre la précision ϵ , arbitrairement petite, il suffit d'itérer ce processus n fois, où n est le plus petit entier tel $\frac{\lvert a-b\rvert}{2^n} \leq \epsilon$. On peut aussi dire que n est le plus petit entier supérieur ou égal à $log_2(\frac{\lvert a-b\rvert}{\epsilon})$. —— À chaque étape, il faut veiller à ce que la fonction f ait une racine sur le nouvel intervalle. Cette condition, indispensable pour la validité de l'algorithme n'est pas toujours respectée, et bon nombre de versions que l'on peut rencontrer dans la littérature peuvent dans certains cas donner des résultats erronés. On rencontre aussi, très souvent, une erreur dans la condition d'arrêt : le test n'est pas fait sur la comparaison de la longueur de l'intervalle avec ϵ, mais sur la comparaison $\lvert f(c)\rvert \leq \epsilon$, où c est un point de l'intervalle qui contient une racine exacte. Clairement, $\lvert f(c)\rvert \leq \epsilon$ n'implique pas $\lvert x_0 - c\rvert \leq \epsilon$.

Aspects de l'algorithme dans A8 (extrait dichotomie)

OUTIL	
Effectivité	- programmation : algorithmes exprimés dans le langage de programmation VBA en fin de brochure. - ordinateur/machine : idem - finitude : implicite, à travers la discussion sur le nombre d'étapes nécessaire pour obtenir la précision voulue. - non-ambiguïté : implicite, par le choix d'exprimer les algorithmes dans un langage proche d'un langage de programmation.
Problème	- entrée/sortie : explicitement précisé pour chaque algorithme « Entrées » et « Sortie »

OBJET	
Preuve	- Preuve de correction : Très légèrement présent, implicite à travers la justification mathématique de la méthode. - Invariant : explicité dans le préambule de chaque algorithme mais non questionné.
Complexité	×
Modèles théoriques	×

On trouve aussi dans le document, en annexe, les algorithmes présentés avec Excel et VBA.

La recherche d'une racine par dichotomie est donnée avec l'utilisation d'un tableur (dont il faut faire glisser la formule jusqu'à constater que l'on a atteint la précision voulue) et l'algorithme est aussi proposé en langage VBA. Cela illustre un passage du paradigme AM vers AI mais aucun détail n'est donné concernant ce passage.

Les aspects présents dans l'extrait choisi du document sont résumés dans le second tableau.

L'étude des conceptions et celle des aspects présents montrent toutes les deux l'absence de traitement de l'algorithme comme objet dans cet extrait. Cela peut être lié à l'objectif de la brochure de s'adresser à des enseignants n'ayant aucune connaissance en algorithmique.

Document A9

Ce document se présente comme destiné à des enseignants qui n'ont aucune base algorithmique et ne fait volontairement aucune référence à une pratique en classe de lycée.
Il aborde, dans une section intitulée « Itération non bornée », le principe de dichotomie à travers deux exemples : le jeu du *Juste prix* et la recherche de zéros d'une fonction par dichotomie. Trois problèmes sont en jeu pour cela : la recherche sans information "C'est plus ou c'est moins" du juste prix, la recherche avec cette information et la recherche d'un zéro d'une fonction.
Le document met en jeu la conception Outil du paradigme AI, présentée dans le tableau ci-contre, pour les trois problèmes évoqués plus-haut.

On trouve aussi un autre problème dans le document qui est une variante du problème de recherche de zéros approchés d'une fonction : la question 5 demande d'améliorer l'algorithme afin qu'il prenne aussi en entrée une marge d'approximation. Aucune précision supplémentaire n'est donnée mais on peut imaginer que ce problème sera aussi traité avec la conception AI-outil.

Il faut aussi noter le traitement qui est proposé du jeu du "Juste Prix" pour la version dichotomique. Le document propose ceci :

Nous utiliserons l'algorithme suivant (très nettement hors programme, mais sans doute éclairant pour des prolongements) :

```
   Algorithme : Tirage
   Données : Aucune donnée.
   Résultat : La fonction qui renseigne sur la réussite : p -> 0 si p est le
              nombre tiré, -1 si p est inférieur, 1 si p est supérieur.
   Variables : N entier
1  début
2  |  N ← alea(1001) ;
3  |  renvoyer (p -> si p=N alors 0 sinon si p<N alors -1 sinon 1);
4  fin algorithme
```

Algorithme 29: Tirage d'un nombre qui renvoie la fonction de réussite.

Ce programme tire aléatoirement un nombre entre 1 et 1000 et produit un programme qui répond comme le joueur qui fait deviner ce nombre. On est ici plus dans une problématique de simulation d'un joueur que dans une problématique d'algorithmique.
L'algorithme de recherche dichotomique est proposé sous la forme suivante :

Conception AM-outil dans A8 (extrait dichotomie)

	Conception	Extraits du document (Juste Prix “oui/non”)	Extraits du document (Juste Prix “plus/moins/gagné”)	Extraits du document (Recherche de zéros)
P	Les trois problèmes appartiennent à $\mathscr{P}_{\text{A}}$ Nous ne les détaillons pas ici (Instance, Question) mais cela peut être fait comme dans les tableaux précédents.	Dans notre version algorithmique, le prix à rechercher est un nombre entier compris entre 1 et 1000. [...] Nous vous proposons deux algorithmes solutions différents, dont le résultat est le nombre d'essais avant de trouver le « Juste Nombre ». La première version n'utilise aucun renseignement concernant la position relative du nombre proposé et du « Juste Nombre ».	Une deuxième version utilise un renseignement supplémentaire, fourni dans le jeu réel. Ce renseignement est à 3 valeurs : - « Égal » indiquant que le Nombre proposé est égal au « Juste Nombre » - « Plus petit » indiquant que le Nombre proposé est strictement inférieur au « Juste Nombre » - « Plus grand » dans le dernier cas.	f est une fonction définie sur un intervalle $[a;b]$. Elle a la bonne propriété d'être continue d'une part, et d'avoir ses valeurs de signe contraire aux bornes de l'intervalle ($f(a) * f(b) < 0$). On sait donc qu'elle a un « zéro », c'est à dire une valeur c dans l'intervalle $]a;b[$ telle que $f(c) = 0$. On ne souhaite pas vraiment découvrir la valeur exacte de c, mais seulement une valeur approchée, par exemple à 0.01 près.
R	Affectations, instructions conditionnelles, boucles et tests d'un langage proche d'un langage de programmation et algorithmes dans divers langages	**Algorithme** : JusteNombreV1 **Données** : A le nombre compris entre 0 et 1000 qu'il faut rechercher. **Résultat** : Un entier indiquant le nombre de propositions faites avant de trouver le nombre A. **Variables** : LeNbPropose, NbPropositions entiers 1 **début** 2 LeNbPropose ← ; 3 NbPropositions ← ; 4 **tant que** LeNbPropose **faire** 5 LeNbPropose ← ; 6 **fin tq;** 7 **renvoyer** ; 8 **fin algorithme** **Algorithme 28:** Le Juste Nombre. Une version correcte mais sans doute non optimale.	**Algorithme** : JusteNombreV2 **Données** : Il n'y a pas de donnée à cet algorithme qui suppose juste l'existence de la fonction renseigne : x -> -1, 0 ou 1 suivant la position de x par rapport au nombre recherché **Résultat** : Un entier indiquant le nombre d'essais pour trouver le nombre recherché **Variables** : a,b,m, nbPropositions ← ; 1 **début** **tant tq;** 14 **renvoyer** ; 15 **fin algorithme** **Algorithme 30:** Le Juste Nombre. Une version meilleure ?	``` 1 : Zero () 2 : Func 3 : Local a , b , m 4 : 0 => a 5 : 2 => b 6 : While b−a > 0.01 7 : MoyArith (a , b) => m 8 : If f(a)∗f(m)>0 Then 9 : m => a 10 : Else 11 : m => b 12 : EndIf 13 : EndWhile 14 : Return m 15 : EndFunc ``` Listing 62 – xcas-TI-Zero-simple
L	Pour R : langage mathématique et pseudo-code très proche d'un langage de programmation Pour Σ : langage mathématique et langue naturelle			
Σ	Description informelle de l'algorithme à mettre en œuvre Expérience de la méthode sur quelques exemples Les algorithmes sont toujours des algorithmes à compléter Validation par la programmation	Cet algorithme version simple procède par augmentation successives de 1 de la proposition. Ainsi on est certain de trouver le « Juste Nombre », à condition de commencer avec une proposition convenable. Complétez l'algorithme.	`renseigne` fait partie de l'environnement courant. Dans cette solution on va tenter de se rapprocher de plus en plus du nombre recherché en « encadrant » au mieux ce nombre. Les bornes de cet intervalle encadrant $[a, b]$ sont les entiers a et b initialement 0 et 1000. On choisit de proposer le milieu m de $[a, b]$, et suivant la réponse de recommencer avec un nouvel intervalle encadrant, et ceci jusqu'à trouver. On comptabilise le nombre d'essais. De manière surprenante, ce nombre est au plus ... 11.	2. Faites la trace du calcul itératif proposé dans un tableau, en déterminant l'étape à laquelle on s'arrête de façon à trouver une solution à 0.01 près. étape | a | b | m | $\frac{b-a}{2}$ | $signe(f(a))$ | $signe(f(b))$ | $signe(f(m))$ initialement | 0 | 2 | 1 | 1 | + | - | + 1 2 ....

```
   Algorithme : JusteNombreV2
   Données : Il n'y a pas de donnée à cet algorithme qui suppose juste
             l'existence de la fonction renseigne : x -> -1, 0 ou 1 suivant la
             position de x par rapport au nombre recherché resp. <, = , >
   Résultat : Un entier indiquant le nombre de propositions faites avant de
              trouver le nombre recherché.
   Variables : a,b,m, NbPropositions entiers
1  début
2      a ← 0 ; b ← 1000 ;
3      m ← MoyArit(a,b) ;
4      NbPropositions ← [          ];
5      tant que [ renseigne(m)     ] faire
6          si renseigne(m) = -1 alors
7              [          ]
8          sinon
9              [          ]
10         fin si;
11         m ← MoyArit(a,b) ;
12         NbPropositions ← [          ];
13     fin tq;
14     renvoyer [          ];
15 fin algorithme
```

Algorithme 30: Le Juste Nombre. Une version meilleure ?

Il s'agit donc de faire jouer ce programme contre le programme *renseigne* écrit précédemment, avec la stratégie de dichotomie, et de renvoyer le nombre de coups utilisés. *JusteNombreV2* ne prend donc pas d'entrée mais utilise le programme *renseigne*. Tout cela cache le problème de recherche dichotomique et notamment le type d'instance qu'il permet de traiter : le problème pourrait être formulé ainsi :

> **Instance :** Un choix de nombre entre 1 et 1000.
> **Question :** Quel est le nombre choisi ? Sachant que l'on peut interroger un seul nombre à la fois et qu'interroger un nombre conduit à l'information : le nombre est celui-ci, le nombre cherché est plus petit ou le nombre cherché est plus grand

Dans le document, l'instance (un choix de nombre) est représentée par le programme *renseigne*.
On voit bien ici les difficultés de simulation et d'expression du problème de la recherche dichotomique posées par le problème "Juste Prix" vis-à-vis de la conception AI. Au regard de la conception AI, la recherche dichotomique semble plus adaptée sous la forme de la recherche d'un élément dans une liste triée, comme nous l'avons vue dans les documents précédents.
Notons que le problème du "Juste Prix" ne poserait pas ces difficultés dans la conception PA, en recherchant par exemple une stratégie optimale pour deviner le nombre. En se détachant d'une modélisation informatique du problème, on peut alors se centrer sur la résolution algorithmique du problème (où l'algorithme ici serait une stratégie pour un joueur et où les opérations de base sont données par les règles du jeu).

Concernant la recherche du zéro d'une fonction, on retrouve la même question concernant l'instance du problème :

Algorithme : Zéro
Données : Aucune donnée. La fonction f est connue, globale donc. Les bornes initiales sont $a = 0$, $b = 2$. La marge d'approximation est fixée 0.01
Résultat : Un zéro de f sur $]a; b[$ à 0.01 près.
Variables : a,b et m des nombres flottants.

```
1  début
2  |  a ← 0 ; b ← 2 ;
3  |  tant que (b - a ) [          ] faire
4  |  |  m := MoyArit(a,b);
5  |  |  si ( f(a) * f(m) [          ] ) alors
6  |  |  |  [          ]
7  |  |  sinon
8  |  |  |  [          ]
9  |  |  fin si;
10 |  fin tq;
11 |  renvoyer [          ];
12 fin algorithme
```

Algorithme 31: Détermination d'un zéro de f à 0.01 près par dichotomie

Si l'on s'intéresse au problème posé et à l'algorithme décrit, la fonction f fait partie de l'instance du problème et donc de l'entrée de l'algorithme. Il y a un amalgame entre le programme qui n'a pas d'entrée et l'algorithme sous-jacent qui s'applique bien à une fonction. Un programme implémentant cet algorithme peut ne pas prendre de données en entrée si ses entrées font partie de l'environnement.

On constate donc ici que le paradigme AI est très fortement marqué, au point que certains problèmes qui seraient plus clairement ancrés dans un autre paradigme sont tout de même traités dans AI, soulevant des difficultés supplémentaires de simulation et de modélisation qui peuvent être un obstacle à l'étude des algorithmes en jeu. L'amalgame algorithme-programme est aussi caractéristique d'un fort ancrage dans AI. Enfin, la structure du document qui répartit et hiérarchise les problèmes selon le type de structure de programmation (instructions conditionnelles, puis itérations bornées et finalement itérations non-bornées) va aussi dans ce sens : l'objectif de l'algorithmique est ramené à la maîtrise de ces structures, fortement liées à la programmation.

Les aspects présents dans l'extrait choisi du document sont résumés dans le tableau suivant :

Aspects de l'algorithme dans A9 (extrait dichotomie)

OUTIL	
Effectivité	- programmation : présence des algorithmes sous forme de programmes et l'amalgame algorithme-programme - ordinateur/machine : idem
Problème	- entrée/sortie : explicitement précisées pour chaque algorithme « Données » et « Résultat » mais un amalgame est fait entre saisie dans un programme et entrée d'un algorithme
OBJET	
Preuve	×
Complexité	×
Modèles théoriques	×

L'étude des conceptions et celle des aspects présents montrent toutes les deux l'absence de traitement de l'algorithme comme objet dans cet extrait. On note aussi très nettement un placement dans le paradigme AI avec une place très marquée de la programmation.

Document A13

Le document propose une activité pour la classe « ludique et progressive », autour du jeu de recherche d'un nombre "C'est plus, c'est moins". Il se présente sous la forme d'une fiche d'activité à distribuer aux élèves. Des codes de programmes sont donnés pour les enseignants.
Dans la première partie, l'activité consiste à découvrir la fonction de génération d'un nombre aléatoire de la calculatrice et à jouer à deux au jeu "c'est plus, c'est moins" avec un nombre choisi aléatoirement entre 0 et 100.
Ensuite, l'activité propose de d'écrire un « algorithme pour jouer seul ». Il s'agit de simuler le joueur qui fait deviner avec sa calculatrice. Le document parle d'algorithme en langage naturel puis en langage TI. Il s'agit plutôt d'un programme. On demande ensuite de lui ajouter un compteur pour donner le nombre de coups joués.
La troisième partie consiste à faire deviner le nombre par la calculatrice. C'est ici qu'intervient la dichotomie.
Dans cette dernière partie, le document met en jeu la conception Outil du paradigme AI :

Conception AI-outil dans A13

	Conception	Extraits du document (illustration)
P	Instance : le choix d'un nombre entre 1 et 100 Question : quel est le nombre ? Le problème appartient à $\mathscr{P}_{\text{A}}$	L'objectif de cette activité est de penser à un entier de 0 à 100 et de le faire deviner par la calculatrice. Pour cela il faut mettre en place une stratégie rigoureuse qui permet à tous les coups d'aboutir.
R	Affectations, instructions conditionnelles, boucles et tests d'un langage proche d'un langage de programmation et opérations mathématiques.	Langage naturel / Traduction en langage TI **Entrée** : ..... **Traitement** : tant que ..... **Sortie** : EUREKA Afficher le nombre auquel vous pensiez
L	Langage très proche d'un langage de programmation, langage de programmation, langue naturelle.	
Σ	Essai de méthodes sur des instances particulières pour deviner l'algorithme et sa condition d'arrêt.	1°) Prenons un premier exemple où l'entier à deviner est **20**. Sachant que c'est un entier entre 0 et 100, compléter les différentes étapes suivantes pour réduire au fur et à mesure l'intervalle de recherche $[A; B]$ [...] 2°) Prenons un deuxième exemple où l'entier à deviner est **43**. [...] 3°) En observant les deux exemples précédents, pouvez-vous donner le test qui permet d'arrêter le processus ? 4°) En utilisant les différentes étapes précédentes, écrire en langage naturel un algorithme correspondant.

On peut noter le signe d'un amalgame entre l'affichage à l'écran que peut faire un programme et la sortie d'un algorithme, à savoir la réponse à la question du problème auquel répond l'algorithme. Par exemple, la sortie de l'algorithme demandé est :

Sortie : EUREKA
Afficher le nombre auquel vous pensiez

Les aspects présents dans l'extrait choisi du document sont résumés dans le tableau suivant :

Aspects de l'algorithme dans A8 (extrait dichotomie)

OUTIL	
Effectivité	- programmation : algorithmes demandés dans un langage proche d'un langage de programmation puis dans un langage de programmation. - ordinateur/machine : idem - non-ambiguïté : implicite, par le choix d'exprimer les algorithmes dans un langage proche d'un langage de programmation.
Problème	- entrée/sortie : explicitement précisé pour chaque algorithme « Entrées » et « Sortie »

OBJET	
Preuve	×
Complexité	×
Modèles théoriques	×

L'étude des conceptions et celle des aspects présents montrent toutes les deux l'absence de traitement de l'algorithme comme objet dans ce document. La validité de l'algorithme n'est pas du tout questionnée, ni la complexité. C'est pourtant la question d'optimiser le nombre d'étapes qui justifie la dichotomie.
On peut noter aussi que l'amalgame entre programme et algorithme est présent lors de la simulation du joueur qui fait deviner ou encore avec l'amalgame entre sortie et affichage à l'écran. Cela peut être lié à une influence du paradigme AI.

Document A21

C'est un document d'accompagnement de stage d'initiation à l'algorithmique pour les enseignants du secondaire. Il est fait dans ce document de nombreuses références à une pratique en classe de lycée.
Il aborde, dans la section intitulée « Recherche dichotomique », le principe de dichotomie à travers deux exemples : le jeu *Je cherche un nombre entre 1 et 1000* et la recherche de solutions de $f(x) = 0$ par dichotomie.
Le document met en jeu les conceptions Outil et Objet des paradigmes AM et AI dans les deux problèmes. Nous résumons dans les tableaux de la page suivante toutes les conceptions présentes dans l'extrait.

Les aspects présents dans l'extrait du document sont résumés dans le troisième tableau.
On trouve dans cet extrait l'algorithme à la fois comme outil et comme objet. Cependant, le document propose que les questions concernant l'objet ne soient pas conservées pour une proposition de l'activité en classe :

> **Question 6** – Quels sont les nombres à choisir pour que l'ordinateur trouve la solution en au moins 9 coups ?
> **Question 7** – Montrer que l'algorithme « optimum » permet de toujours trouver l'entier en au plus 10 essais. [...]
> *Remarque* – Dans une situation pour la classe, on utilisera les mêmes exercices sans les questions 6 et 7. On essayera de faire émerger chez les élèves la méthode de recherche par dichotomie.

Le document ne montre pas la différence entre entrée-sortie et saisie-affichage [1]. De même, la simulation du joueur qui fait deviner semble avoir le même statut que l'algorithme de

1. Telle que nous l'avons définie p. 128.

Conceptions dans A21 (Je cherche un nombre entre 1 et 1000)

	Conception outil	Conception objet	
P	Instance : le choix d'un nombre entre 1 et 1000 Question : quel est le nombre ? Le problème appartient à $\mathscr{P}_{\text{A}}$	Instance : un algorithme A et un entier n Question : pour quelles instances l'algorithme A nécessite-t-il plus de n étapes ? Le problème appartient à $\mathbb{P}$, il est instancié pour l'algorithme donné et $n = 9$	Instance : un algorithme A et un entier n Question : L'algorithme A a-t-il une complexité au pire en n étapes ? Le problème appartient à $\mathbb{P}$, il est instancié pour l'algorithme donné et $n = 10$
R	Opérations autorisées pour le joueur Affectations, instructions conditionnelles, boucles et tests d'un langage proche d'un langage de programmation instructions et opérateurs du langage Python.	programmation : calcul du nombre de coups nécessaires pour chaque valeur du nombre à deviner et listage de ceux demandant au moins 9 essais.	Propriété mathématique : « si $B - A \leq 2^K$, alors on peut trouver le nombre compris strictement entre A et B en au plus K essais.
L	Langage pseudo-code, langage de programmation, langue naturelle.	langage de programmation	langage mathématique
Σ	*non explicités* programmation et test du programme	Exécution ?	Logique et raisonnement mathématiques (preuve par récurrence)

Conceptions dans A21 (Solution de $f(x) = 0$)

	Conception outil	Conception objet	Conception objet
P	Instance : deux réels a et b, une fonction f croissante et s'annulant sur $[a, b]$, une précision p Question : Quelle est, avec une précision p, une solution à $f(x) = 0$? Le problème appartient à $\mathscr{P}_{\text{A}}$	Instance : un algorithme A et un problème P de $\mathscr{P}_{\text{A}}$ (instancié pour le problème "$f(x) = 0$" et l'algorithme de dichotomie) Question : A résout-il P (après un nombre fini d'étapes) ? Le problème appartient à $\mathbb{P}$	Instance : un algorithme A qui résout un problème P en calcul exact sur les réels (instancié pour la recherche de zéros d'une fonction par dichotomie) Question : A résout-il P en calcul approché (arithmétique flottante) Le problème appartient à $\mathbb{P}$
R	Instructions et opérateurs d'un pseudo-code	Propriétés des objets mathématiques en jeu Étude des différents cas possibles Invariant	Étude des diverses formes d'erreurs d'approximation possibles étude du comportement de l'algorithme pour ces erreurs
L	Langage pseudo-code, langage mathématique, langue naturelle.	Langage mathématique	Langage mathématique
Σ	Propriétés de la fonction étude des différents cas pouvant se présenter invariant de boucle	Logique mathématique et raisonnement	Hypothèses sur les erreurs possibles Logique et raisonnement mathématique

Aspects de l'algorithme dans A21 (extrait dichotomie)

OUTIL	
Effectivité	- programmation : présence des algorithmes sous forme très proche d'un langage de programmation et de programmes Python - ordinateur/machine : idem, questions d'erreurs d'arrondis - opérateur : recherche de la stratégie « optimum » pour la recherche du nombre - finitude : Justification de la terminaison.
Problème	- entrée/sortie : explicitement précisées pour chaque algorithme « Données » et « Résultat » mais un amalgame est fait entre données d'un programme et entrées d'un algorithme - instance : recherches des instances les plus "coûteuses" pour la recherche dichotomique - Résout un problème pour toute instance : présence de la preuve de correction de l'algorithme

OBJET	
Preuve	- Preuve de terminaison : proposée pour la recherche de zéro d'une fonction - Preuve de correction : proposée pour la recherche de zéro d'une fonction - Invariant : invariant de la recherche de zéro d'une fonction
Complexité	- Au pire : étude de la complexité de la recherche dichotomique
Modèles théoriques	×

dichotomie. Ce sont des signes d'un amalgame entre programmation et algorithmique.
Comme dans les autres documents, l'optimalité de la recherche dichotomique n'est pas justifiée, la stratégie est simplement qualifiée d'*optimum*.

Bilan

On peut constater, dans ces six exemples, une forte présence de l'aspect outil. Trois documents mettent en jeu l'aspect objet à travers la preuve de l'algorithme.
Les conceptions se placent dans AM et AI, avec une forte présence de la programmation (qui occulte parfois l'algorithme).
Ces constats sont assez caractéristiques de l'ensemble du corpus, si l'on se concentre sur les problèmes issus de $\mathscr{P}_{\text{A}}$ ayant un potentiel pour traiter l'algorithme comme objet. Nous verrons plus bas que de tels problèmes ne sont pas si courants dans les ressources.

L'absence de l'étude de l'optimalité de la dichotomie soulève une question. Quel est l'objectif des activités étudiées ? Il semble que ce soit plus de présenter et de faire implémenter une méthode "classique" que de trouver la meilleure méthode pour résoudre le problème posé. On le voit bien dans l'activité 3 de la ressource A13 (extrait page suivante).

L'objectif n'est pas de trouver la stratégie optimale mais simplement une *stratégie rigoureuse*. Après deux exemples, le document présente la dichotomie. A priori, toute recherche tenant compte de la réponse "plus" ou "moins" est rigoureuse et permet de trouver l'entier caché. De plus on voit bien que la stratégie doit être indiquée par l'enseignant avant de faire les exemples et est induite par le tableau (colonne $B - A$). Sinon, pourquoi ne pas tester le nombre 20 dès la première question (ou 43 dans le deuxième exemple) ?
La définition proposée de la dichotomie ne souligne pas le fait que l'on découpe en deux l'intervalle de manière équilibrée, alors que tout test d'un nombre conduit bien à découper l'intervalle. On peut donc penser que dans cette activité la stratégie est donnée ou suggérée par l'enseignant et que l'objectif est l'implémentation (les questions suivantes sont d'ailleurs de donner le test d'arrêt, d'écrire la stratégie en langage naturel puis en langage calculatrice).

Activité 3 : *Le Jeu dans l'autre sens*

L'objectif de cette activité est de penser à un entier de 0 à 100 et de le faire deviner par la calculatrice.
Pour cela il faut mettre en place une stratégie rigoureuse qui permet à tout les coups d'aboutir.

1°) Prenons un premier exemple où l'entier à deviner est **20.**
Sachant que c'est un entier entre 0 et 100, compléter les différentes étapes suivantes pour réduire à fur et à mesure l'intervalle de recherche [A. ;B]

	A	B	B – A	proposition
1ère étape	0	100		
2ème étape	0			
3ème étape	0			
4ème étape				
5ème étape				
6ème étape				
7ème étape				

2°) Prenons un deuxième exemple où l'entier à deviner est **43**
Compléter le tableau suivant :

	A	B	B – A	proposition
1ère étape	0	100		
2ème étape	0			
3ème étape	0			
4ème étape				
5ème étape				

Retenir :
Cette méthode itérative, qui consiste à diviser en deux l'intervalle et à déterminer l'une des deux parties dans laquelle se trouve le nombre recherché est appelée **dichotomie**
La **dichotomie** (« couper en deux » en grec)

3°) En observant les deux exemples précédents, pouvez-vous donner le test qui permet d'arrêter le processus

.....

Ressource A13 - activité 3

Il faut aussi noter que la dichotomie est présente dans plusieurs problèmes, issus de différents domaines (recherche dans une liste, jeu "c'est plus, c'est moins", recherche d'un zéro de fonction). Selon ces domaines, le paradigme adopté n'est pas nécessairement le même. Ainsi la recherche dans une liste et la recherche d'un zéro sont des problèmes directement posés dans AI. Par contre, le jeu "c'est plus, c'est moins" se pose hors d'un contexte informatique. Son traitement dans AI demande une simulation des joueurs et de leur stratégie qui peut créer un flou autour de la notion d'algorithme et un amalgame entre la programmation de ces simulations et les algorithmes ou stratégies en jeu.

Concernant nos outils d'analyse, on peut voir ici leur potentiel pour étudier les conceptions-objet (ce que l'on n'avait pas eu l'occasion d'étudier jusqu'alors). On peut aussi noter que l'on a pu repérer des nuances entre les différents traitements d'un même algorithme grâce à la notion de conception.

2.2 Quels Aspects dans les ressources ?

Dans cette section, l'italique dans le texte sera réservé aux citations. Lorsque le contexte ne permet pas de savoir de quel document l'on parle, nous ferons référence aux documents sous la forme (Ai) ou (A$i.j$), où i est le numéro du document et j la page (si nécessaire).

Effectivité

L'aspect EFFECTIVITÉ est omniprésent dans les ressources. On le retrouve dans tous les documents. Sa présence peut être implicite à travers, notamment, les activités de programmation et d'exécution, la description des algorithmes en instructions, etc. Souvent, elle est présente explicitement dans les discours sur l'algorithmique, les définitions d'algorithme ou encore dans l'étude de la terminaison. Par exemple :

> Algorithme : Un algorithme est une suite d'instructions qui, une fois exécutée correctement, conduit à un résultat donné. Un algorithme doit contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter. (A6.1)

> C'est un concept pratique, qui traduit la notion intuitive de procédé systématique, applicable mécaniquement, sans réfléchir, en suivant simplement un mode d'emploi précis. (A8.9)

> Examinons plus particulièrement des algorithmes plus anciens : les recettes de cuisine. Une recette de cuisine comporte 3 parties :
> 1. Réunir les ingrédients 2. Préparer 3. Déguster
> La préparation consiste à exécuter une suite d'instructions : par exemple, plonger les tomates dans une casserole d'eau bouillante pendant quelques instants avant de les peler. On ne sait pas pourquoi il faut procéder de la sorte et d'ailleurs, ça n'a aucune importance. La recette a été écrite par quelqu'un qui sait. Elle marche. (A20.1)

Problème

L'aspect PROBLÈME se retrouve très souvent dans les ressources. Cependant, il faut différencier l'utilisation d'algorithmes dans une démarche de résolution de problème et le fait qu'un algorithme soit la résolution d'un problème. C'est, bien entendu ce deuxième cas qui nous intéresse et que nous cherchons à repérer.
En réalité, une certaine quantité de ressources ne met pas en jeu cette deuxième relation aux problèmes. C'est en particulier le cas de documents qui mettent en jeu des PROGRAMMES DE MODÉLISATION-SIMULATION, des ALGORITHMES-INSTANCIÉS ou qui visent à l'apprentissage d'un langage. Seule une quinzaine de documents [2] met en jeu l'aspect PROBLÈME. Cela peut être exprimé par la présence claire d'une entrée et d'une sortie à l'algorithme ou la vision des algorithmes comme des fonctions (qui associent à chaque entrée une sortie), la référence à la (bonne) notion de problème ou à celle d'instances. Par exemple :

> **Tentative de définition :** Suite d'ordres précis, compréhensibles et exécutables de manière non ambigüe par un automate, devant être exécutés suivant un ordre parfaitement déterminé en vue de résoudre une classe de problèmes. (A8.9)

> Nous privilégions une approche fonctionnelle, où un algorithme a une spécification : ses données (les valeurs qu'il reçoit en arguments, les variables associées à ces valeurs) et ce qu'il calcule ou renvoie, et un corps qui décrit précisément le calcul réalisé. (A9.1)

Enfin, notons que cette notion est parfois indispensable à la description récursive d'algorithme, qui demande que toute instance du problème puisse être découpée en des instances plus petites du même problème. C'est le cas pour les algorithmes "diviser pour régner" :

2. Dont quasiment tous ceux à destination des enseignants.

Les trois phases du paradigme[3] "diviser pour régner" :

- Diviser : division du problème en un certain nombre de sous-problèmes (semblables au problème initial mais de taille moindre, par exemple de taille moitié).
- Conquérir et Régner : résolution des sous-problèmes
  - par des appels récursifs,
  - directement lorsque le sous-problème est de taille "petite".
- Combiner : combinaison des solutions des sous-problèmes pour constituer la solution globale. (A3.1)

Les activités choisies jouent un rôle important dans la présence ou non de l'aspect PROBLÈME. L'absence de problème de $\mathscr{P}_{\text{A}}$ implique l'impossibilité d'étudier des problèmes de $\mathbb{P}$, et donc que l'algorithme soit objet.

Preuve

L'aspect PREUVE est rare dans les documents. On le retrouve dans 7 ressources : A1, A2, A3, A4, A8, A20 et A21. L'engagement dans cet aspect peut être plus ou moins fort, de la simple évocation d'un lien (A20) jusqu'à en faire une question centrale du document (A1, A8) :

> La démarche présentée tente, sans introduire un formalisme excessif, d'apporter des éléments de réponse aux questions fondamentales qui se posent en algorithmique :
> - Comment élaborer un algorithme ?
> - L'algorithme s'arrête-t-il toujours ?
> - Comment prouver qu'un algorithme résout effectivement le problème posé ? (A1.1)

Dans les autres ressources (A2, A3, A4, A8, A21) la question de valider par la preuve (correction et terminaison) un algorithme revient très souvent :

> Justifier que l'algorithme [pour les fractions égyptiennes] se termine et qu'il donne une suite finie $(n_1, ..., n_r)$ satisfaisant aux contraintes imposées. (A4.2)

Ou encore concernant l'optimalité de la solution produite par un algorithme :

> Glouton 1 – On trie les épreuves par durée croissante, on choisit la plus courte, puis la plus courte parmi celles qui lui sont compatibles, puis... Ce choix mène-t-il au déroulement d'un nombre d'épreuves maximal ? (A2.3)

L'intégralité des ressources mettant en jeu l'aspect PREUVE est à destination des enseignants. L'exemple de la dichotomie dans A1 montre bien que les questions de preuve, même reconnues comme importantes, ne se retrouvent pas du côté des ressources pour la classe.

Complexité

L'aspect COMPLEXITÉ n'apparaît que dans 5 documents (A1, A2, A3, A19, A21). Quatre d'entre eux sont destinés aux enseignants. Dans certains, la complexité est évoquée concernant certains algorithmes. Dans d'autres, la complexité est abordée par l'évaluation directe de la complexité (comme pour la dichotomie dans A21) ou bien par la comparaison d'algorithmes :

3. Le document A3 utilise le terme de *paradigme* "diviser pour régner". Ce terme n'a aucun lien avec notre notion de paradigme. Il est courant en informatique mais nous avons choisi de lui substituer d'autres termes dans nos propos pour éviter toute confusion.

1. Combien, au plus, d'itérations de la boucle auront lieu lors d'une recherche pour chacun des deux programmes [recherche par balayage et par dichotomie] ? Pour simplifier on se placera dans le cas où le nombre d'éléments dans la liste est de la forme $n = 2p$.
2. On suppose que $n = 2100$. Admettons que la machine fasse un million de boucles en 1 seconde, quel temps (dans le pire des cas) sera utilisé pour chacun des deux programmes lors d'une recherche ? (A3.2)

Dans le cas de questions d'évaluation de complexité, on peut penser qu'il s'agit vraiment de traiter un problème de $\mathbb{P}$, on trouve des opérateurs (et des structures de contrôle) pour étudier cette complexité.
Attardons-nous sur l'unique ressource pour la classe mettant en jeu l'aspect COMPLEXITÉ : A19. Il s'agit ici de discuter de la complexité du tri à bulle :

3 - Qu'est-ce que le « tri à bulles » ?

Supposons que l'on ait à ordonner dans l'ordre croissant la suite

$$A = 5, 1, 4, 8, 2.$$

La *manipulation de base* du « tri à bulles » consiste à examiner *successivement* les couples formés par les nombres de rang 1 et 2, puis par les nombres de rang 2 et 3 et ainsi de suite jusqu'à la fin et de leur appliquer le traitement suivant :

On laisse inchangé tout couple qui est déjà rangé dans l'ordre croissant ; dans le cas contraire, on échange ses 2 termes.

Voici les transformations successives subies par A au cours de ce processus :

$$A = \underline{5, 1}, 4, 8, 2 \ \rightarrow\ 1, \underline{5, 4}, 8, 2 \ \rightarrow\ 1, 4, \underline{5, 8}, 2 \ \rightarrow\ 1, 4, 5, \underline{8, 2} \ \rightarrow\ B = 1, 4, 5, 2, 8$$

La suite B n'est pas ordonnée dans l'ordre croissant mais *nous avons fait des progrès* dans le classement car le maximum de $5, 1, 4, 8, 2$, qui est 8, se trouve maintenant à la fin de la suite, qui est sa place définitive.

Cela conduit naturellement à appliquer à B la *manipulation de base*. Voici ce que cela donne

$$B = \underline{1, 4}, 5, 2, 8 \ \rightarrow\ 1, \underline{4, 5}, 2, 8 \ \rightarrow\ 1, 4, \underline{5, 2}, 8 \ \rightarrow\ 1, 4, 2, \underline{5, 8} \ \rightarrow\ C = 1, 4, 2, 5, 8$$

C'est mieux car les 2 derniers termes sont maintenant à leurs places définitives. Recommençons en appliquant la manipulation de base à C :

$$C = \underline{1, 4}, 2, 5, 8 \ \rightarrow\ 1, \underline{4, 2}, 5, 8 \ \rightarrow\ 1, 2, \underline{4, 5}, 8 \ \rightarrow\ 1, 2, 4, \underline{5, 8} \ \rightarrow\ 1, 2, 4, 5, 8$$

On voit que le rangement de A dans l'ordre croissant est terminé. Pour cela, on a dû appliquer 3 fois la manipulation de base.

Si on imagine maintenant que A est une suite quelconque de $n \geq 2$ nombres quelconques, il est clair qu'après une première application de la *manipulation de base*, le dernier terme du nouvel état de A sera à sa place définitive ; après la seconde application de la *manipulation de base*, les 2 derniers termes seront à leurs places définitives, et ainsi de suite. Par conséquent, après la $(n-1)^{\text{ème}}$ application de la *manipulation de base*, les $(n-1)$ derniers termes - et par conséquent le premier - seront à leurs places définitives, ce qui veut dire que le rangement sera terminé. L'exemple $A = 5, 1, 4, 8, 2$ montre que, quelquefois, il n'y a pas besoin d'effectuer les $(n-1)$ applications de la *manipulation de base*.

Question - Combien faut-il de *manipulations de base* pour ranger $5, 4, 3, 2, 1$ dans l'ordre croissant ?

Nous retiendrons que nous sommes sûrs que A est rangé dans l'ordre croissant au bout de $(n-1)$ *manipulations de base*.

On trouve donc la preuve de la complexité de l'algorithme, avec opérateurs de preuve et structures de contrôle de la logique et du raisonnement. On se situe bien dans $\mathbb{P}$. Cependant la tâche demandée à l'élève est bien moindre, il s'agit de compter le nombre d'étapes nécessaires sur un exemple, afin de justifier que la complexité au pire calculée est atteinte. Malgré cela, cette question se place dans une véritable problématique de complexité (plus loin on trouve d'ailleurs une expérience de comparaison avec le tri rapide).

Modèles Théoriques

L'aspect MODÈLES THÉORIQUES n'est présent dans aucun des documents.

Outil-Objet

Les aspects nous permettent déjà de repérer 3 types de documents :

– des documents où l'algorithme est uniquement outil, une majorité,

– des documents où l'algorithme est *outil* et *objet* et qui, à l'exception d'un seul, sont destinés aux enseignants,

– des documents où l'algorithme (à notre sens) n'est pas vraiment en jeu. Dans ces derniers, l'algorithmique est assimilée à la programmation ou à l'écriture de programmes-papiers.

Cela nous donne déjà une idée des conceptions que l'on va retrouver dans les ressources. Nous allons maintenant nous intéresser à ces conceptions et aux paradigmes dans lesquels se placent les documents.

2.3 Quelles conceptions dans les ressources

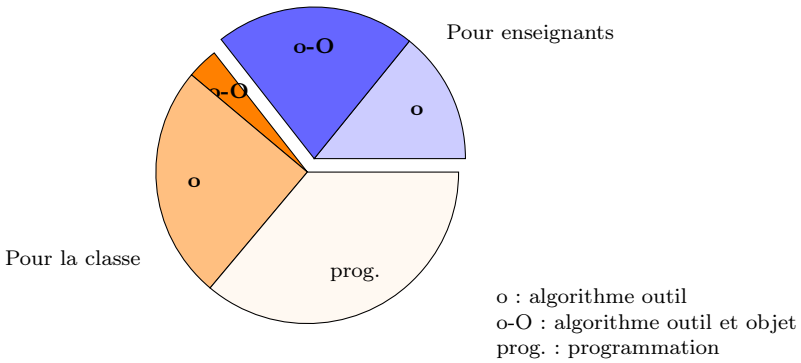
En s'intéressant aux conceptions présentes, en particulier aux quadruplets (P,R,L,Σ) et en s'appuyant sur le modèle des μ-conceptions, on peut préciser nos analyses.

Problèmes, $\mathscr{P}_{\text{A}}$, $\mathbb{P}$

Le résultat du paragraphe outil-objet précédent peut être reformulé et raffiné en terme de problèmes en jeu.

Les documents où l'algorithme est outil correspondent à la présence de problèmes de $\mathscr{P}_{\text{A}}$. Rappelons que nous considérons que l'algorithme est présent si les problèmes de $\mathscr{P}_{\text{A}}$ sont non-instanciés. Si les problèmes de $\mathscr{P}_{\text{A}}$ sont instanciés, on considère que la notion d'algorithme n'est plus totalement en jeu. Enfin, si les problèmes ne sont pas dans $\mathscr{P}_{\text{A}}$ (en particulier si l'on ne reconnaît pas un problème sous la forme instances-question) alors on considère que l'algorithme n'est pas en jeu.

Lorsque les ressources mettent aussi en jeu l'objet, on peut reconnaître des problèmes de $\mathbb{P}$. Ces problèmes seront souvent instanciés, ce qui n'empêche pas d'être du côté objet d'une conception. En général, lorsque l'on veut traiter de tels problèmes de manière non-instanciée, il est nécessaire de s'appuyer sur des modèles théoriques (qui sont absents de toutes les ressources ici).
On peut donc résumer la place de l'algorithme en tant qu'outil et objet comme suit [4] :



4. étant donné le nombre de documents étudiés, un pourcentage exact ne serait pas significatif. Ce diagramme se veut seulement le reflet d'une tendance.

La Preuve Algorithmique PA absente

La présence très restreinte de l'aspect PREUVE ne laisse pas beaucoup de chance de rencontrer PA. En fait, cette conception ne se retrouve dans aucune ressource.

Une ressource propose une conception de l'algorithme approchant de PA : A1. Dans ce document, la méthode proposée pour construire des algorithmes itératifs tout en les validant, propose de chercher d'abord un invariant dont l'algorithme découle ensuite. On reconnaît certains aspects de PA, en particulier le fait que l'algorithme découle de la preuve (ici via l'invariant). Cependant, il ne s'agit pas tout à fait de PA car la preuve dont l'algorithme découle n'est pas tout à fait celle de la solution du problème :

> On sait que ce qui caractérise une itération est son invariant. Trouver l'invariant d'une boucle n'est pas toujours chose aisée. L'idée maîtresse de la méthode est de construire cet invariant parallèlement à l'élaboration de l'algorithme et non pas de concevoir l'algorithme itératif pour ensuite en rechercher l'invariant.[...] Cette façon de procéder montre comment on prouve la validité de l'algorithme au fur et à mesure de son élaboration. En effet la situation générale choisie en [1] est en fait l'invariant qui caractérise la boucle *tantque*. Cette situation est satisfaite au départ à cause de l'étape [4], elle reste vraie à chaque itération (étape [3]). Ainsi lorsque la condition d'arrêt est atteinte cette situation nous permet d'affirmer que le problème est résolu. C'est également en analysant l'étape [3] qu'on peut prouver la terminaison de l'algorithme. (A1.2)

Algorithme Mathématique AM

On rencontre ce paradigme de manière franche dans 7 documents (A1, A2, A4, A8, A9, A19, A20) avec divers degrés dans l'expression des algorithmes. Cela peut aller d'un langage avec très peu de contraintes (A2, A19) jusqu'à des langages très contraints, décrits par un pseudo-code (A1, A4, A8, A9). Par exemple (voir aussi l'extrait de A19, plus haut) :

> 1. Un voleur dévalisant un magasin trouve n objets. L'objet numéro i vaut v_i euros et pèse w_i kg. Le voleur veut que son butin ait la plus grande valeur (en euros) possible mais ne peut pas emporter plus de W kg dans son sac à dos.
>
> **Glouton 1** – Le résultat est-il optimal en choisissant l'objet le plus cher parmi ceux qui peuvent tenir dans le sac, puis le plus cher parmi ceux qui peuvent encore tenir...
>
> **Glouton 2** – Le résultat est-il optimal en choisissant d'abord les objets de plus grand prix au kg ? (A2.11)

$x \leftarrow a$
$y \leftarrow b$
$z \leftarrow 0$
tant que $y \neq 0$ **faire**
  $z \leftarrow z + x \times y$ **mod** 10
  $x \leftarrow 10 \times x$
  $y \leftarrow y$ **div** 10
résultat z

(A8.17, un algorithme de multiplication)

Ou encore cet exemple du document A20 :

> Le crible d'Ératosthène fournit la liste des nombres premiers de 1 à n, pour tout entier $n \geqslant 3$, en exécutant le programme suivant :

1. Lire n.
2. Écrire la liste des nombres entiers de 1 à n.
3. Encercler 1 et 2, supprimer les autres multiples de 2 figurant dans la liste.
4. Entourer d'un cercle le plus petit des nombres non encerclés restants, s'il y en a, et supprimer les autres multiples de ce nombre.
5. Répéter l'étape 4 jusqu'à ce qu'il n'y ait plus dans la liste restante de nombres non encerclés.
6. Afficher la liste restante. (A20.3)

Les ressources en question sont toutes, à l'exception de A19, à destination des enseignants. Il semble que ce soit le paradigme AI que l'on va retrouver majoritairement dans les ressources pour la classe.

À chaque fois, ce paradigme cohabite avec AI. Rappelons cependant que la présence d'éléments relatifs à l'ordinateur ou aux spécificités d'un langage de programmation précis, même dans un pseudo-code, relève uniquement de AI. Nous verrons des exemples dans le paragraphe suivant.

Algorithme Informatique AI et programmation

Les langages de programmation sont présents dans quasiment tous les documents. On peut même se demander si, comme dans les instructions officielles, l'algorithmique n'est pas parfois identifiée à la programmation. On trouve dans les documents proposés une très grande diversité d'outils et de langages de programmation (Scratch, Algobox, VBA, TI, Casio, Xcas,...) ainsi que plusieurs expressions du type PROGRAMME-PAPIER. Tout cela relève du système de représentation de AI.

Cependant, on n'est en présence de conception de AI que si l'algorithme est en jeu, c'est-à-dire que des problèmes de $\mathscr{P}_{\text{A}}$ ou $\mathbb{P}$ sont étudiés. Ainsi, il faut mettre de côté tous les documents qui ne relèvent pas de l'algorithmique directement (voir paragraphes PROBLÈME et problèmes, $\mathscr{P}_{\text{A}}$, $\mathbb{P}$, plus haut).
Au total, nous relevons que moins de la moitié des documents mettent en jeu des conceptions de AI. Par exemple :

Algorithme 1 : Calcul du pgcd de deux entiers a et b

```
Entrées :
a : entier naturel
b : entier naturel
Sorties :
pgcd de a et b
début
    r ← reste de la division euclidienne de a par b
    tant que r ≠ 0 faire
        a ← b
        b ← r
        r ← reste de la division euclidienne de a par b
    fin
    Afficher : "le pgcd de a et b vaut b"
fin
```

(A15.1)

Xcas

```
sequentiel(liste,valeur):={
local rang;
rang:=0;
tantque liste[rang]<>valeur  faire
rang:=rang+1;
ftantque;
retourne rang;
};;
```

(A3.2, recherche séquentielle)

Programme en VBA

```
Public Function MultSimple(a As Integer, b As Integer) As Integer
Dim r As Integer
r = 0
Do While b <> 0
 r = r + a
 b = b - 1
Loop
MultSimple = r
End Function
```

(A8.38, algorithme de multiplication)

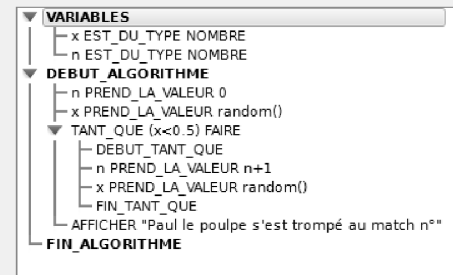
Dans pratiquement tous les cas, on retrouve beaucoup de tâches de rédaction d'algorithmes, mais surtout d'implémentation et d'exécution. Que le concept en jeu soit le programme ou l'algorithme, c'est avant tout en tant qu'outil qu'on le rencontre.

Les documents qui mettent en jeu les langages de programmation (ou des PROGRAMMES-PAPIER) sans mettre en jeu l'algorithme se répartissent en trois catégories :

- Les documents pour l'apprentissage d'un langage ou d'un logiciel, par exemple A5, A6 ou A7, qui proposent d'écrire des programmes pour apprendre les commandes du langage.
- Les documents qui proposent des activités qui permettent d'introduire une nouvelle instruction (instruction conditionnelle, boucles, etc.) voire qui proposent une progression dans l'introduction des différentes instructions (comme A29).
- Les documents qui utilisent la programmation comme un outil pour résoudre des problèmes mathématiques, souvent en lien avec une démarche expérimentale. C'est souvent le cas avec les probabilités et statistiques comme dans les documents A23 ou A24. On le retrouve aussi concernant l'évaluation de termes de suites pour conjecturer leur comportement (stationnaire par exemple) comme dans A10 ou A12.

Dans tous ces cas les programmes en jeu sont rarement des algorithmes, on retrouve des ALGORITHMES-INSTANCIÉS et des PROGRAMMES DE MODÉLISATION-SIMULATION ou des traductions de processus systématiques :

> Activité : créer, avec Algobox, un programme nommé IMCFH qui demande le sexe de la personne et qui, suivant la réponse, donnera ce que serait le poids idéal. (A6.5, une traduction d'une fonction avec conditionnelle)
>
> Points alignés : A, B et C étant trois points du plan, définis par leurs coordonnées, on veut tester s'ils sont alignés. Écrire l'algorithme qui à partir des coordonnées des trois points répond par « Oui » ou « Non » à la question : « A, B et C sont-ils alignés ? » (A7.6, traduction d'une formule)



(A23.2, un PROGRAMME DE MODÉLISATION-SIMULATION)

Phase n° 2 : [résolution dans $\mathbb{N}$ de l'équation $y^2 - x^2 = 24$]
On propose de réaliser cette recherche par balayage.
Ainsi, on prend d'abord $y = 0$,
on fait ensuite varier x de 0 à 24,
on teste à chaque fois si $y^2 - x^2 = 24$
si on a une solution, le logiciel le signale
on passe au y suivant, et on reprend le même processus.
(A26.2, une automatisation de vérification d'un grand nombre de cas)

début
  Donner à *res* la valeur 1
  pour *i de 1 à 10* **faire**
    Donner à *res* la valeur $res * i$
  fin
  Afficher *res*
fin

Algorithme 7 : Factorielle "Pour"

(A29.7, un ALGORITHME-INSTANCIÉ)

Algorithme 1 : Tri d'une série statistique
Entrées :
A : suite de nombres réels (série statistique)
Sorties :
B : suite obtenue en ordonnant A
début
  Trier A
  Afficher B
fin

(A18.1, un exemple d'amalgame programme-algorithme)

Enfin, relevons cette remarque concernant la stratégie gagnante au jeu de NIM dans une activité pour la classe du document A21 :

> Remarque – Peut-être qu'il faut donner la stratégie gagnante aux élèves pour leur permettre de ne pas être bloqués. Ou alors on peut laisser plusieurs jours (semaines ?) aux élèves pour trouver la stratégie gagnante. (A21.43)

On voit bien que la recherche de la stratégie gagnante n'est pas reconnue comme relevant de l'algorithmique, elle est donnée ou laissée en travail personnel, pour se focaliser sur sa programmation. Ici encore, la distinction algorithme-programme n'est pas claire.

Enfin, un autre critère permettant de repérer les programmes qui ne mettent pas en jeu l'algorithme peut être l'équivalence d'utilisation des tableurs et des langages de programmation (on trouve cela dans A23 et A26 par exemple). L'enjeu de ces documents relève alors plus souvent de l'utilisation des TICE que de l'algorithmique :

4. *Avec le tableur*

Supposons que l'animal fût assez coriace pour vivre le temps de mille coupes du monde. Nous allons nous intéresser à chaque fois au nombre de réponses correctes sur huit matchs pronostiqués.

1. Ouvrir le fichier "Paul le Poulpe".
2. Saisir en C4 la formule =ENT(ALEA()*2) : 1 correspond à un pronostic juste; 0 à un faux.
3. Répéter la formule de C8 en J8.
4. Entrer en K8 une formule permettant de compter le nombre de succès de Paul.
5. Répéter l'opération pour simuler les 1000 coupes du monde.
6. En O7, il s'agit de comptabiliser le nombre de coupes du mondes où Paul a donné 0 pronostics corrects. A l'aide de la fonction NB.SI, entrer la formule qui permet ce calcul.
7. Faire de même de O8 à O15.
8. Construire l'histogramme des effectifs de ces résultats.

(A23.3, une expérience qui peut être programmée ou simulée avec un tableur)

2.4 Quels algorithmes dans les ressources ?

Comme nous avons déjà pu le voir, beaucoup d'algorithmes en jeu sont assez pauvres et ne peuvent pas être objet d'étude. On retrouve aussi beaucoup de méthodes effectives qui ressemblent plus à des formules ou à des fonctions (parfois avec conditionnelle). Une grande majorité de ces algorithmes et méthodes se trouve dans les ressources pour la classe.

On rencontre tout de même de nombreux algorithmes qui ont un potentiel pour mettre en jeu l'algorithme en tant qu'objet. Même si on les rencontre régulièrement dans un cadre où ils ne sont qu'outils, il nous semble important de relever que de tels algorithmes sont proposés. En grande majorité ces algorithmes sont dans les ressources pour les enseignants mais on en retrouve aussi quelques uns dans les ressources pour la classe.

Les algorithmes “riches” les plus courants dans les ressources pour les enseignants sont l'algorithme d'Euclide, les recherches dichotomiques, les tris, les tests de primalité et le crible d'Ératosthène. On retrouve aussi beaucoup d'autres algorithmes propres à une étude algorithmique riche que nous ne listerons pas tous ici. Ce sont entre autres des algorithmes de traitement des données (minimum/maximum, médiane, k^e plus petit élément, partition autour d'un pivot, etc.), des algorithmes d'arithmétique (fractions égyptiennes, changement de base, ...), des algorithmes de calcul formel (Karatsuba, Horner, etc.) ou encore des algorithmes d'optimisation combinatoire (algorithmes gloutons exacts et d'approximation, etc.).

Dans les ressources pour la classe, parmi les algorithmes “riches”, on retrouve à plusieurs reprises la dichotomie, l'algorithme d'Euclide et la division euclidienne. Les autres n'apparaissent que dans une ressource mais sont suffisamment peu nombreux pour que nous les citions tous : minimum d'une liste, médiane, tri bulle, simplification de fraction, liste des diviseurs d'un entiers et jeu de NIM. On ne retrouve ici quasiment que les domaines du traitement de données et de l'arithmétique.

D'un point de vue quantitatif, les algorithmes ayant un potentiel pour l'activité algorithmique sont très peu présents dans les ressources pour la classe et leur potentiel n'est jamais exploité.

2.5 Classification des documents

Nous proposons une classification des ressources étudiées, en différentes familles, afin de donner une idée de la répartition des documents [5]. Les ressources à destination des enseignants sont repérées en caractères gras. Nous avons mis de côté A25 (car hors propos) et A16 et A22 (car trop peu d'informations étaient accessibles). A21 est découpé en A21a pour ce qui concerne les enseignants et A21b pour ce qui concerne les activités en classe. Les familles sont les suivantes :

$\mathcal{F}_1 =$ {**A1**} : document mettant en jeu AM-objet avec des opérateurs clairement précisés pour étudier certains problèmes de $\mathbb{P}$, forte présence de la preuve.

$\mathcal{F}_2 =$ {**A2**, **A3**, **A4**} : étude de problèmes de $\mathbb{P}$, avec une forte présence de la preuve et des algorithmes dans AI-objet et AM-objet.

$\mathcal{F}_3 =$ {A5, A6, A7} : ressources pour apprendre un logiciel avec amalgame algorithme-programme.

$\mathcal{F}_4 =$ {**A8**, **A21a**} : ressources pour apprendre l'algorithmique avec introduction pas à pas des diverses instructions d'un pseudo-code, présence de la preuve et de problèmes de $\mathbb{P}$.

$\mathcal{F}_5 =$ {**A9**, A10, A11, A12, A13, **A20**, A21b, **A29**} : ressources pour l'apprentissage des instructions d'un pseudo-code. Certaines ressources proposent une sorte de progression pour introduire leur pseudo-code (A21b, **A9**, **A29**). A20 se rapproche aussi de ces documents sans proposer spécifiquement d'activités. Les autres documents proposent des activités permettant d'introduire certaines instructions en particulier (A10, A11, A12, A13).

$\mathcal{F}_6 =$ {A14, A15, A18, A19} : les ressources pour la classe qui mettent en jeu AI-outil (problèmes dans $\mathbb{P}$). A19 met en plus en jeu la complexité donc AI-objet.

$\mathcal{F}_7 =$ {A17, A23, A24, A26, **A30**} : les documents où l'algorithme n'est pas réellement présent, utilisant la programmation comme outil expérimental.

Conclusions

2.6 Résumé des résultats

Cette analyse met en lumière les points suivants :

- L'algorithme est uniquement OUTIL dans une majorité de ressources, y compris des ressources destinées à la formation des enseignants.
- L'algorithme en tant qu'objet n'est présent que dans les ressources pour la formation des enseignants. En général, cet aspect apparaît via la preuve d'algorithmes.
- Les conceptions sont majoritairement situées dans AI et dans une bien moindre mesure dans AM. PA est absent bien que quelques ressources en soient proches.
- Les algorithmes "riches" mis en jeu sont principalement dans les ressources pour les enseignants. Dans les ressources pour la classe on retrouve très peu de ces algorithmes et ceux présents ne sont pas exploités.

5. Ces familles pourraient constituer un point de départ pour des recherches futures sur les activités proposées en classe par les enseignants.

- De très nombreuses ressources se focalisent sur les aspects de programmation, quitte, souvent, à se concentrer sur des problèmes de très faible intérêt pour l'algorithmique. Cette focalisation sur la programmation explique la prédominance de AI et de l'aspect OUTIL ainsi que l'absence de PA. Les PROGRAMMES-PAPIER sont de bons indicateurs d'un amalgame avec la programmation.
- Une motivation récurrente est la simulation. Cela mène parfois à des activités dont l'algorithme est absent. Dans d'autres cas, il semble que les algorithmes en jeu aient besoin d'être motivés par une question technologique (Comment la calculatrice fait ceci ? Comment avoir un programme qui fait cela pour la suite de l'année ?). Cela est aussi lié à un objectif de programmation.
- On rencontre assez souvent des programmes ou méthodes considérés comme des algorithmes, qui n'en sont pas à notre sens : PROGRAMMES DE MODÉLISATION-SIMULATION et ALGORITHMES INSTANCIÉS.

2.7 Interprétations

Conception

Cet ensemble de ressources nous permet de décrire la conception de l'institution *site des IREM*. Ce point de vue est soutenu par le fait que les ressources sont soumises à un système de relecture par un membre du comité. On pourrait aller plus loin, en interprétant nos constats comme un reflet des conceptions des enseignants et chercheurs ayant produit les ressources. Cette conception est perçue au travers de la transposition qu'ils proposent de l'algorithmique et ne décrit que leur rapport à l'algorithme en tant que membre de l'institution. Cependant, les résultats concernant les conceptions des chercheurs interviewés nous permettent d'appuyer ce point de vue.

La cohabitation de conceptions très diverses de l'algorithme (et parfois presque contradictoires) dans cette institution est tout de même à souligner. De plus, d'autres facteurs entrent en compte dans la production de ces ressources, que l'on peut formuler en termes de conditions et contraintes.

Contraintes

Le besoin urgent de ressources, pour les enseignants, peut être vu comme un poids qui a accéléré la production et la diffusion de ressources du site des IREM.

Cependant, on peut penser que la contrainte principale pour la production des ressources a été le respect des instructions officielles. Les résultats du chapitre précédent montrent que l'attente se situe autour de la programmation de méthodes et l'implémentation d'algorithmes, à insérer le plus souvent dans une démarche d'investigation. Cette approche est très courante dans les ressources IREM. On retrouve aussi, par exemple, l'amalgame entre entres-sorties et interfaces homme-machine, fortement présent dans les programmes.
Ces contraintes des programmes n'expliquent pas tout. Certains documents s'en sont libérés lorsque celles-ci sont trop en contradiction avec leur conception (on peut comprendre que ce soit plutôt les ressources pour la formation des enseignants) alors que d'autres semblent simplement les accepter.
Par exemple, certaines ressources soulignent l'intérêt de voir les algorithmes comme des fonctions et de ne pas confondre saisie et entrée. Alors que l'amalgame entrée-saisie[6] est

6. Tel que nous l'avons défini p. 128.

repris dans d'autres documents, pour lesquels on peut conclure, au moins, que leur conception de l'algorithme n'entre pas en contradiction avec cet amalgame.

Conditions

L'autre contrainte des instructions officielles est celle de traiter l'algorithmique au travers de l'ensemble des thématiques des programmes. En effet, on peut constater que la plupart des documents qui mettent en jeu l'algorithme en tant qu'objet se place dans des champs mathématiques qui sont peu ou pas du tout présents dans les programmes du lycée comme le traitement des données, l'arithmétique, la théorie des graphes [7], la combinatoire, les heuristiques pour l'algorithmique, le calcul formel ou encore la théorie des jeux.
Parallèlement, la plupart des autres documents se place dans le cadre des programmes.

Cette contrainte soulève la question des conditions propices à la présence de l'algorithme-objet dans les mathématiques. L'étude de ces conditions pourrait apporter des pistes permettant de comprendre pourquoi l'algorithme ne peut que difficilement vivre (en tant qu'objet en particulier) dans certains champs des programmes.
Pour répondre à cette problématique, il nous semble qu'une étude écologique du concept serait adaptée. Nous reviendrons sur cette question en conclusion de la thèse, dans les perspectives de travail.

Retour sur les outils

Les résultats obtenus montrent une bonne adaptabilité des outils aux différents types de ressources. Les outils nous ont aussi permis de classifier les ressources d'une manière générale et d'analyser plus spécifiquement certaines activités proposées de manière fine. Cela devrait nous permettre, par la suite, de proposer un outil systématique d'analyse, adapté à l'étude des conceptions dans des documents, chez des personnes ou des institutions, voire même à l'étude de corpus hétérogènes.
Ainsi, nous devrions pouvoir étudier dans les manuels, à la fois les exercices, les activités et les cours sur l'algorithmique.

7. On retrouve arithmétique et théorie des graphes uniquement dans des enseignements de spécialité de terminale.

Chapitre 8

Manuels du lycée

Sommaire

Introduction 177
1 Méthodologie 178
1.1 Collections de manuels étudiées 178
1.2 Hypothèses 178
1.3 Grille d'analyse 179
2 Résultats 181
2.1 Collection Indice 181
2.2 Collection Transmath 187
2.3 Enseignements de spécialité 192
Conclusion 196

Introduction

Après avoir étudié des ressources en ligne pour l'algorithmique, nous proposons d'étudier un autre type de ressources : les manuels de mathématiques pour le lycée. Cela doit permettre de poursuivre l'étude de la transposition didactique en jeu au lycée. Par rapport aux programmes, ils constituent une étape intermédiaire de la transposition et leur étude devrait apporter des précisions quant au savoir à enseigner et nous permettre de formuler des hypothèses sur le savoir enseigné. Ces hypothèses seront à mettre à l'épreuve avec une analyse des pratiques enseignantes [1], et peuvent constituer une analyse préalable pour concevoir un outil d'étude de ces pratiques.

À la suite du développement de nos modèles épistémologiques, nous avions reformulé deux questions de recherche pour les manuels :

Question $Q6_{\text{aspects}}$. *Quels aspects de l'algorithme sont présents dans les manuels de mathématiques ? Les aspects outils et objets sont-ils représentés ?*

Question $Q6_{\text{conceptions}}$. *Quelles conceptions sont représentées dans les manuels de mathématiques du lycée ? Les conceptions présentes sont-elles complètes ? Sont-elles modifiées par rapport à celles du savoir savant ? Relèvent-elles de l'outil ou de l'objet ? Quelle place est donnée aux structures de contrôle ?*

1. Nous ne réaliserons pas de telle étude ici mais il s'agit d'une poursuite du travail indispensable pour l'analyse de la transposition.

Nous allons donc nous attacher à répondre à ces questions en essayant de développer une étude quantitative sur les manuels étudiés.

1 Méthodologie

1.1 Collections de manuels étudiées

Nous avons sélectionné deux collections de manuels de mathématiques pour cette étude. Nous avons choisi des collections de deux éditeurs différents. Ces deux collections proposent des manuels pour toutes les filières et tous les niveaux. Il s'agit des collections *Indice*, éditée par *Bordas*, et *Transmath*, éditée par *Nathan*.

Pour ces deux collections nous étudierons tous les manuels, sans les spécialités, c'est-à-dire, seconde 1e ES-L, Te ES-L, 1e S et Te S. Nous étudierons ces manuels suivant une même méthodologie.

À ces manuels nous ajoutons ceux des spécialités mathématiques en Te S et ES dans la collection *Indice*. Étant donné les différences entre les programmes du lycée et les programmes de ces deux spécialités, nous choisirons de les étudier à part.

Pour cet ensemble de manuels, notre objectif est non seulement d'étudier la transposition en jeu dans les manuels mais aussi de montrer comment la grille d'analyse que nous avons utilisée jusqu'à maintenant peut être utilisée de manière systématique.

1.2 Hypothèses

Avant de proposer une grille pour l'étude quantitative, nous devons préciser les critères qui nous guideront. Nous faisons l'hypothèse que les manuels vont très peu sortir du cadre donné par les instructions officielles. Nous proposerons donc des critères d'analyse en fonction des résultats du chapitre 6, dont les conclusions nous poussent à faire les hypothèses de recherche suivantes [2] :

Hypothèse 1 Nous nous attendons à rencontrer des nombreux "algorithmes" mettant en jeu des formules, implémentant des fonctions, simulant des expériences aléatoires, calculant des termes de suites, etc. Ces algorithmes ne permettent pas de mettre en jeu les aspects objet. Nous étudierons spécifiquement les algorithmes qui relèvent de notre définition, que nous supposons plus rares, et qui peuvent mettre en jeu l'algorithme-objet.

Nous avons pu constater que, dans les programmes, ces algorithmes plus riches relevaient principalement des méthodes numériques. Nous quantifierons la présence de ces méthodes numériques.

Hypothèse 2 Nous supposons que les questions majoritairement posées sur les algorithmes seront de l'ordre de l'écriture et de la mise en œuvre (en particulier : écriture d'algorithmes, de programmes, exécution, implémentation d'autres notions mathématiques et

2. Ces hypothèses seront confrontées aux résultats de l'analyse, ce qui permettra de comparer la transposition dans les manuels à celle proposée par les programmes. Ces hypothèses nous permettent aussi de proposer une grille ayant une granularité correspondant aux attentes, quitte à la modifier si besoin.

de simulations, ...). Nous pensons que les manuels vont centrer l'algorithmique sur une activité d'écriture, comme cela est le cas dans les programmes. Nous quantifierons les questions relatives au langage (sa maîtrise, sa compréhension, sa manipulation).

Hypothèse 3 Le paradigme AI sera très majoritaire dans les manuels. Nous serons attentifs à la part d'algorithmes présentés ou demandés en langages de programmation et à la présence de PROGRAMMES-PAPIER.

Hypothèse 4 La validation des algorithmes sera peu présente. Elle sera de l'ordre du langage (via la programmation), du test (via l'exécution sur des exemples), ou découlera de la validité des méthodes mathématiques traduites (lorsque par exemple on implémentera une méthode d'approximation). Les structures de contrôle spécifiques à l'algorithme seront absentes.

Hypothèse 5 On devrait constater de grandes variations entre les différents domaines mathématiques abordés concernant la présence d'algorithmes et leur rôle.

Pour les manuels de spécialité, nous ne faisons pas d'hypothèse étant donné le contenu de leurs programmes concernant l'algorithmique. Nous nous attendons simplement à rencontrer des algorithmes plus riches, au vu des thématiques abordées.

1.3 Grille d'analyse

Ces hypothèses, basées sur l'analyse des instructions officielles, jouent aussi le rôle d'analyse préalable. Elles nous donnent une idée de ce que l'on peut attendre des manuels et nous permettent de proposer une grille d'analyse adaptée. Par exemple, nous ne proposerons pas une granularité trop fine concernant les aspects objet que nous attendons peu nombreux.

Nous étudierons deux points dans les manuels : les algorithmes proposés et les questions posées dans les exercices proposés à l'élève. Tous les algorithmes reconnus par le manuel seront pris en compte. Pour les questions, nous étudierons celles qui sont posées à l'élève dans toutes les activités qui sont à sa charge, c'est-à-dire dans les exercices, exercices résolus, les points méthode, les TP et les activités.

Analyse quantitative

C'est surtout cette partie qui va nous intéresser, afin de quantifier les types d'activités proposés à l'élève. En particulier, nous étudierons le nombre d'algorithmes de différents types, les questions les plus souvent posées sur les algorithmes, les systèmes de représentation majoritaires et les structures de contrôle les plus répandues.

Cela concerne donc les questions suivantes :

- Algorithmes en jeu :
  - Quels types d'algorithmes sont proposés ?
  - En quelles proportions ?
  - Sont-ils des algorithmes selon notre définition ?
- Questions :
  - Quelles sont les principales questions posées à l'élève concernant ces algorithmes ?
  - Cela varie-t-il selon les chapitres ?
- Aspects :
  - Quels ASPECTS les questions mettent-elles en avant ?
  - Relèvent-ils de l'outil ou de l'objet ?
- Conceptions :
  - Dans quel(s) paradigme(s) se trouve-t-on ? (en lien avec le système de représentation)
  - Quelles conceptions de l'algorithme sont majoritaires ?

Concernant *algorithmes* et *questions*, il nous faut préciser selon quels critères nous les classifions avant de nous lancer dans l'analyse quantitative.

Types d'algorithmes Nous proposons la classification suivante. Elle inclue de nombreuses procédures que nous ne considérons pas comme des algorithmes mais qui sont présentées comme telles dans les programmes. On retrouve aussi beaucoup de catégories d'algorithmes que nous avons considérées comme n'ayant pas d'intérêt pour pouvoir faire vivre l'aspect objet. Notre hypothèse étant que ces algorithmes seront nombreux, nous avons choisi de tout de même laisser apparaître les nuances qu'ils recouvrent.

- Formules
- Formules conditionnelles
- Formule "pour" : somme ou produit de termes, répétition d'un nombre fixé d'instructions
- "Programmes de calcul" et manipulations de variables
- Constructions géométriques et autres constructions en nombre d'étapes fixe
- Calcul des termes d'une suite, ou équivalent
- Recherche du premier terme d'une suite vérifiant une propriété (seuils, balayages...), ou équivalent
- Simulation d'une expérience aléatoire ou d'une série d'expériences
- Méthodes numériques (méthodes d'approximation de zéros, d'aires...)
- Autres algorithmes (nous les étudierons à part)

Précisons que pour chaque manuel, nous dénombrons l'ensemble des algorithmes présents dans le cours, ainsi que ceux dans les exercices d'algorithmique (dans toutes les collections, les exercices et activités mettant en jeu des algorithmes sont marqués par un logo spécifique). Nous n'étudierons pas les exercices non repérés comme tels : l'objectif étant d'étudier le rapport à l'algorithme, il ne nous semble pas pertinent ici de chercher les algorithmes non répertoriés [3].

Type de questions Nous nous appuierons sur le découpage suivant, inspiré par nos hypothèses. Nous avons modifié certaines catégories a posteriori pour que la granularité

3. Ce pourrait être l'objet d'un travail futur, consistant à étudier les champs où l'algorithme n'est pas reconnu malgré sa présence. Nous n'aborderons pas cette question ici mais l'on peut affirmer que certains algorithmes ne sont pas reconnus comme tels dans certains manuels. Citons, par exemple, le raisonnement par récurrence qui n'est jamais associée à l'algorithmique, la recherche de cycles eulériens dans certains manuels de T^{e} ES, ou encore certaines méthodes numériques lorsqu'elles ne sont pas implémentées.

soit adaptée à la présentation.

- Écrire un algorithme qui fait...
- Écrire un programme qui fait...
- Implémenter un algorithme donné ou écrit précédemment
- Compléter ou modifier un algorithme ou un programme
- Questions de compréhension des instructions de l'algorithme ou du programme (décrire ce qu'il fait, à quoi sert telle variable, telle instruction,...)
- Exécuter un algorithme à la main (parfois en décrivant chaque pas)
- Faire exécuter un programme (sur une entrée donnée ou au choix)
- Écrire l'algorithme d'après un programme
- Comparer deux algorithmes
- Étudier la complexité d'un algorithme
- Justifier ou valider un algorithme
- Autres (dans ce cas, préciser)

Nous repèrerons ces questions dans les exercices des manuels et dans les autres activités où l'élève est questionné (TP, "Pour aller plus loin", activité, etc). Lorsqu'une question est répétée plusieurs fois dans un même exercice, nous ne la compterons qu'une fois (par exemple une consigne du genre "tester votre programme pour 1, 4, 13 et 19" sera comptabilisée comme une seule question du type *exécuter un programme*).

Ces types de questions peuvent être repérés comme relevant de l'outil ou de l'objet, des opérateurs, du langage ou des structures de contrôle.

Cours et discours

Pour tenir compte de l'intégralité de l'algorithmique proposée par les manuels, nous utiliserons la grille des deux chapitres précédents pour étudier les conceptions présentes. En particulier, cela nous permettra de prendre en compte les parties de cours et les discours sur l'algorithmique. Nous avons vu que dans les discours les ASPECTS sont souvent plus faciles à repérer, mais les exemples et illustrations devraient permettre de décrire des conceptions. Nous confronterons les résultats avec les statistiques des exercices.

2 Résultats

2.1 Collection Indice (Bordas)

Algorithmes

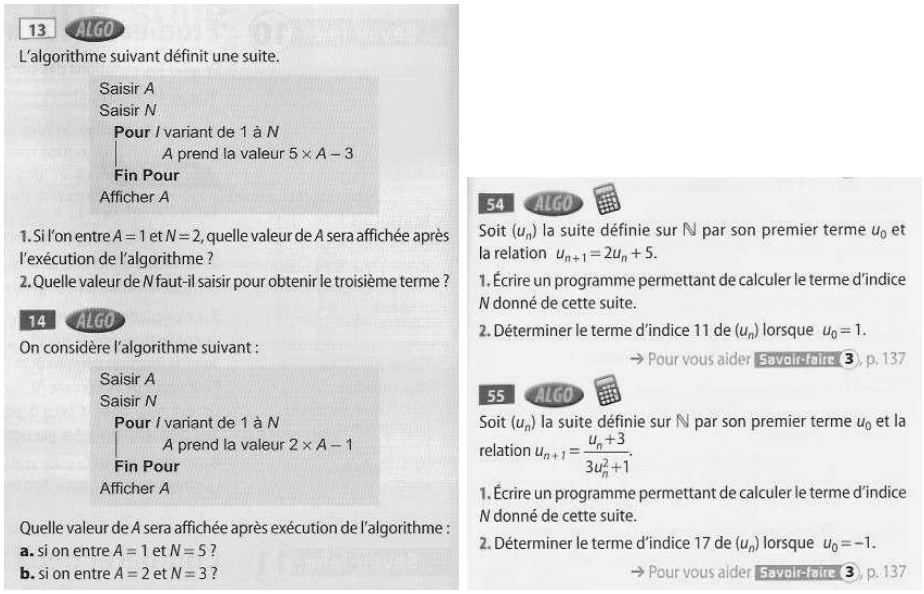
Les algorithmes présents dans les manuels se répartissent selon le tableau ci-après [4].
On retrouve toutes les attentes spécifiques des programmes en algorithmique. En particulier les *capacités attendues* des programmes sont présentes (zéro par dichotomie en seconde, calcul d'un terme d'une suite en 1e ES-L, seuil pour (q^n), $0 < q < 1$ en Te ES-L, terme d'une suite et liste des termes en 1e S, seuil pour une suite croissante et encadrement d'intégrales en Te S). Quasiment aucun algorithme ne sort des propositions des programmes [5].
Dans chaque classe, on peut d'ailleurs remarquer des pourcentages plus élevés dans les catégories d'algorithmes correspondant aux *capacités attendues* de la classe (à l'exception le la dichotomie).

4. Nous indiquons des effectifs et des pourcentages dans chacun des tableaux de ce chapitre, afin que le lecteur puisse percevoir facilement, à la fois le nombre d'éléments représentés par la catégorie, et la proportion que cela représente. Il ne faut pas voir plus que des tendances dans l'ensemble de ces statistiques.

5. Le manuel de Te S propose d'ailleurs une liste des algorithmes *exigibles* à la fin de ses compléments d'algorithmique correspondant aux algorithmes des *capacités attendues* des programmes.

Indice	Formules	Formules cond.	Formule "Σ"	Prog. de calc.	Const. géom.	Suite	Suite "tant que"	Simul. aléa.	Méthode num.	Autres
Seconde	7 12%	12 21%	8 14%	3 5%	5 9%	7 12%	6 10%	9 16%	1 2%	- -
1e ES-L	8 29%	4 14%	1 4%	- -	- -	9 32%	3 11%	3 11%	- -	- -
Te ES-L	7 28%	2 8%	1 4%	- -	- -	3 12%	7 28%	2 8%	3 12%	- -
1e S	7 16%	10 23%	1 2%	- -	- -	13 30%	3 7%	6 14%	4 9%	- -
Te S	18 33%	7 13%	2 4%	- -	- -	3 6%	9 17%	4 7%	11 20%	- -

On retrouve une très grande majorité d'algorithmes pauvres pour l'aspect OBJET. Seules les méthodes numériques, ici, peuvent mettre en jeu l'aspect OBJET. Cependant, beaucoup de ces algorithmes représentent en fait des suites dont on veut calculer le premier terme qui est plus petit qu'un ε donné. Pour ce genre de suites comme pour les méthodes numériques rencontrées, les manuels proposent beaucoup d'ALGORITHMES INSTANCIÉS (notamment en Te ES-L,). Il s'agit alors souvent de modifier l'algorithme dès que l'on veut changer la fonction étudiée, l'intervalle d'étude, le pas ou la précision. Nous pensons que cela peut être lié au besoin de proposer plusieurs exercices pour une même technique : si l'algorithme est considéré comme une technique, il devient nécessaire d'avoir "plusieurs" algorithmes de recherche de seuil d'une suite par exemple. C'est le cas de l'exemple suivant :



13 ALGO

L'algorithme suivant définit une suite.

Saisir A
Saisir N
Pour I variant de 1 à N
  A prend la valeur $5 \times A - 3$
Fin Pour
Afficher A

1. Si l'on entre $A = 1$ et $N = 2$, quelle valeur de A sera affichée après l'exécution de l'algorithme ?
2. Quelle valeur de N faut-il saisir pour obtenir le troisième terme ?

14 ALGO

On considère l'algorithme suivant :

Saisir A
Saisir N
Pour I variant de 1 à N
  A prend la valeur $2 \times A - 1$
Fin Pour
Afficher A

Quelle valeur de A sera affichée après exécution de l'algorithme :
a. si on entre $A = 1$ et $N = 5$?
b. si on entre $A = 2$ et $N = 3$?

54 ALGO

Soit (u_n) la suite définie sur $\mathbb{N}$ par son premier terme u_0 et la relation $u_{n+1} = 2u_n + 5$.

1. Écrire un programme permettant de calculer le terme d'indice N donné de cette suite.
2. Déterminer le terme d'indice 11 de (u_n) lorsque $u_0 = 1$.

→ Pour vous aider Savoir-faire 3, p. 137

55 ALGO

Soit (u_n) la suite définie sur $\mathbb{N}$ par son premier terme u_0 et la relation $u_{n+1} = \frac{u_n + 3}{3u_n^2 + 1}$.

1. Écrire un programme permettant de calculer le terme d'indice N donné de cette suite.
2. Déterminer le terme d'indice 17 de (u_n) lorsque $u_0 = -1$.

→ Pour vous aider Savoir-faire 3, p. 137

Indice 1e L-ES : Quatre exercices mettant en jeu le calcul d'un terme d'une suite fixée.

Cela peut aussi être lié au fait de ne pas pouvoir définir de sous-fonction avec la notion d'"entrée-sortie" des programmes (dans notre exemple, pour calculer u_N, connaissant u_0, N et f telle que $u_{n+1} = f(u_n)$: l'algorithme peut prendre en entrée u_0 et N mais il ne peut pas prendre une fonction f générique car une telle fonction ne peut pas être décrite séparément et "appelée" dans l'algorithme).

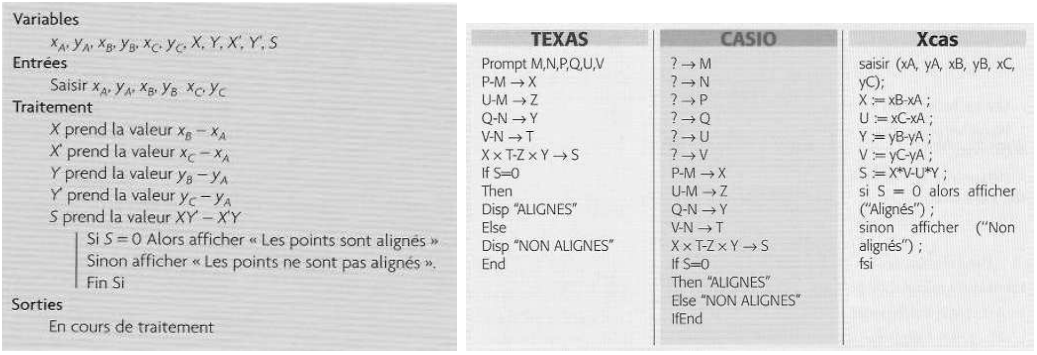
Les types d'algorithmes en jeu montrent bien que l'aspect dominant est l'EFFECTIVITÉ. Il s'agit d'algorithmes pour mettre en œuvre une méthode. L'autre aspect présent est l'aspect PROBLÈME. Cependant, la présence d'ALGORITHMES INSTANCIÉS, de PROGRAMMES DE MODÉLISATION-SIMULATION et de beaucoup de mises en œuvre de formules montre que cet aspect n'est pas fondamental ici.

Forme des algorithmes

Les algorithmes vivent quasi-exclusivement sous deux formes :

- les programmes proposés dans différents langages (Casio, TI, Algobox, etc.)
- les *algorithmes*, qui suivent certains critères formels bien précis : une structure globale (initialisation, traitement, sortie), une liste de commandes et d'opérateurs autorisés (boucle, affectation), une gestion des entrées-sorties par saisie-affichage et une déclaration des variables dans l'entête.

On reconnaît ce que nous avons introduit sous le nom de PROGRAMME-PAPIER. On constate dans ces manuels que le terme *algorithme* signifie PROGRAMME-PAPIER si rien n'est précisé. Nous reviendrons sur ce point dans l'étude des "chapitres" *algorithmique*.



Variables
$x_A, y_A, x_B, y_B, x_C, y_C, X, Y, X', Y', S$
Entrées
Saisir $x_A, y_A, x_B, y_B, x_C, y_C$
Traitement
X prend la valeur $x_B - x_A$
X' prend la valeur $x_C - x_A$
Y prend la valeur $y_B - y_A$
Y' prend la valeur $y_C - y_A$
S prend la valeur $XY' - X'Y$
Si $S = 0$ Alors afficher « Les points sont alignés »
Sinon afficher « Les points ne sont pas alignés ».
Fin Si
Sorties
En cours de traitement

TEXAS

```
Prompt M,N,P,Q,U,V
P-M → X
U-M → Z
Q-N → Y
V-N → T
X × T-Z × Y → S
If S=0
Then
Disp "ALIGNES"
Else
Disp "NON ALIGNES"
End
```

CASIO

```
? → M
? → N
? → P
? → Q
? → U
? → V
P-M → X
U-M → Z
Q-N → Y
V-N → T
X × T-Z × Y → S
If S=0
Then "ALIGNES"
Else "NON ALIGNES"
IfEnd
```

Xcas

```
saisir (xA, yA, xB, yB, xC,
yC);
X := xB-xA ;
U := xC-xA ;
Y := yB-yA ;
V := yC-yA ;
S := X*V-U*Y ;
si S = 0 alors afficher
("Alignés") ;
sinon afficher ("Non
alignés") ;
fsi
```

Indice seconde : Test de l'alignement de trois points, version PROGRAMME-PAPIER et programmes informatiques.

En seconde, quelques algorithmes sont présentés en langage naturel. C'est alors toujours dans un cas où l'on doit décrire une procédure de la vie quotidienne, ou une activité matérielle en mathématiques (comme les constructions géométriques). Ces algorithmes en langage naturel se situent au début du chapitre qui introduit l'algorithmique, avant que ne soit introduit le formalisme.

Cela témoigne d'une domination des conceptions de AI. Nous considérons que AM n'est pas vraiment présent étant donnée la forme imposée des algorithmes et ses liens avec les langages de programmation.

Questions posée à l'élève

Les questions posées sur les algorithmes se répartissent selon le tableau ci-après.
On peut tout d'abord noter que les questions présentes ne concernent que l'algorithme en tant qu'outil : ni validité, ni complexité, ni comparaison d'algorithmes ne sont questionnées[6]. Nous n'avons pas considéré que l'équivalence entre programmes de calcul[7], dans

6. Cela ne signifie pas nécessairement qu'elles sont absentes, mais qu'elles ne relèvent pas de l'activité attendue de l'élève. Nous reviendrons sur cette question plus loin.

7. Un programme de calcul est une suite d'opérations de calcul à appliquer successivement à un nombre quelconque, comme : prendre un nombre, ajouter 2, élever au carré, diviser par 3.

Indice	Écrire algo.	Écrire prog.	Algo.→ prog.	Modif.	Compr.	Exéc. algo.	Exéc. prog.	Prog.→ algo.
Seconde	38 44%	3 3%	11 13%	2 2%	12 14%	8 9%	10 12%	2 2%
1e ES-L	11 24%	- -	13 28%	6 13%	4 9%	6 13%	6 13%	- -
Te ES-L	6 11%	- -	11 21%	8 15%	16 30%	2 4%	10 19%	- -
1e S	22 21%	2 2%	23 22%	13 12%	18 17%	5 5%	19 18%	3 3%
Te S	25 26%	4 4%	17 18%	13 14%	12 13%	5 5%	19 20%	- -

le cadre du calcul algébrique, relève de la comparaison d'algorithmes. Les questions du type "Que fait l'algorithme ?" présentes dans ces manuels mettent en jeu uniquement la compréhension des instructions du langage [8].

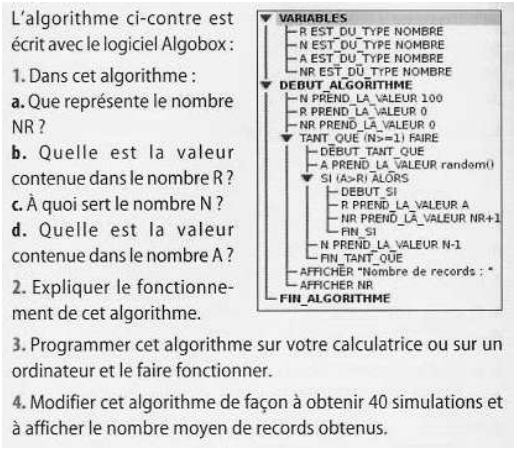
De même, concernant les conceptions, on constate que les éléments mis en jeu dans les questions sont principalement les opérateurs et les systèmes de représentation. On est essentiellement dans une activité du type : problème → écriture d'algorithme → écriture de programme → exécution. Précisons d'ailleurs que la relation algorithme-problème est souvent très simple : il s'agit bien souvent d'adapter la question posée aux contraintes du langage du PROGRAMME-PAPIER. L'écriture du programme devient souvent une traduction. Comme dans les programmes et le document ressource, on est dans une activité centrée sur le langage.

Les questions de compréhension sont aussi très présentes. Elles relèvent presque toujours du langage ou de la compréhension des instructions du langages (valeurs des variables, choix des critères d'arrêt, etc.). Modifications ou extensions d'algorithmes sont souvent utilisées pour faire modifier les sorties, répéter l'exécution d'un algorithme déjà écrit ou changer un paramètre (qui aurait simplement pu être en entrée). Encore une fois, ces questions de modification ont un rôle qui est lié à l'impossibilité de produire des algorithmes comme fonction : on ne crée donc pas un nouvel algorithme faisant appel à l'ancien ou le généralisant, on apporte des modifications à l'ancien.

Enfin, on retrouve aussi beaucoup d'algorithmes à trous dans lesquels il faut compléter une affectation ou une condition. Ces types de questions mettent en avant la compréhension du langage et des instructions associées à l'algorithmique.

Voici un exemple mettant en jeu des questions de compréhension et de modification sur un PROGRAMME DE MODÉLISATION-SIMULATION :

8. De plus, nous avons souvent pu constater que beaucoup d'*effets de contrat* permettent de répondre à ces questions. Par exemple, la phrase qui doit s'afficher avec la sortie indique ce que fait l'algorithme.



L'algorithme ci-contre est écrit avec le logiciel Algobox :

1. Dans cet algorithme :
a. Que représente le nombre NR ?
b. Quelle est la valeur contenue dans le nombre R ?
c. À quoi sert le nombre N ?
d. Quelle est la valeur contenue dans le nombre A ?
2. Expliquer le fonctionnement de cet algorithme.

```
VARIABLES
    R EST_DU_TYPE NOMBRE
    N EST_DU_TYPE NOMBRE
    A EST_DU_TYPE NOMBRE
    NR EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    N PREND_LA_VALEUR 100
    R PREND_LA_VALEUR 0
    NR PREND_LA_VALEUR 0
    TANT_QUE (N>=1) FAIRE
        DEBUT_TANT_QUE
        A PREND_LA_VALEUR random()
        SI (A>R) ALORS
            DEBUT_SI
            R PREND_LA_VALEUR A
            NR PREND_LA_VALEUR NR+1
            FIN_SI
        N PREND_LA_VALEUR N-1
        FIN_TANT_QUE
    AFFICHER "Nombre de records : "
    AFFICHER NR
FIN_ALGORITHME
```

3. Programmer cet algorithme sur votre calculatrice ou sur un ordinateur et le faire fonctionner.

4. Modifier cet algorithme de façon à obtenir 40 simulations et à afficher le nombre moyen de records obtenus.

Indice 1e S : Simulation de records (extrait).

Algorithmique

Les manuels de la collection *Indice* contiennent tous un chapitre ou une section consacrés à l'algorithmique. En seconde, il s'agit d'une introduction sous la forme cours-exercices. Dans les autres manuels, il s'agit simplement d'un *complément* qui rappelle les notions. Dans tous les cas, l'organisation suit le même plan :

- Quelques exemples d'algorithmes (uniquement en seconde),
- L'affectation,
- Programmation d'un calcul itératif avec un nombre d'itérations borné,
- Programmation d'une instruction conditionnelle,
- Programmation d'un calcul itératif avec une fin de boucle conditionnelle,
- Récapitulatif des instructions utilisées en programmation.

Chacun de ces chapitres introduit donc une nouvelle instruction. L'algorithmique est organisée autour de l'écriture de procédures à l'aide de ces instructions.

La définition proposée est toujours la suivante :

> Un **algorithme** est l'énoncé d'une suite d'instructions permettant de donner la réponse à un problème. Il comprend une phase **d'initialisation** (on entre les données), une phase de **traitement** du problème et une phase de **sortie** des résultats.

Cette définition, très brève, met en évidence les aspects EFFECTIVITÉ et PROBLÈME mais sans les détailler (pas de précisions concernant finitude, instances, ...). Elle insiste surtout sur le découpage : initialisation, traitement, sortie. On retrouve ce découpage dans tous les algorithmes proposés, quitte à avoir parfois des sorties du type "affiché en cours de traitement" comme dans l'exemple de la partie *forme des algorithmes* ci-dessus.

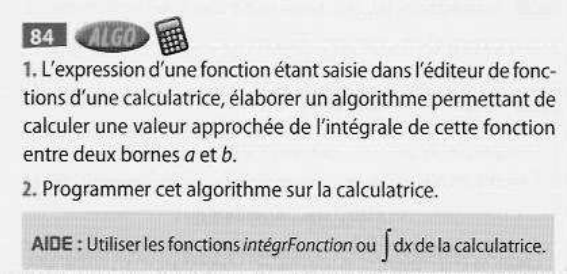
Aspects et conceptions

Nous avons vu que les seuls aspects présents sont les aspects outils : EFFECTIVITÉ et PROBLÈME. Et le rapport au problème est vu dans un sens très vague : il est seulement présent derrière l'idée que l'on écrit un algorithme dans un but précis, pour obtenir un résultat demandé.
Les autres aspects sont absents.

Concernant les conceptions en jeu, nous avons déjà vu que le système de représentation dominant est celui de AI. Revenons sur les problèmes, opérateurs et structures de contrôle.

Les problèmes en jeu sont ceux qui apparaissent au travers des algorithmes présentés. Nous avons déjà abordé cela plus haut : beaucoup sont pauvres pour l'algorithmique et les autres sont peu exploités.
Les questions posées sur les algorithmes montrent que l'algorithme n'est pas objet dans l'activité de l'élève. En fait, il ne l'est pas non plus dans le discours des manuels. Pour la dichotomie par exemple, présente dans le manuel de seconde, celui de T^e ES-L et celui de T^e S, le choix de la dichotomie par rapport à une autre méthode n'est jamais motivé et la complexité ou l'efficacité n'est pas évoquée.

Les opérateurs utilisés sont les opérateurs des langages de programmation et les fonctions mathématiques autorisées dans certains langages, ainsi que leurs équivalents dans les PROGRAMMES-PAPIERS. Cela donne lieu à des algorithmes pauvres qui utilisent des fonctions complexes pré-implémentées. Par exemple :


84 ALGO

1. L'expression d'une fonction étant saisie dans l'éditeur de fonctions d'une calculatrice, élaborer un algorithme permettant de calculer une valeur approchée de l'intégrale de cette fonction entre deux bornes *a* et *b*.
2. Programmer cet algorithme sur la calculatrice.

AIDE : Utiliser les fonctions *intégrFonction* ou $\int dx$ de la calculatrice.

Indice T^e S : Calcul d'intégrale en utilisant les primitives.

Des structures de contrôle sont présentes. Elles peuvent être considérées comme de l'ordre des mathématiques, mais ce ne sont pas des contrôles relatifs à l'algorithmique. C'est-à-dire que la méthode implémentée (formule, suite, fonction, ...) est validée comme répondant au problème, soit dans l'exercice, soit dans le cours. L'algorithme n'étant qu'une traduction de cette méthode, il ne nécessite pas de validation spéciale. Les seules structures de contrôle nécessaires sont alors celles du bon fonctionnement des programmes.

Bilan

- Seul les aspects outil sont présents et les algorithmes proposés ne permettent pas de considérer l'algorithme comme un objet. Les quelques algorithmes qui pourraient mettre en jeu l'algorithme-objet ne le questionnent pas.
- L'aspect PROBLÈME est lui-même peu reconnu : on retrouve des ALGORITHMES INSTANCIÉS et les exercices consistent souvent à reproduire un algorithme connu pour une instance fixée. On trouve aussi des PROGRAMMES DE MODÉLISATION-SIMULATION.
- La majorité des questions soulevées concerne le langage d'expression des algorithmes et l'algorithmique est introduite instruction par instruction.
- On rencontre d'ailleurs uniquement des programmes et des PROGRAMMES-PAPIER. C'est donc le paradigme AI qui domine largement et uniquement par la conception AI-outil.

2.2 Collection Transmath (Nathan)

Beaucoup de points sont communs avec l'analyse des manuels de la collection Indice. Nous passerons plus rapidement et montrerons les quelques différences repérées.

Algorithmes

Les algorithmes présents dans les manuels se répartissent ainsi :

Transmath	Formules	Formules cond.	Formule "Σ"	Prog. de calc.	Const. géom.	Suite	Suite "tant que"	Simul. aléa.	Méthode num.	Autres
Seconde	7 11%	19 29%	7 11%	22 33%	2 3%	- -	4 6%	5 8%	- -	- -
1e ES-L	4 11%	4 11%	- -	1 3%	- -	11 29%	8 21%	9 24%	1 3%	- -
Te ES-L	6 15%	4 10%	- -	- -	- -	13 33%	7 18%	7 18%	3 8%	- -
1e S	8 13%	11 17%	3 5%	8 13%	1 2%	16 25%	7 11%	5 8%	2 3%	2 3%
Te S	8 18%	5 11%	- -	2 4%	2 4%	3 7%	3 7%	11 24%	8 18%	3 7%

Le constat que l'on peut faire est le même que pour les manuels de la collection Indice :

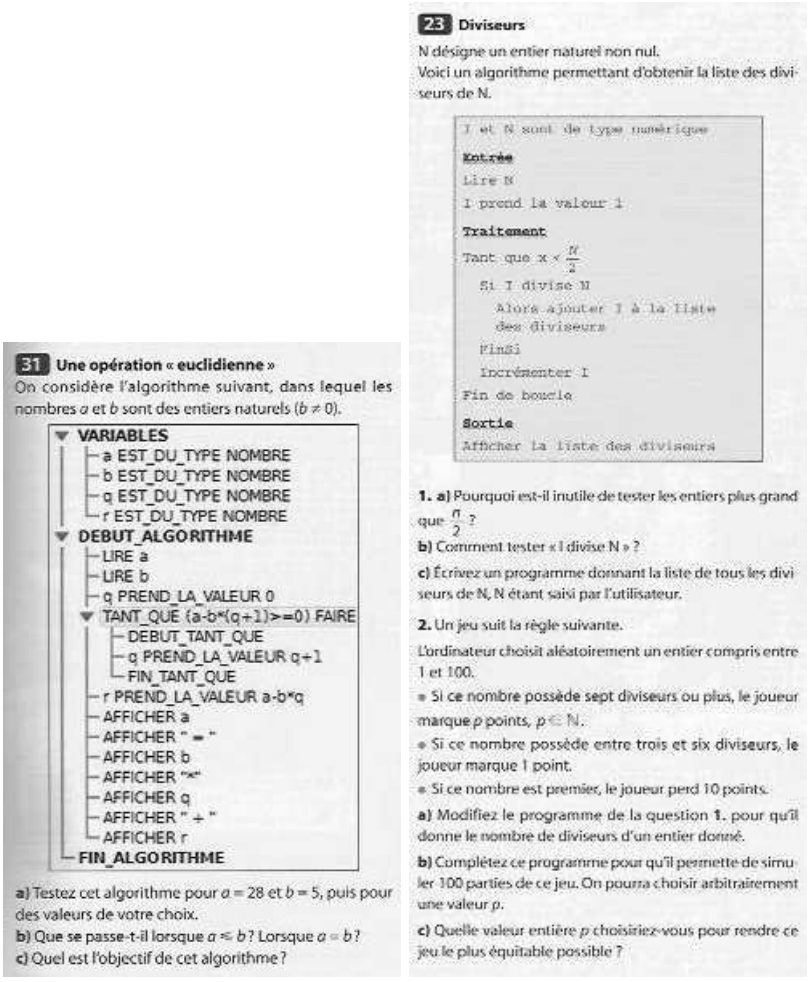
- On retrouve tous les algorithmes des programmes, en particulier ceux des *capacités attendues*[9].
- Les classes principales sont celles que nous considérons pauvres pour l'algorithme-objet.
- L'aspect dominant est l'EFFECTIVITÉ.
- On retrouve des ALGORITHMES-INSTANCIÉS, notamment dans les exercices sur les suites (surement pour les mêmes raisons que celles que nous avons déjà évoquées).
- Ces ALGORITHMES INSTANCIÉS, le nombre d'exercices de mise en œuvre de formules et la présence de PROGRAMMES DE MODÉLISATION-SIMULATION, montrent que l'aspect PROBLÈME n'est pas primordial ici non plus.

Nous avons noté que l'on trouve tout de même quelques algorithmes plus riches. Ces algorithmes sont dans la catégorie "méthodes numériques" mais aussi hors du classement. On trouve notamment :

- 1e ES-L : recherche de zéro par dichotomie,
- Te ES-L : recherche de zéro par dichotomie et par balayage,
- 1e S : division euclidienne par soustraction, dichotomie pour rechercher un nombre entre 1 et 100,
- Te S : recherche de zéro par dichotomie, algorithme d'Euclide, algorithme pour la liste des diviseurs d'un entier.

Dans ces algorithmes, pourtant, ce sont uniquement les aspects Outil qui sont mis en avant. Par exemple, comme précédemment, la dichotomie n'est pas justifiée par son efficacité. Un autre exemple est l'algorithme de division euclidienne suivant :

9. À l'exception de la recherche de zéro par dichotomie en seconde.



31 Une opération « euclidienne »

On considère l'algorithme suivant, dans lequel les nombres a et b sont des entiers naturels ($b \neq 0$).

```
VARIABLES
    a EST_DU_TYPE NOMBRE
    b EST_DU_TYPE NOMBRE
    q EST_DU_TYPE NOMBRE
    r EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    LIRE a
    LIRE b
    q PREND_LA_VALEUR 0
    TANT_QUE (a-b*(q+1)>=0) FAIRE
        DEBUT_TANT_QUE
        q PREND_LA_VALEUR q+1
        FIN_TANT_QUE
    r PREND_LA_VALEUR a-b*q
    AFFICHER a
    AFFICHER " = "
    AFFICHER b
    AFFICHER "*"
    AFFICHER q
    AFFICHER " + "
    AFFICHER r
FIN_ALGORITHME
```

a) Testez cet algorithme pour $a = 28$ et $b = 5$, puis pour des valeurs de votre choix.
b) Que se passe-t-il lorsque $a < b$? Lorsque $a = b$?
c) Quel est l'objectif de cet algorithme ?

23 Diviseurs

N désigne un entier naturel non nul.
Voici un algorithme permettant d'obtenir la liste des diviseurs de N.

```
I et N sont de type numérique
Entrée
Lire N
I prend la valeur 1
Traitement
Tant que x < N/2
  Si I divise N
    Alors ajouter I à la liste
    des diviseurs
  FinSi
  Incrémenter I
Fin de boucle
Sortie
Afficher la liste des diviseurs
```

1. a) Pourquoi est-il inutile de tester les entiers plus grand que $\frac{n}{2}$?
b) Comment tester « I divise N » ?
c) Écrivez un programme donnant la liste de tous les diviseurs de N, N étant saisi par l'utilisateur.

2. Un jeu suit la règle suivante.
L'ordinateur choisit aléatoirement un entier compris entre 1 et 100.
- Si ce nombre possède sept diviseurs ou plus, le joueur marque p points, $p \leqslant N$.
- Si ce nombre possède entre trois et six diviseurs, le joueur marque 1 point.
- Si ce nombre est premier, le joueur perd 10 points.

a) Modifiez le programme de la question **1.** pour qu'il donne le nombre de diviseurs d'un entier donné.
b) Complétez ce programme pour qu'il permette de simuler 100 parties de ce jeu. On pourra choisir arbitrairement une valeur p.
c) Quelle valeur entière p choisiriez-vous pour rendre ce jeu le plus équitable possible ?

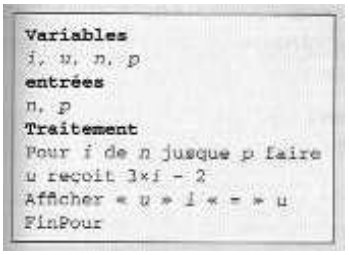
Transmath 1e S et Te S : Division euclidienne et énumération des diviseurs "outils"

Forme des algorithmes

Les constats que nous avons faits pour la collection *Indice* sont les mêmes ici. On a deux formes d'algorithmes :

- les programmes (TI, Casio, Algobox, etc.),
- des PROGRAMMES-PAPIERS suivant la forme entrée-saisie, traitement, sortie-affichage avec déclaration préalable des variables. À cela s'ajoute une mise en forme des résultats propre à l'affichage.

On est dans le paradigme AI.



```
Variables
i, u, n, p
entrées
n, p
Traitement
Pour i de n jusque p faire
u reçoit 3×i - 2
Afficher « u » i « = » u
FinPour
```

Transmath 1e S : PROGRAMME-PAPIER pour l'affichage d'une table de multiplication.

Questions posées à l'élève

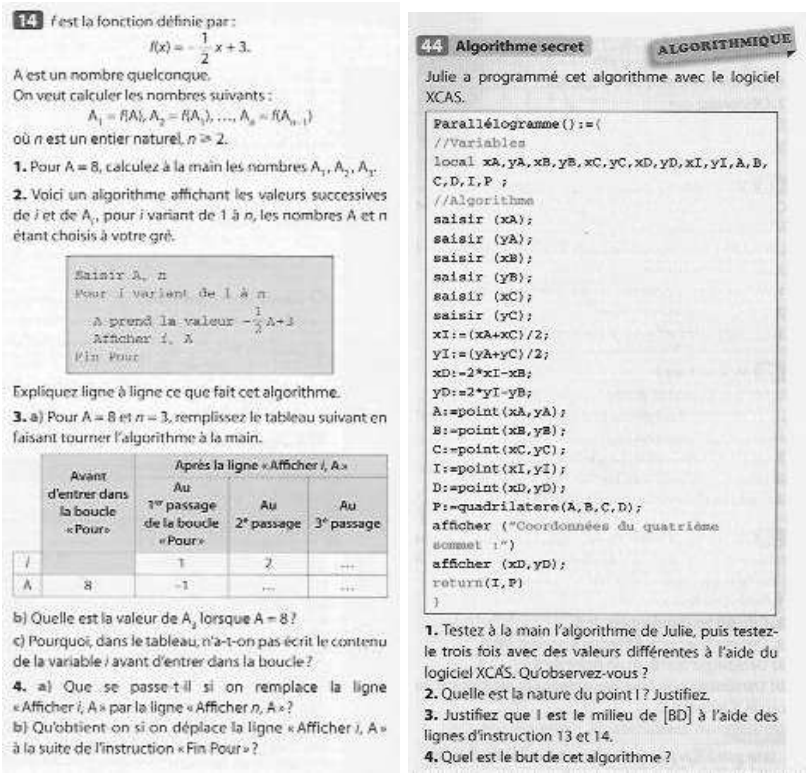
Les questions posées dans les exercices et activités sont les suivantes :

Transmath	Écrire algo.	Écrire prog.	Algo.→ prog.	Modif.	Compr.	Exéc. algo.	Exéc. prog.	Prog.→ algo.
Seconde	17 19%	- -	2 2%	14 16%	30 34%	17 19%	8 9%	1 1%
1e ES-L	7 8%	-	15 16%	9 10%	30 32%	13 14%	19 20%	- -
Te ES-L	11 11%	- -	15 16%	7 7%	35 36%	9 9%	19 20%	- -
1e S	21 24%	- -	5 6%	7 8%	35 40%	9 10%	9 10%	1 1%
Te S	6 7%	1 1%	14 16%	21 24%	25 28%	- -	21 24%	- -

Ce sont les mêmes types de questions que dans la collection Indice que l'on retrouve et l'analyse est essentiellement la même :

- Les questions ne mettent pas en jeu l'algorithme qu'en tant qu'outil : validité et complexité ne sont pas questionnées.
- Les questions relèvent principalement des opérateurs et des systèmes de représentation.
- La relation du problème à l'algorithme est souvent très simple et l'algorithme (PROGRAMME-PAPIER) joue souvent le rôle d'étape intermédiaire entre le problème et le programme.
- Les questions liées à la compréhension du langage et de ses instructions sont très présentes.

Exemples de questions de compréhension du langage :



14 f est la fonction définie par : $f(x) = -\frac{1}{2}x + 3$.

A est un nombre quelconque.
On veut calculer les nombres suivants :
$A_1 = f(A)$, $A_2 = f(A_1)$, ..., $A_n = f(A_{n-1})$
où n est un entier naturel, $n \geq 2$.

1. Pour A = 8, calculez à la main les nombres A_1, A_2, A_3.

2. Voici un algorithme affichant les valeurs successives de i et de A_i, pour i variant de 1 à n, les nombres A et n étant choisis à votre gré.

```
Saisir A, n
Pour i variant de 1 à n
  A prend la valeur -1/2 A+3
  Afficher i, A
Fin Pour
```

Expliquez ligne à ligne ce que fait cet algorithme.

3. a) Pour A = 8 et n = 3, remplissez le tableau suivant en faisant tourner l'algorithme à la main.

	Avant d'entrer dans la boucle « Pour »	Après la ligne « Afficher i, A »		
		Au 1er passage de la boucle « Pour »	Au 2e passage	Au 3e passage
i		1	2	...
A	8	-1	...	...

b) Quelle est la valeur de A_3 lorsque A = 8 ?
c) Pourquoi, dans le tableau, n'a-t-on pas écrit le contenu de la variable i avant d'entrer dans la boucle ?

4. a) Que se passe-t-il si on remplace la ligne « Afficher i, A » par la ligne « Afficher n, A » ?
b) Qu'obtient-on si on déplace la ligne « Afficher i, A » à la suite de l'instruction « Fin Pour » ?

44 Algorithme secret — ALGORITHMIQUE

Julie a programmé cet algorithme avec le logiciel XCAS.

```
Parallélogramme():=(
//Variables
local xA,yA,xB,yB,xC,yC,xD,yD,xI,yI,A,B,
C,D,I,P ;
//Algorithme
saisir (xA);
saisir (yA);
saisir (xB);
saisir (yB);
saisir (xC);
saisir (yC);
xI:=(xA+xC)/2;
yI:=(yA+yC)/2;
xD:=2*xI-xB;
yD:=2*yI-yB;
A:=point(xA,yA);
B:=point(xB,yB);
C:=point(xC,yC);
I:=point(xI,yI);
D:=point(xD,yD);
P:=quadrilatere(A,B,C,D);
afficher ("Coordonnées du quatrième
sommet :")
afficher (xD,yD);
return(I,P)
)
```

1. Testez à la main l'algorithme de Julie, puis testez-le trois fois avec des valeurs différentes à l'aide du logiciel XCAS. Qu'observez-vous ?
2. Quelle est la nature du point I ? Justifiez.
3. Justifiez que I est le milieu de [BD] à l'aide des lignes d'instruction 13 et 14.
4. Quel est le but de cet algorithme ?

Transmath 1e ES-L et seconde : Des questions de compréhension et une justification mathématique de formules.

Algorithmique

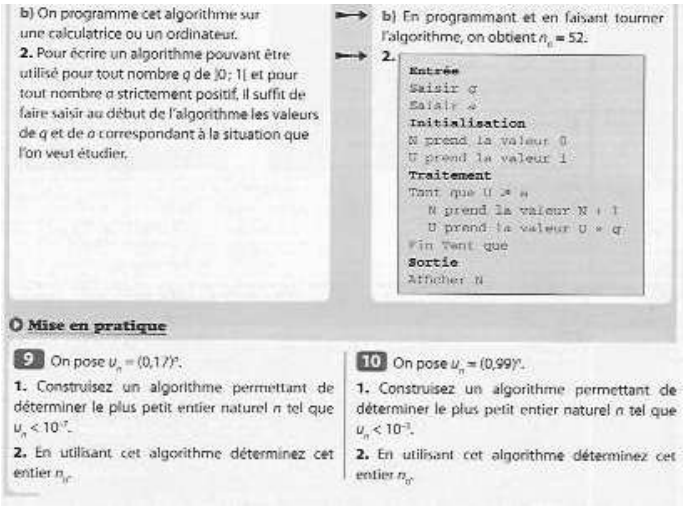
Tout comme dans la collection *Indice*, un chapitre d'algorithmique est proposé dans chaque manuel. Ce chapitre est lui aussi découpé suivant chaque type d'instruction. Les exercices sont ensuite repartis selon le type d'instruction qu'ils mettent en jeu. L'algorithmique est, ici aussi, organisée autour des instructions issues de la programmation.

Des exercices sont aussi proposés, parmi lesquels on retrouve les quelques algorithmes "riches" mais dont le potentiel n'est pas exploité, qui ne font pas partie des instructions officielles et que nous avons pointés plus haut.

Aspects et conceptions

Les aspects présents sont ceux du côté outil. L'aspect EFFECTIVITÉ domine. L'aspect PROBLÈME est présent mais la relation algorithme-problème est vague : on rencontre des ALGORITHMES INSTANCIÉS et des PROGRAMMES DE MODÉLISATION-SIMULATION. En particulier, les ALGORITHMES INSTANCIÉS soulèvent la même question qui se posait dans la collection *Indice* : pourquoi modifier les paramètres dans un algorithme alors que l'on peut intégrer ces paramètres dans l'entrée ?

L'exemple ci-dessous illustre bien ce paradoxe : le cours propose un algorithme général et les exercices demandent l'algorithme avec certains paramètres fixés.


b) On programme cet algorithme sur une calculatrice ou un ordinateur.
2. Pour écrire un algorithme pouvant être utilisé pour tout nombre q de $]0;1[$ et pour tout nombre a strictement positif, il suffit de faire saisir au début de l'algorithme les valeurs de q et de a correspondant à la situation que l'on veut étudier.

b) En programmant et en faisant tourner l'algorithme, on obtient $n_0 = 52$.
2.
```
Entrée
Saisir q
Saisir a
Initialisation
N prend la valeur 0
U prend la valeur 1
Traitement
Tant que U ≥ a
  N prend la valeur N + 1
  U prend la valeur U × q
Fin Tant que
Sortie
Afficher N
```

Mise en pratique

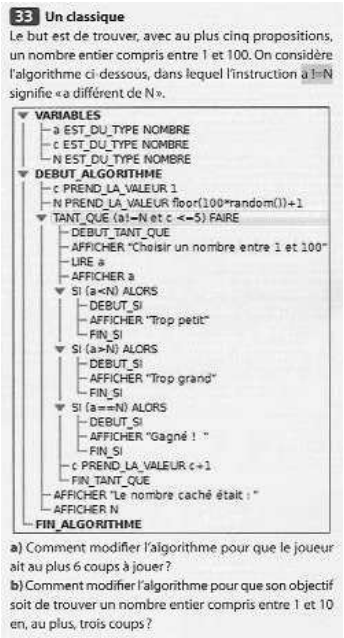
9 On pose $u_n = (0{,}17)^n$.
1. Construisez un algorithme permettant de déterminer le plus petit entier naturel n tel que $u_n < 10^{-7}$.
2. En utilisant cet algorithme déterminez cet entier n_0.

10 On pose $u_n = (0{,}99)^n$.
1. Construisez un algorithme permettant de déterminer le plus petit entier naturel n tel que $u_n < 10^{-3}$.
2. En utilisant cet algorithme déterminez cet entier n_0.

Transmath T^e ES-L : Un algorithme générique et appliqué pour des algorithmes instanciés.

On peut aussi voir cette question du rapport algorithme-problème sur l'exemple, ci-après, de dichotomie en 1^e S : on ne sait pas quel est l'algorithme en jeu, la simulation d'un adversaire ou la stratégie du joueur.

L'aspect objet est totalement absent des manuels *Transmath*. On perçoit quelques remarques concernant le comportement des algorithmes ou leur comparaison, comme ces remarques comparant la recherche de zéro par balayage et par dichotomie et qui ne posent que la question de la précision de l'encadrement (voir extrait du Transmaths T^e ES-L ci-après).
Cela montre bien que les problématiques sur l'algorithme objet sont absentes. Les problèmes en jeu sont donc soit des problèmes de $\mathscr{P}_A$ (majoritairement pauvres), soit des problèmes consistant à rendre effectives certaines procédures en les programmant.



33 **Un classique**

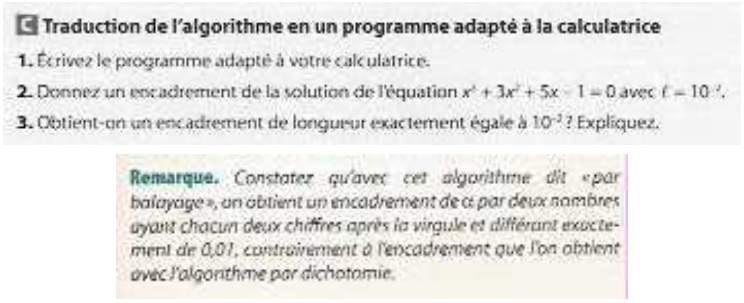
Le but est de trouver, avec au plus cinq propositions, un nombre entier compris entre 1 et 100. On considère l'algorithme ci-dessous, dans lequel l'instruction a !=N signifie « a différent de N ».

```
VARIABLES
    a EST_DU_TYPE NOMBRE
    c EST_DU_TYPE NOMBRE
    N EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    c PREND_LA_VALEUR 1
    N PREND_LA_VALEUR floor(100*random())+1
    TANT_QUE (a!=N et c <=5) FAIRE
        DEBUT_TANT_QUE
        AFFICHER "Choisir un nombre entre 1 et 100"
        LIRE a
        AFFICHER a
        SI (a<N) ALORS
            DEBUT_SI
            AFFICHER "Trop petit"
            FIN_SI
        SI (a>N) ALORS
            DEBUT_SI
            AFFICHER "Trop grand"
            FIN_SI
        SI (a==N) ALORS
            DEBUT_SI
            AFFICHER "Gagné ! "
            FIN_SI
        c PREND_LA_VALEUR c+1
        FIN_TANT_QUE
    AFFICHER "Le nombre caché était : "
    AFFICHER N
FIN_ALGORITHME
```

a) Comment modifier l'algorithme pour que le joueur ait au plus 6 coups à jouer ?

b) Comment modifier l'algorithme pour que son objectif soit de trouver un nombre entier compris entre 1 et 10 en, au plus, trois coups ?

Transmath 1e S : Simulation d'un jeu ou dichotomie ?



Traduction de l'algorithme en un programme adapté à la calculatrice

1. Écrivez le programme adapté à votre calculatrice.

2. Donnez un encadrement de la solution de l'équation $x^3 + 3x^2 + 5x - 1 = 0$ avec $\ell = 10^{-2}$.

3. Obtient-on un encadrement de longueur exactement égale à 10^{-2} ? Expliquez.

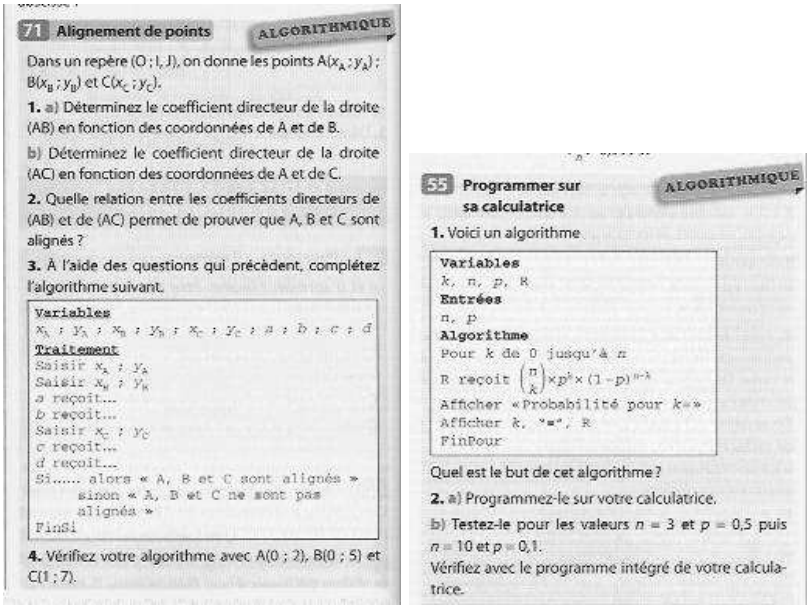
Remarque. *Constatez qu'avec cet algorithme dit « par balayage », on obtient un encadrement de ci par deux nombres ayant chacun deux chiffres après la virgule et différant exactement de 0,01, contrairement à l'encadrement que l'on obtient avec l'algorithme par dichotomie.*

Transmath Te ES-L : Question sur la dichotomie et remarque sur le balayage.

Les opérateurs sont, comme pour la collection Indice, les instructions du langage de programmation et du langage des PROGRAMMES-PAPIER.
Les seules structures de contrôle présentes sont les vérifications par des tests sur des données. Deux exemples en sont donnés page suivante.

Bilan

- Les aspects outil dominent, notamment l'EFFECTIVITÉ. La majorité des algorithmes proposés ne permettraient pas de mettre en jeu d'autres aspects.
- L'aspect PROBLÈME n'est que très partiellement mis en jeu et les algorithmes proposés sont des ALGORITHMES INSTANCIÉS, des algorithmes du cours dont on fixe une partie des paramètres ou encore des PROGRAMMES DE MODÉLISATION-SIMULATION.
- Le langage d'expression des algorithmes est très contraint par celui de la programmation. Cela produit des PROGRAMMES-PAPIER qui sont caractéristiques d'une focalisation sur les structures de représentation de AI. L'organisation du chapitre d'algorithmique autour des instructions usuelles des langages de programmation confirme cet ancrage.
- Les structures de contrôle relèvent de la validité mathématique des formules implémentées ou du test des programmes sur des exemples. On est bien dans une conception AI-outil.



71 Alignement de points ALGORITHMIQUE

Dans un repère (O ; I, J), on donne les points $A(x_A ; y_A)$; $B(x_B ; y_B)$ et $C(x_C ; y_C)$.

1. a) Déterminez le coefficient directeur de la droite (AB) en fonction des coordonnées de A et de B.

b) Déterminez le coefficient directeur de la droite (AC) en fonction des coordonnées de A et de C.

2. Quelle relation entre les coefficients directeurs de (AB) et de (AC) permet de prouver que A, B et C sont alignés ?

3. À l'aide des questions qui précèdent, complétez l'algorithme suivant.

```
Variables
x_A ; y_A ; x_B ; y_B ; x_C ; y_C ; a ; b ; c ; d
Traitement
Saisir x_A ; y_A
Saisir x_B ; y_B
a reçoit...
b reçoit...
Saisir x_C ; y_C
c reçoit...
d reçoit...
Si..... alors « A, B et C sont alignés »
     sinon « A, B et C ne sont pas
     alignés »
FinSi
```

4. Vérifiez votre algorithme avec A(0 ; 2), B(0 ; 5) et C(1 ; 7).

55 Programmer sur sa calculatrice ALGORITHMIQUE

1. Voici un algorithme

```
Variables
k, n, p, R
Entrées
n, p
Algorithme
Pour k de 0 jusqu'à n
R reçoit (n k) × p^k × (1 - p)^(n-k)
Afficher « Probabilité pour k= »
Afficher k, "=", R
FinPour
```

Quel est le but de cet algorithme ?

2. a) Programmez-le sur votre calculatrice.

b) Testez-le pour les valeurs $n = 3$ et $p = 0{,}5$ puis $n = 10$ et $p = 0{,}1$.

Vérifiez avec le programme intégré de votre calculatrice.

Transmath seconde et 1e S : Validation par tests ou comparaison avec les programmes intégrés de la calculatrice.

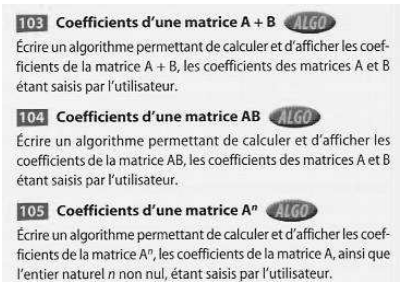
2.3 Enseignements de spécialité (Indice)

Spécialité de Te ES

Les algorithmes présents en terminale ES sont peu nombreux malgré les thèmes abordés. Comme les questions sont peu diverses, nous résumons l'ensemble en un seul tableau :

Chapitre	Algorithmes	Questions
1. Problèmes sur les matrices	- Opérations usuelles sur les matrices	- Écrire un algorithme qui... (3)
2. Problèmes de graphes	- Recherche d'une chaîne eulérienne	- Appliquer l'algorithme (3)
3. Graphes pondérés et probabilistes	- Recherche d'un plus court chemin (Dijkstra)	-Appliquer l'algorithme (14) - Interpréter sa solution (3)

Les questions du chapitre 1 correspondent aux trois exercices suivants :



103 Coefficients d'une matrice A + B ALGO

Écrire un algorithme permettant de calculer et d'afficher les coefficients de la matrice A + B, les coefficients des matrices A et B étant saisis par l'utilisateur.

104 Coefficients d'une matrice AB ALGO

Écrire un algorithme permettant de calculer et d'afficher les coefficients de la matrice AB, les coefficients des matrices A et B étant saisis par l'utilisateur.

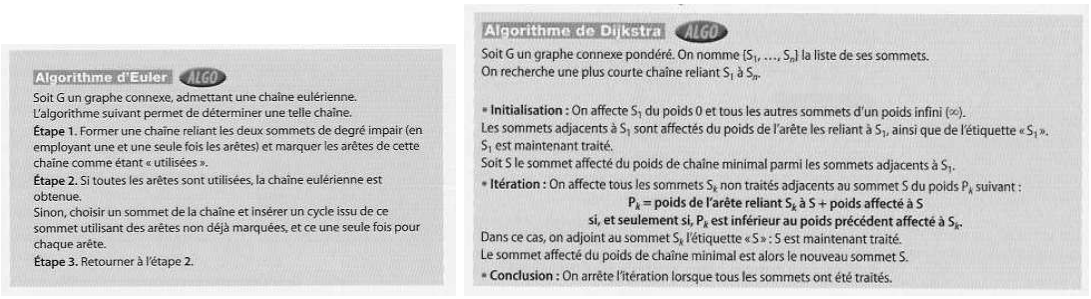
105 Coefficients d'une matrice A^n ALGO

Écrire un algorithme permettant de calculer et d'afficher les coefficients de la matrice A^n, les coefficients de la matrice A, ainsi que l'entier naturel n non nul, étant saisis par l'utilisateur.

Indice Te ES (spé.) : Algorithmes opératoires sur les matrices.

Il s'agit simplement ici de faire formaliser les opérations demandées. Notons que la demande de saisie et d'affichage fait partie du problème. On est dans AI, dans la continuité des manuels étudiés plus haut.

Concernant les chapitres 2 et 3, les algorithmes prennent une forme moins formelle. En effet, il s'agit de faire appliquer par l'élève les algorithmes du cours. Ils sont donc formulés en langage courant :



Algorithme d'Euler ALGO

Soit G un graphe connexe, admettant une chaîne eulérienne.
L'algorithme suivant permet de déterminer une telle chaîne.

Étape 1. Former une chaîne reliant les deux sommets de degré impair (en employant une et une seule fois les arêtes) et marquer les arêtes de cette chaîne comme étant « utilisées ».

Étape 2. Si toutes les arêtes sont utilisées, la chaîne eulérienne est obtenue.
Sinon, choisir un sommet de la chaîne et insérer un cycle issu de ce sommet utilisant des arêtes non déjà marquées, et ce une seule fois pour chaque arête.

Étape 3. Retourner à l'étape 2.

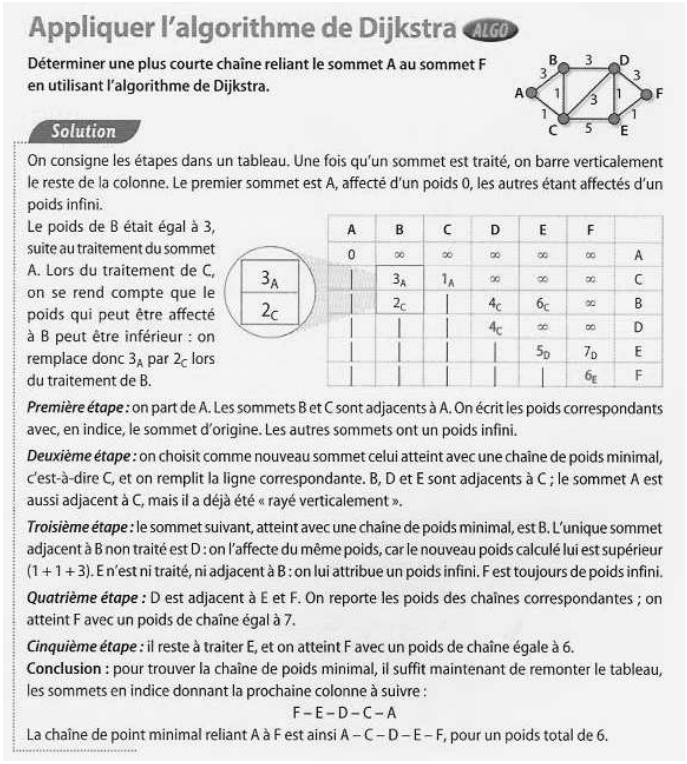
Algorithme de Dijkstra ALGO

Soit G un graphe connexe pondéré. On nomme $\{S_1, \ldots, S_n\}$ la liste de ses sommets.
On recherche une plus courte chaîne reliant S_1 à S_n.

- **Initialisation :** On affecte S_1 du poids 0 et tous les autres sommets d'un poids infini (∞).
Les sommets adjacents à S_1 sont affectés du poids de l'arête les reliant à S_1, ainsi que de l'étiquette « S_1 ».
S_1 est maintenant traité.
Soit S le sommet affecté du poids de chaîne minimal parmi les sommets adjacents à S_1.
- **Itération :** On affecte tous les sommets S_k non traités adjacents au sommet S du poids P_k suivant :
P_k = poids de l'arête reliant S_k à S + poids affecté à S
si, et seulement si, P_k est inférieur au poids précédent affecté à S_k.
Dans ce cas, on adjoint au sommet S_k l'étiquette « S » : S est maintenant traité.
Le sommet affecté du poids de chaîne minimal est alors le nouveau sommet S.
- **Conclusion :** On arrête l'itération lorsque tous les sommets ont été traités.

Indice T^e ES (spé.) : Algorithmes en langage courant sur les graphes.

Ces deux algorithmes sont ensuite spécifiés pour une exécution à la main en remplissant un tableau :



Appliquer l'algorithme de Dijkstra ALGO

Déterminer une plus courte chaîne reliant le sommet A au sommet F en utilisant l'algorithme de Dijkstra.

Solution

On consigne les étapes dans un tableau. Une fois qu'un sommet est traité, on barre verticalement le reste de la colonne. Le premier sommet est A, affecté d'un poids 0, les autres étant affectés d'un poids infini.

Le poids de B était égal à 3, suite au traitement du sommet A. Lors du traitement de C, on se rend compte que le poids qui peut être affecté à B peut être inférieur : on remplace donc 3_A par 2_C lors du traitement de B.

A	B	C	D	E	F	
0	∞	∞	∞	∞	∞	A
|	3_A	1_A	∞	∞	∞	C
|	2_C	|	4_C	6_C	∞	B
|	|	|	4_C	∞	∞	D
|	|	|	|	5_D	7_D	E
|	|	|	|	|	6_E	F

Première étape : on part de A. Les sommets B et C sont adjacents à A. On écrit les poids correspondants avec, en indice, le sommet d'origine. Les autres sommets ont un poids infini.

Deuxième étape : on choisit comme nouveau sommet celui atteint avec une chaîne de poids minimal, c'est-à-dire C, et on remplit la ligne correspondante. B, D et E sont adjacents à C ; le sommet A est aussi adjacent à C, mais il a déjà été « rayé verticalement ».

Troisième étape : le sommet suivant, atteint avec une chaîne de poids minimal, est B. L'unique sommet adjacent à B non traité est D : on l'affecte du même poids, car le nouveau poids calculé lui est supérieur (1 + 1 + 3). E n'est ni traité, ni adjacent à B : on lui attribue un poids infini. F est toujours de poids infini.

Quatrième étape : D est adjacent à E et F. On reporte les poids des chaînes correspondantes ; on atteint F avec un poids de chaîne égal à 7.

Cinquième étape : il reste à traiter E, et on atteint F avec un poids de chaîne égale à 6.

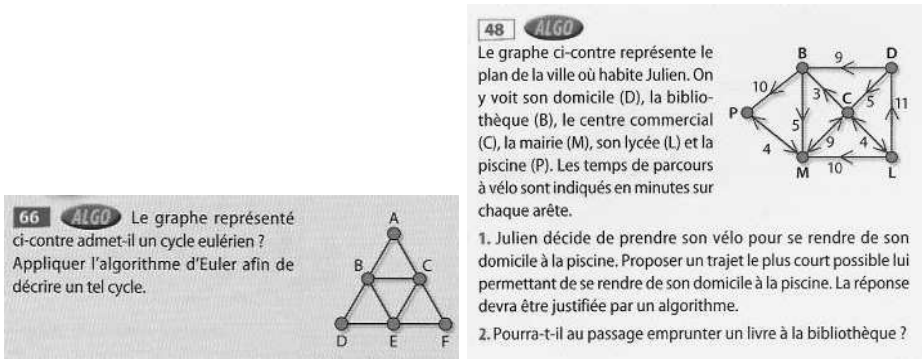
Conclusion : pour trouver la chaîne de poids minimal, il suffit maintenant de remonter le tableau, les sommets en indice donnant la prochaine colonne à suivre :

F – E – D – C – A

La chaîne de point minimal reliant A à F est ainsi A – C – D – E – F, pour un poids total de 6.

Indice T^e ES (spé.) : Algorithme de Dijkstra, version pour exécution à la main.

Les exercices qui suivent sont tous des applications de ces deux algorithmes [10]. Parfois, on demande d'interpréter la solution obtenue :



66 ALGO Le graphe représenté ci-contre admet-il un cycle eulérien ? Appliquer l'algorithme d'Euler afin de décrire un tel cycle.

48 ALGO Le graphe ci-contre représente le plan de la ville où habite Julien. On y voit son domicile (D), la bibliothèque (B), le centre commercial (C), la mairie (M), son lycée (L) et la piscine (P). Les temps de parcours à vélo sont indiqués en minutes sur chaque arête.

1. Julien décide de prendre son vélo pour se rendre de son domicile à la piscine. Proposer un trajet le plus court possible lui permettant de se rendre de son domicile à la piscine. La réponse devra être justifiée par un algorithme.

2. Pourra-t-il au passage emprunter un livre à la bibliothèque ?

Indice T^e ES (spé.) : Exercices d'application des algorithmes du cours.

10. La situation étant modélisée à l'avance par le graphe concerné. Voir (Cartier, 2008) pour une étude de l'enseignement de la théorie des graphes, notamment en spécialité de T^e ES.

Notons que dans l'exercice de recherche de plus courts chemins, c'est l'algorithme qui justifie l'optimalité du chemin obtenu. La plupart des exercices d'utilisation de l'algorithme de Dijkstra spécifie que l'*on veillera à justifier la réponse par un algorithme*. Dans ce cas le rôle de l'algorithme est de fournir une solution et d'être une garantie (sans assurance que l'algorithme a bien été appliqué).

Aucun des algorithmes présents n'est prouvé ou validé. Leur justification s'appuie sur des exemples introduisant l'"idée" de l'algorithme sur un cas particulier.

On retrouve donc uniquement les aspects outils de l'algorithme.
Les conceptions AI et AM cohabitent, mais on peut supposer que AM est présent car la programmation d'algorithmes sur les graphes demande des structures de données complexes.

Spécialité de T^e S - Indice

Nous présentons l'ensemble des algorithmes en jeu dans les quatre chapitres du manuel :

- Ch. 1 : Divisibilité et nombres premiers.
  - Liste des diviseurs d'un entiers
  - Test de primalité
  - Décomposition en facteurs premiers
  - Nombre de diviseurs
  - Représentant de A modulo n (+4 fois avec n fixé)
  - Test de divisibilité
  - Nombre de zéros à la fin d'un entier (2 versions)
  - Écriture en base b
  - Puissance de p dans la décomposition en facteurs premiers de N (+ 2 fois avec p fixé)
  - Reste de A^N dans la division par M
  - Formules (5), formules conditionnelles (2) et autres (2)
- Ch. 2 : Problèmes de chiffrement
  - Algorithme des différences
  - Algorithme d'Euclide (3)
  - Algorithme d'Euclide étendu
  - Recherche des triplets pythagoriciens
  - Résolution de $ax \equiv 1[b]$.
- Ch. 3 : Problèmes sur les matrices
  - Calculer les termes d'un système de deux suites récurrentes (3)
  - Calculer les termes d'une suite récurrente d'ordre 2 (3)
  - Test d'inversibilité d'une matrice 2×2
- Ch. 4 : Problèmes d'évolution
  - Simulation
  - Calculer les termes d'une suite récurrente d'ordre 2 (2)
  - Formule

On trouve ici un ensemble d'algorithmes de $\mathscr{P}_{\text{A}}$, qui semble avoir un intérêt pour que l'algorithme soit objet. Il s'agit des algorithmes des chapitres 1 et 2 : ceux mettant en jeu l'arithmétique. Cela va dans le sens de nos hypothèses sur les disciplines privilégiées pour l'algorithme-objet.

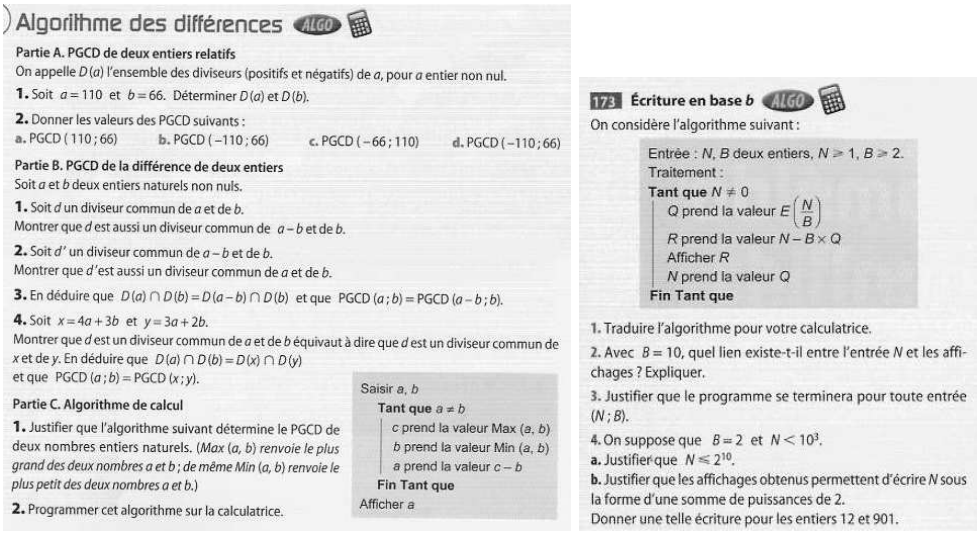
Les questions proposées sont présentées dans le tableau ci-après. La majorité des questions traite de l'outil, mais l'algorithme est aussi questionné en tant qu'objet. On constate que cela se situe, là aussi, dans le champ de l'arithmétique.

	Écrire un algo.	Écrire un prog.	algo → prog.	Exéc. algo.	Exéc. prog.	Compléter	Modifier	Compréhension	Justifier un algo.	Justif. une prop. de l'algo.	Terminaison
Ch. 1 : Divisibilité et nombre premiers	2	5	2	8	9	2	6	6	4	5	2
Ch. 2 : Problèmes de chiffrement	1	-	5	6	-	-	0	1	3	-	-
Ch. 3 : Problèmes sur les matrices	1	3	2	-	1	1	0	1	-	-	-
Ch. 4 : Problèmes d'évolution	2	-	2	-	1	2	1	0	-	-	-
Total	4 5%	8 10%	11 13%	14 17%	11 13%	5 6%	7 9%	8 10%	7 9%	5 6%	2 2%

Plus précisément, on retrouve trois types de questions :

- les questions relatives à la correction de l'algorithme, où un travail mathématique, souvent guidé, amène à valider la réponse fournie pour toute entrée,
- les questions qui concernent les propriétés de l'algorithme, parfois pour permettre de répondre aux précédentes,
- les questions relatives à la terminaison des algorithmes (qui s'appuient sur la recherche d'un invariant simple).

Les aspects mis en avant par ces questions sont EFFECTIVITÉ et PROBLÈME mais aussi l'aspect PREUVE.



Algorithme des différences ALGO

Partie A. PGCD de deux entiers relatifs

On appelle $D(a)$ l'ensemble des diviseurs (positifs et négatifs) de a, pour a entier non nul.

1. Soit $a = 110$ et $b = 66$. Déterminer $D(a)$ et $D(b)$.

2. Donner les valeurs des PGCD suivants :
a. PGCD (110 ; 66) **b.** PGCD (−110 ; 66) **c.** PGCD (−66 ; 110) **d.** PGCD (−110 ; 66)

Partie B. PGCD de la différence de deux entiers

Soit a et b deux entiers naturels non nuls.

1. Soit d un diviseur commun de a et de b.
Montrer que d est aussi un diviseur commun de $a - b$ et de b.

2. Soit d' un diviseur commun de $a - b$ et de b.
Montrer que d' est aussi un diviseur commun de a et de b.

3. En déduire que $D(a) \cap D(b) = D(a-b) \cap D(b)$ et que PGCD $(a ; b)$ = PGCD $(a - b ; b)$.

4. Soit $x = 4a + 3b$ et $y = 3a + 2b$.
Montrer que d est un diviseur commun de a et de b équivaut à dire que d est un diviseur commun de x et de y. En déduire que $D(a) \cap D(b) = D(x) \cap D(y)$
et que PGCD $(a ; b)$ = PGCD $(x ; y)$.

Partie C. Algorithme de calcul

1. Justifier que l'algorithme suivant détermine le PGCD de deux nombres entiers naturels. (*Max (a, b) renvoie le plus grand des deux nombres a et b ; de même Min (a, b) renvoie le plus petit des deux nombres a et b.*)

2. Programmer cet algorithme sur la calculatrice.

```
Saisir a, b
Tant que a ≠ b
    c prend la valeur Max (a, b)
    b prend la valeur Min (a, b)
    a prend la valeur c − b
Fin Tant que
Afficher a
```

173 Écriture en base b ALGO

On considère l'algorithme suivant :

```
Entrée : N, B deux entiers, N ⩾ 1, B ⩾ 2.
Traitement :
Tant que N ≠ 0
    Q prend la valeur E(N/B)
    R prend la valeur N − B × Q
    Afficher R
    N prend la valeur Q
Fin Tant que
```

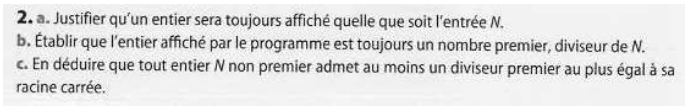
1. Traduire l'algorithme pour votre calculatrice.

2. Avec $B = 10$, quel lien existe-t-il entre l'entrée N et les affichages ? Expliquer.

3. Justifier que le programme se terminera pour toute entrée $(N ; B)$.

4. On suppose que $B = 2$ et $N < 10^3$.
a. Justifier que $N \leqslant 2^{10}$.
b. Justifier que les affichages obtenus permettent d'écrire N sous la forme d'une somme de puissances de 2.
Donner une telle écriture pour les entiers 12 et 901.

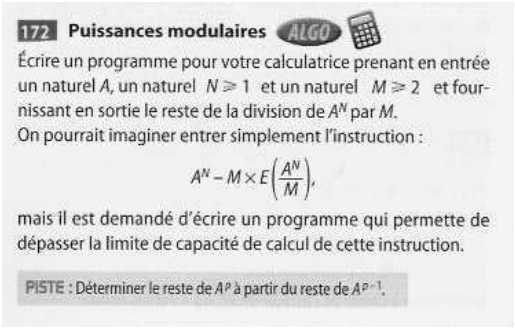
Indice T^e S (spé.) : Questions de correction et de terminaison.



2. a. Justifier qu'un entier sera toujours affiché quelle que soit l'entrée N.
b. Établir que l'entier affiché par le programme est toujours un nombre premier, diviseur de N.
c. En déduire que tout entier N non premier admet au moins un diviseur premier au plus égal à sa racine carrée.

Indice T^e S (spé.) : Étude de propriétés d'une sortie (à propos de la recherche d'un diviseur premier).

L'aspect COMPLEXITÉ est absent. Des questions pourraient surement apparaître lorsque l'on propose deux algorithmes pour un même problème. La seule référence à la complexité est celle-ci :



172 Puissances modulaires ALGO

Écrire un programme pour votre calculatrice prenant en entrée un naturel A, un naturel $N \geqslant 1$ et un naturel $M \geqslant 2$ et fournissant en sortie le reste de la division de A^N par M.
On pourrait imaginer entrer simplement l'instruction :

$$A^N - M \times E\left(\frac{A^N}{M}\right),$$

mais il est demandé d'écrire un programme qui permette de dépasser la limite de capacité de calcul de cette instruction.

PISTE : Déterminer le reste de A^p à partir du reste de A^{p-1}.

Indice T^e S (spé.) : Recherche d'un algorithme "efficace".

La forme des algorithmes suit celle des manuels *Indice* que nous avons décrite, programmes et algorithmes. Mais le langage pour les algorithmes est plus souple, il n'y a pas de déclaration de variable. Quelques algorithmes sont aussi exprimés en langage courant (l'algorithme d'Euclide par exemple, ou celui d'Euclide étendu). On peut penser que cela est lié au fait que ces algorithmes peuvent être exécutés à la main (il est d'ailleurs proposé de présenter leur exécution dans un tableau).

Avec Xcas (dans le TP1 sur le nombre de zéros terminant $n!$, par exemple), les algorithmes sont programmés sous forme de fonctions. On sort du cadre qui impose saisie pour l'entrée et affichage pour la sortie (même si cela est très présent dans l'ensemble des algorithmes).

On peut considérer que les deux paradigmes AI et AM sont présents dans la conception de l'algorithme en spécialité de T^e S. Les opérateurs ne sont pas uniquement ceux du langage de programmation et leur adaptation sur papier : on retrouve des algorithmes à manipuler à la main qui sont exprimés de manières différentes.

Les structures de contrôle de l'algorithme sont de l'ordre de la preuve (de correction et quelquefois de terminaison). La conception-outil a donc une structure de contrôle forte. L'algorithme-objet est en jeu et fait même l'objet de questions.

Conclusion

Enseignement obligatoire

L'analyse des deux collections de manuels (hors enseignements de spécialité) fait ressortir plusieurs points :

- L'algorithme a un rôle d'outil de résolution, mais surtout de mise en œuvre de méthodes (domination de l'EFFECTIVITÉ). Un grand nombre de questions concerne l'écriture d'algorithmes, leur implémentation puis leur exécution. Cela va dans le sens des hypothèses 1 et 2.
- La relation algorithme-problème est vague et l'on rencontre régulièrement des ALGORITHMES-INSTANCIÉS ou des algorithmes dont une partie des entrées a été instanciée. Cela correspond à l'hypothèse 1.

- Les aspects PREUVE et COMPLEXITÉ sont totalement absents comme attendu dans l'hypothèse 1.
- Le paradigme dominant est AI. Les algorithmes sont soit des programmes, soit exprimés dans un langage extrêmement proche et adapté à la traduction immédiate en programmes. C'est l'hypothèse 3.
- L'algorithmique est majoritairement restreinte à une activité langagière. Beaucoup de questions portent sur l'interprétation d'un langage, son écriture et sa traduction. On retrouve à nouveau l'hypothèse 2.
- Les seules structures de contrôle sont d'ordre mathématique lorsqu'il s'agit d'écrire, sous la structure formelle d'un algorithme, une technique ou une méthode du cours ou bien des tests d'exécution lorsque l'on écrit un programme. Cela confirme l'hypothèse 4.

En ajoutant au bilan des manuels de l'enseignement obligatoire celui des spécialités mathématiques, on peut ajouter :

- À l'instar des programmes, les manuels proposent des "algorithmes" spécifiques à chaque domaine : calcul de termes de suites dans les chapitres sur les suites, PROGRAMMES DE MODÉLISATION-SIMULATION dans les chapitres de probabilité et statistique, méthodes numériques en analyse, etc. Certains chapitres ne proposent aucun algorithme. D'autre part les algorithmes plus riches se trouvent dans certains domaines bien précis : l'arithmétique, la théorie des graphes, les méthodes numériques et quelques fois dans les chapitres d'algorithmique qui permettent de sortir des domaines mathématiques du programme. Cela confirme notre hypothèse 5.

Ces résultats valident nos hypothèses et renforcent les conclusions des chapitres précédents concernant la transposition au lycée.

Enseignements de spécialité

T^e ES Le manuel de spécialité de T^e ES étudié met en avant le paradigme AM.
Les algorithmes présents sont donnés dans le cours pour être appliqués à la main dans les exercices. On ne rencontre donc quasiment aucun exercice de production d'algorithme, il s'agit principalement d'appliquer ces algorithmes. Par rapport aux manuels de l'enseignement obligatoire, il semble que ce n'est pas l'écriture d'algorithmes et de programmes qui est outil mais seulement les algorithmes donnés par le cours.
Nous faisons l'hypothèse que le paradigme AM est seulement présent car le paradigme AI n'est pas adapté à la manipulation des graphes.

T^e S La spécialité de terminale S présente une conception très différente de l'algorithme. L'algorithme est outil, de la même façon que dans les manuels de l'enseignement obligatoire, mais il est aussi objet : on rencontre des questions de preuve d'algorithmes (validité des solutions produites et terminaison) qui s'appuient sur des raisonnements mathématiques (souvent guidés).
Nous pensons que la présence de l'arithmétique est un facteur important dans la présence de l'algorithme comme objet.
Beaucoup d'algorithmes proposés (en arithmétique) sont des algorithmes riches. Cependant, coexistent des traductions de formules, et des PROGRAMMES DE MODÉLISATION-SIMULATION considérés aussi comme des algorithmes.
Bien que le paradigme AI domine fortement, on rencontre parfois le paradigme AM (mais peut-être pour les mêmes raisons qu'en spécialité de T^e ES).

Analyse systématique

Les modèles de conceptions et d'aspects que nous avons proposés s'adaptent bien à une analyse quantitative des algorithmes en jeu et des types de questions posées.
Ils nous ont permis d'analyser les types d'activités proposés dans les manuels et d'en déduire la conception de l'algorithmique présente.
Le modèle par conception rend possible une étude du rapport à l'algorithme de manière globale, sans entrer dans le détail des techniques de résolution attendues.

Une systématisation plus poussée de la méthodologie d'analyse utilisée est envisageable, en produisant par exemple un instrument d'analyse facilement transmissible. Un tel outil comprendrait à la fois une grille de questions et les réponses possibles avec leur interprétation. Développer une tel instrument demande des analyses de ressources supplémentaires, notamment de documents mettant plus en jeu l'aspect objet, afin de fournir un outil complet.

Conclusions sur la transposition au lycée

À travers l'étude des instructions officielles, de ressources en ligne et de manuels, nous avons pu porter un regard sur les phénomènes de transposition didactique en jeu concernant l'algorithmique au lycée. Notre analyse s'est appuyée sur les modèles épistémologiques développés dans les chapitres précédents.
Nous allons revenir maintenant sur les résultats de cette étude, et sur les outils que nous avons utilisés.

Transposition

Instructions officielles de mathématiques

L'étude des programmes et du document d'accompagnement de seconde révèle une grande distance entre le concept algorithme du savoir savant et celui proposé.
En particulier, la notion d'algorithme proposée dans les instructions officielles est un moyen de décrire toute procédure effective des mathématiques. Cela englobe alors beaucoup d'éléments qui ne nous paraissent pas permettre à l'algorithme de vivre en tant qu'objet. En particulier, les notions centrales d'instance et de problème n'apparaissent plus ici comme nécessaires. Nous avons introduit les notions de PROGRAMME DE MODÉLISATION-SIMULATION et d'ALGORITHME INSTANCIÉ pour décrire certains de ces phénomènes.

D'autre part, l'analyse épistémologique montre que les algorithmes vivent sous différentes formes parmi lesquelles l'écriture dans un langage de programmation ne représente qu'une de leurs expressions. Ici, l'algorithme s'appuie essentiellement sur la notion de programme, au point que les algorithmes exprimés hors de ces langages doivent en conserver les caractéristiques. Nous avons proposé le terme de PROGRAMME-PAPIER pour décrire ces algorithmes très influencés par les programmes et leurs interfaces utilisateurs.

En résumé, une seule conception domine les programmes : AI-objet.

Manuels de mathématiques

Les manuels de mathématiques étudiés se situent dans la suite des instructions officielles. L'étude quantitative des algorithmes en jeu et des exercices proposés va dans le sens de l'étude des programmes.

L'algorithme est outil et principalement orienté vers AI. Les algorithmes proposés sont des mises en œuvre, propres à la programmation d'objets mathématiques implémentables qui

ne permettent globalement pas de faire vivre le concept algorithme comme un objet des mathématiques [11].

Ressources en lignes

L'étude des ressources en lignes des IREM, proposée en complément de cette analyse, montre des transpositions diverses.

Les ressources pour la classe proposent globalement les mêmes types d'algorithmes que les programmes et les manuels. La différence peut être que le paradigme AM apparaît dans certaines ressources. Dans tous les cas l'algorithme n'apparaît pas comme objet mais seulement comme outil. En particulier, on ne trouve aucune question de complexité ou de preuve d'algorithmes.

Pour rencontrer l'algorithme objet, il faut se tourner vers certaines ressources pour la formation des enseignants. Nous avons pu constater qu'alors, les domaines mathématiques abordés diffèrent de ceux des programmes du lycée.

ISN

Les programmes de la spécialité *Informatique et Sciences du Numérique* ont été analysés parallèlement à cela. Cela nous a permis de dévoiler une transposition toute autre, dans laquelle l'algorithme est un objet de la discipline à part entière.

En particulier l'algorithme est clairement distingué de la notion de programme est l'importance de la notion d'instance est soulignée. L'aspect outil de l'algorithme est très présent mais l'aspect objet n'est pas laissé de côté, les questions de complexité devant être abordées.

Outils utilisés

Les modèles épistémologiques proposés, Aspects et μ-conceptions, se sont avérés efficaces pour mener cette analyse de la transposition. Nous avons pu répondre à toutes nos questions ($Q5_{\text{aspects}}$ et $Q5_{\text{conceptions}}$, $Q6_{\text{aspects}}$ et $Q6_{\text{conceptions}}$, $Q7_{\text{aspects}}$ et $Q7_{\text{conceptions}}$).

Les outils dérivés de nos modèles nous ont permis l'analyse de documents variés par leurs formes et leurs contenus. Nous avons aussi pu les adapter à l'analyse de la transposition dans le cadre d'un enseignement d'informatique. L'analyse des manuels a montré la possibilité de les utiliser dans un cadre plus quantitatif.

La théorie cK$\not$ (en particulier la recherche des constituants des conceptions) contribue, à travers notre modèle, à la souplesse des outils d'analyse proposés. Le modèle d'aspects complète assez bien cet outil lorsque l'on n'a pas accès à l'activité ou à ses traces (dans les discours généraux sur l'algorithmique par exemple).

11. Soulignons l'exception relevée en spécialité de T^e^ S qui montre aussi l'aspect objet de l'algorithme, dans la partie arithmétique. Cela se produit autour de l'aspect PREUVE, avec des questions de correction et de terminaison d'algorithmes.

Perspectives

Nous donnons ici quelques perspectives de recherche que nos résultats et les questions de transpositions de l'algorithme soulèvent :

- La suite de l'étude de la transposition peut se faire par l'observation en classe du savoir enseigné. Nos modèles devraient pouvoir être adaptés à une telle étude et cela complète l'étude de la "chaîne" de transposition. L'usage des ressources en algorithmique par les enseignants et l'analyse de leurs conceptions peuvent compléter une telle étude.
- L'étude d'autres transpositions du concept, en informatique, en mathématiques (à l'étranger par exemple) ou dans des enseignements mixtes mathématiques-informatique nous semble aussi être une piste à explorer.
- Dans cet objectif et dans le but de produire un instrument d'analyse transmissible, il faut envisager la construction d'un outil systématique d'analyse de ressources en algorithmique. Un tel outil devra prendre en compte l'ensemble des transpositions envisageables et détailler les critères permettant de les repérer.
- L'écologie de l'algorithme nous permettrait de mieux comprendre les conditions et contraintes qui influencent la transposition [12].
- Les constats concernant la transposition peuvent nous guider dans la construction de situations, en orientant le travail sur les aspects et conceptions absents du savoir enseigné.

C'est cette dernière question que nous allons aborder dans la partie suivante.

12. Il nous semble que l'écologie des savoirs peut permettre d'expliquer les difficultés pour l'algorithme de vivre en tant qu'objet dans l'enseignement. Le rôle mathématique qui lui est donné et qu'il peut prendre est à étudier mais son rôle didactique est aussi à prendre en compte. En effet, attribuer un rôle de médiation d'autres concepts à l'algorithme peut être un obstacle à son existence en tant qu'objet mathématique à part entière.

Quatrième partie

Problèmes et situations pour l'algorithme

Introduction :
Un travail en cours

Cette dernière partie s'attache aux questions de conception, de développement et d'analyse de situations didactiques pour l'algorithme et l'algorithmique.

Il s'agit d'un chantier commencé en 2009. Les difficultés liées à la construction de telles situations nous ont amené à développer l'analyse épistémologique et les modèles qui constituent les premiers chapitres de cette thèse. Il nous semble important de souligner que ce travail théorique a pour origine le développement de situations.

D'autre part, ce travail de production de situations est une contrepartie nécessaire à l'analyse de la transposition de l'algorithmique au lycée. Il nous semble important que l'étude du curriculum et des ressources ne soit pas une fin en soi, mais s'accompagne du développement d'outils et de moyens pour les enseignants (au moins à long terme).

Le développement de situations et leur expérimentation permet aussi de requestionner les modèles épistémologiques et de les faire évoluer. C'est aussi un de nos objectifs avec le développement de situations didactiques pour l'algorithme.

Bien que cette partie présente des travaux en cours, elle avance des pistes sérieuses que nous souhaitons détailler. Elle montre aussi en quoi le travail théorique effectué apporte des réponses à la problématique de développement de situations. Cette partie se veut donc le témoin de notre ancrage dans une recherche didactique où la théorie est motivée par le développement d'outils d'enseignement. Elle permet aussi une ouverture sur nos perspectives de travail actuelles.

Chapitre 9

Problèmes fondamentaux - Vers des situations pour l'algorithme

Sommaire

Introduction 207
1 Problèmes fondamentaux 208
1.1 Problèmes et dialectique outil-objet 208
1.2 La notion de problème fondamental 209
2 Quelques propositions de problèmes 211
3 Quels types de situations pour l'algorithme ? 215
3.1 Objectifs 215
3.2 SiRC 216
3.3 Problèmes d'optimisation 216
4 Une proposition de situation 217
4.1 Le problème des pesées 217
4.2 Analyse mathématique du problème 218
4.3 Un problème fondamental 227
4.4 Analyse a priori de la situation 228
Conclusion et perspectives 232

Introduction

La partie précédente s'est intéressée à l'analyse de la transposition du concept algorithme dans l'enseignement de mathématiques au lycée. Les principaux résultats montrent une forte distance entre le concept enseigné et celui du savoir savant. En particulier, l'algorithme est essentiellement outil et vit principalement dans le paradigme AI et dans des activités de programmation.

Nous souhaitons proposer des situations didactiques[1] qui mettent en jeu l'algorithme, à la fois comme outil et comme objet. Étant donnés les liens privilégiés entre algorithme et problème et entre algorithme et preuve, nous pensons que les situations proposées doivent mettre en jeu une activité de preuve ou de recherche dans un contexte de résolution de problème. Pour construire de telles situations, il nous semble important de considérer des problèmes de $\mathscr{P}_{\text{A}}$ qui soient suffisamment riches au niveau épistémologique. Pour cela, les

1. Dans la suite du chapitre nous dirons simplement *situation* pour *situation didactique*

modèles proposés dans la première partie sont un guide indispensable [2]. Ce questionnement a été présenté en deux temps :

Question Q8. *Comment caractériser les problèmes à fort potentiel pour l'algorithmique à l'aide du modèle épistémologique développé ? Quels critères doivent-il vérifier ? Quels problèmes répondant à ces critères peut-on proposer ?*

Question Q9. *Quels éléments épistémologiques sont à prendre en compte pour construire des situations autour de l'algorithme ? Quelles situations peut-on proposer ?*

La réponse à ces questions est un travail en cours. Nous nous restreindrons ici à faire des propositions de problèmes et nous nous concentrerons sur l'apport de nos modèles épistémologiques pour la conception et l'analyse de situations pour la classe autour de l'algorithme.

1 Problèmes fondamentaux

1.1 Problèmes et dialectique outil-objet

Les problèmes : au cœur de l'activité algorithmique

Dans l'analyse épistémologique par ASPECTS, nous avons insisté sur l'importance de l'aspect PROBLÈME pour le concept algorithme. Plus précisément, le concept algorithme ne prend son sens que s'il permet de résoudre une famille d'instances d'un problème. Cet aspect est fondamental, et le développement de situations pour l'algorithme doit mettre la notion de problème au centre de l'activité.

Lors du développement du modèle des μ-conceptions au chapitre 3, nous avons proposé une définition d'un problème, reprise de (Giroud, 2011), et inspirée de la théorie de la complexité algorithmique :

> Un problème p est un couple (I, Q), où I est l'ensemble des instances du problème et Q une question portant sur ces instances (et spécifiant les propriétés de la solution attendue).

Cette notion de problème, dans le cadre du modèle cK¢, nous a permis de spécifier différents ensembles de problèmes en précisant leurs rapports à l'algorithme. En particulier, grâce aux familles $\mathscr{P}_{\text{A}}$ et $\mathbb{P}$, nous avons pu préciser la dualité outil-objet [3].

Dialectique outil-objet

La dualité outil-objet peut être un outil efficace pour sélectionner des problèmes adaptés à la construction de situations. Les problèmes de $\mathscr{P}_{\text{A}}$ qui peuvent aussi être l'objet de problèmes de $\mathbb{P}$ [4] sont de bons candidats pour mette en place une dialectique outil-objet.

Nous différencions la dualité outil-objet, qui décrit deux statuts différents de l'algorithme, de la dialectique outil-objet, qui s'intéresse à la relation qu'entretiennent ces deux statuts.

2. Comme nous l'avons précisé en introduction, c'est la construction de situations qui est à l'origine des questionnements épistémologiques.

3. Rappelons que $\mathscr{P}_{\text{A}}$ représente l'ensemble des problèmes algorithmiquement résolubles et $\mathbb{P}$ l'ensemble des problèmes d'algorithmique, c'est-à-dire qui portent sur des algorithmes ou des problèmes de $\mathscr{P}_{\text{A}}$. Dans la résolution de $\mathscr{P}_{\text{A}}$, l'algorithme est outil, dans $\mathbb{P}$ l'algorithme est objet.

4. Ces problèmes de $\mathbb{P}$ prendrons en instance le problème de $\mathscr{P}_{\text{A}}$ ou un algorithme résolvant ce dernier.

Nous avons un peu abordé les interactions entre algorithme-outil et algorithme-objet dans la première partie, mais la façon dont algorithmes outils et objets interagissent dans l'activité et les allers-retours qui permettent la construction d'un algorithme et sa validation demanderaient d'être étudiés plus en détail [5].

Il nous semble qu'un des objectifs d'une situation pour l'algorithme peut être de faire entrer l'élève [6] dans cette dialectique, autrement dit de mettre en jeu la relation entre des problèmes de $\mathscr{P}_{\text{A}}$ et des problèmes de $\mathbb{P}$.

1.2 La notion de problème fondamental pour l'algorithme

Une nouvelle famille de problèmes

Cela nous amène à définir une nouvelle famille, celle des problèmes adaptés à la construction de situations pour l'algorithme par la richesse de l'activité qu'ils permettent. Nous noterons cette sous-famille $\mathscr{P}_{\text{FA}}$, pour *problèmes fondamentaux pour l'algorithme.*

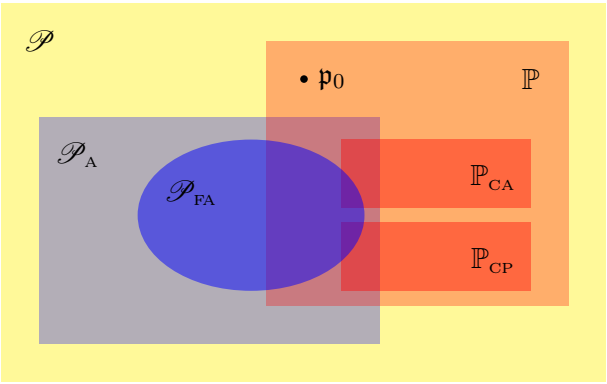
Le terme fondamental fait volontairement référence à la notion de *situation fondamentale* de la *théorie des situations didactiques* :

> **Situation fondamentale (correspondant à un savoir)**
> C'est un schéma de situation capable d'engendrer par le jeu des variables didactiques qui la déterminent, l'ensemble des situations correspondant à un savoir déterminé. Une telle situation, lorsqu'on peut l'identifier, offre des possibilités d'enseignement mais surtout une représentation du savoir par les problèmes où il intervient permettant de restituer le sens du savoir à enseigner. (Brousseau, 2010, p. 1)

Dans notre cas, il s'agit précisément d'identifier les problèmes où intervient le concept algorithme et permettant de « restituer son sens ».

Nous considérons que ces problèmes doivent être des problèmes de $\mathscr{P}_{\text{A}}$, afin que soit en jeu la relation problème-algorithme. C'est en questionnant cette relation entre le problème et un algorithme le résolvant qu'entrent en jeu le problème $\mathfrak{p}_0$ et la famille $\mathbb{P}$.

Le schéma proposé au chapitre 3, enrichi de la famille $\mathscr{P}_{\text{FA}}$, devient :



5. De manière générale, il nous semble que le *concept-problème*, proposé par Giroud (2011), en permettant d'expliciter les relations entre les problèmes, constitue un cadre favorable à l'explicitation de la dialectique outil-objet.

6. Nous utiliserons *élève* comme terme générique pour désigner le *sujet* dans les situations.

Caractéristiques

Caractériser l'ensemble P d'une conception donnée est complexe :

> La difficulté est de caractériser une conception par les problèmes dans lesquels elle est impliquée, soit par des problèmes générateurs, soit par les caractéristiques de sa sphère de pratique. [...] il s'agit de trouver une solution qui permette d'éviter de renvoyer à un ensemble de situations trop large (comme ce pourrait être le cas chez Vergnaud), tout en évitant également d'associer une conception à un ensemble de problèmes spécifiques mais difficilement constructible (comme le suggère Brousseau à partir du concept épistémologique de situation fondamentale). (Balacheff & Margolinas, 2005, p. 7-8)

Caractériser $\mathscr{P}_{\text{FA}}$ soulève les mêmes difficultés. Nous proposerons simplement quelques critères importants que l'on peut attendre d'un problème fondamental pour l'algorithme :

- Il est résoluble algorithmiquement ($\in \mathscr{P}_{\text{A}}$).
- Le concept d'algorithme est indispensable à sa résolution.
- Il soulève des problèmes dans $\mathbb{P}$ (en particulier, autour de la complexité).

Nous ajoutons deux critères non-nécessaires mais pouvant contribuer à la richesse d'une situation qui se baserait sur ce problème :

- Il met en jeu ou peut être traité dans plusieurs paradigmes :
  Un problème peut être riche algorithmiquement en ne mettant en jeu qu'un paradigme mais les changements de paradigmes nous semblent enrichir l'activité et la manipulation du concept algorithme
- Il soulève de nouveaux problèmes dans $\mathscr{P}_{\text{A}}$:
  On peut imaginer qu'un algorithme fondamental ne soulève pas directement d'autres problèmes de $\mathbb{P}$ mais le passage d'un problème à un autre et les différentes relations que peuvent entretenir deux problèmes algorithmiques apportent une dimension supplémentaire au problème.

Reprenons quelques exemples rencontrés dans les ressources pour le lycée :

- Le calcul de la somme des n premiers entiers n'entre pas dans $\mathscr{P}_{\text{FA}}$: la notion d'algorithme n'est pas indispensable à sa résolution.
- De manière générale, formules, traductions de fonctions et programmation de suites n'entrent pas dans $\mathscr{P}_{\text{FA}}$: ce sont simplement des expressions d'autres concepts dans un cadre algorithmique.
- Les problèmes de recherche qui ne proposent qu'un parcours linéaire de l'ensemble (comme le problème de recherche du maximum d'une liste) n'entrent probablement pas dans $\mathscr{P}_{\text{FA}}$, étant donné qu'ils soulèvent très peu de questions dans $\mathbb{P}$ (en particulier au niveau de la complexité).
- Les PROGRAMMES DE MODÉLISATION-SIMULATION et les ALGORITHMES-INSTANCIÉS n'entrent pas dans $\mathscr{P}_{\text{FA}}$: il s'agit simplement d'automatiser une tâche fastidieuse ou répétitive et la relation au problème est ambigüe.
- La recherche du *pgcd*, les problèmes de tri, les problèmes menant à la dichotomie sont très certainement dans $\mathscr{P}_{\text{FA}}$.

Identification d'un problème fondamental

Une description de tous les problèmes de $\mathscr{P}_{\text{FA}}$ n'étant pas envisageable, la question d'identifier un problème comme fondamental se pose. Il est souvent facile de justifier qu'un problème n'est pas fondamental pour l'algorithme (en montrant qu'il invalide l'un de nos critères). Il est bien plus difficile de prouver qu'un problème est dans $\mathscr{P}_{\text{FA}}$. Cela demande une analyse épistémologique et mathématique poussée.

Nous présenterons des problèmes qui nous semblent pouvoir prétendre au titre de problème fondamental pour l'algorithme. Ils forment un ensemble de pistes pour le développement de situations et nous n'allons pas les analyser en détail ici.

Nous développerons l'analyse d'un élément de $\mathscr{P}_{\text{FA}}$ afin d'illustrer notre propos[7]. Nous discuterons de la façon dont ce problème peut fournir une situation pour la classe.

2 Quelques propositions de problèmes

Nous proposons maintenant un ensemble de problèmes, qui appartiennent certainement à $\mathscr{P}_{\text{FA}}$. Nous présenterons les problèmes et les questions qu'ils peuvent mettre en jeu. La majorité de ces problèmes se placent dans le champ des mathématiques discrètes ou des champs voisins. Les résultats des chapitres précédents laissent penser qu'il est plus difficile de proposer des problèmes fondamentaux ailleurs.

Certains problèmes mettent en jeu des concepts mathématiques avancés tandis que d'autres s'appuient sur des notions relativement simples. Notre objectif n'est pas uniquement de produire des situations pour le lycée, c'est pourquoi nous proposons des problèmes très divers.

Pour faciliter la compréhension, nous ne décrirons pas les problèmes en donnant l'ensemble des instances mais en donnant une instance générique. Il faut comprendre que le problème porte sur toute instance de ce type[8].

• Géométrie algorithmique

La géométrie algorithmique s'intéresse à la production et à l'étude d'algorithmes pour des problèmes de géométrie, qui mettent généralement en jeu un nombre arbitraire d'objets (nuages de points, polygones avec un nombre quelconque de sommets, ensembles de droites, etc.). L'instance est souvent une famille de taille arbitraire de ces objets. Souvent, l'algorithme est nécessaire pour résoudre le problème pour toute instance.
Les problèmes de géométrie peuvent être traités dans AM ou dans AI et soulèvent des questions de représentation des objets. Certains problèmes d'existence peuvent être traités dans PA.

Enveloppe convexe :

Instance : Un ensemble fini de points du plan

Question : Quels sont les points qui décrivent l'enveloppe convexe de l'ensemble ?

7. Et de prouver $\mathscr{P}_{\text{FA}} \neq \emptyset$.

8. "Instance : 2 entiers" signifie que l'ensemble des instances est $\mathbb{N}^2$ ou "Instance : une famille finie de points du plan" signifie que l'ensemble des instances est l'ensemble des familles finies de points. La description peut vite devenir pesante.

Sandwich discret :

Instance : Deux ensembles de points (rouges et bleus) dans le plan de cardinaux pairs

Question : Existe-t-il une droite qui partage chacun des ensembles en deux parties de même taille ?

Variante avec une seule couleur.

Ghostbusters :

Instance : Un ensemble de points blancs et un ensemble de points noirs de même cardinal, dans le plan

Question : Peut-on associer chaque point blanc à un point noir de manière à ce que les segments obtenus ne s'intersectent pas ?

Coloration :

Instance : Un ensemble de droites du plan

Question : Combien de couleurs faut-il au minimum pour colorier les régions délimitées sans avoir deux couleurs adjacentes ?

Droite discrète :

Instance : Les coordonnées de deux pixels du plan discret

Question : Quels sont les pixels qui appartiennent à la droite passant par ces deux points ?

Découpage de polygones :

Instance : Deux polygones

Question : Peut-on découper le premier polygone en un nombre fini de morceaux et les assembler pour obtenir le second ?

• Méthodes de recherche

Dichotomie :

Instance : Une liste triée

Question : Comment savoir si un élément est présent dans la liste ?

On veut obtenir la réponse en faisant le moins de requêtes possible.

Mastermind

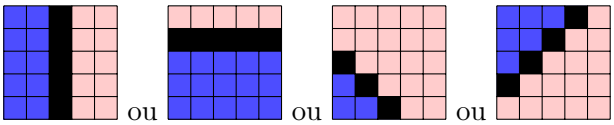
Le jeu du Mastermind consiste à deviner un code composé de quatre couleurs choisies parmi huit. À chaque proposition, l'adversaire répond en donnant le nombre d'éléments du code justes (bonne couleur en bonne position) et le nombre de couleurs présentes mais mal placées.

Instance : Un code choisi par l'adversaire

Question : Comment retrouver le code de l'adversaire ?

On veut, bien sûr, répondre à la question en utilisant le moins de demandes possible.

Cherchez la frontière
Sur une grille, une frontière peut être une ligne horizontale, verticale ou diagonale et sépare deux territoires (rouge et bleu). Par exemple :



La frontière n'est pas connue et l'on peut interroger les cases une par une pour connaître leur couleur.
Le problème peut être formulé comme ceci :

Instance : Un choix inconnu de frontière

Question : Comment retrouver la frontière ?

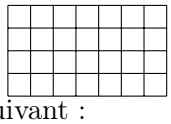
On cherche la méthode qui demandera le moins d'interrogations de cases.

• Optimisation Combinatoire

Problème de téléspectateur :

Instance : Un ensemble d'émissions décrites par leur heure de début et de fin

Question : Quel est le nombre maximal d'émissions que l'on peut regarder ? (on ne peut pas regarder deux émissions en même temps)

Problème de pavages
On se donne une grille et une forme de pavé.
Par exemple la grille  et la forme  .
Le problème est le suivant :

Instance : Une grille et une forme

Question : Combien de pavés au maximum peut on placer dans la grille ?

• Arithmétique

pgcd de plusieurs entiers :

Instance : Un nombre fini d'entiers

Question : Quel est leur *pgcd* ?

Problème de Frobenius :

Instance : Un système de pièces de monnaie

Question : Quelles sommes peut-on payer dans ce système monétaire ?

• Graphes

Coloration :

Instance : Un graphe

Question : Combien de couleurs faut-il au minimum pour colorier le graphe sans que deux sommets voisins soient de la même couleur ?

Euler :

Instance : Un graphe

Question : Existe-t-il un cycle eulérien ?

• Théorie des jeux

Les stratégies en théorie des jeux peuvent être vues comme des algorithmes. Les paradigmes AM et PA peuvent être en jeu. Nous avons vu que AI n'était pas toujours très adapté à la théorie des jeux.

Ici, le problème est toujours :

Instance : Une situation du jeu

Question : Peut-on gagner à coup sûr ? Avec quelle stratégie ?

Variante : On peut chercher la stratégie permettant de gagner le plus rapidement.

Nim
Plusieurs tas d'allumettes sont disposés devant les deux joueurs. À tour de rôle, chaque joueur peut prendre autant d'allumettes qu'il veut dans un seul tas. Le joueur qui ne peut plus jouer perd.

Wythoff
Deux tas d'allumettes sont placés devant les joueurs. Les règles sont les mêmes qu'au jeu de Nim mais l'on peut aussi retirer un même nombre d'allumettes au deux tas.

• Solitaires

Pour les solitaires, c'est aussi la stratégie gagnante qui peut être vue comme un algorithme. Le problème est toujours :

Instance : Une situation

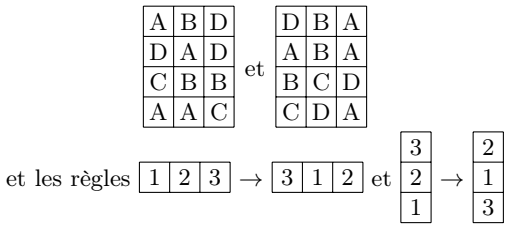
Question : Peut-on finir avec un seul pion ?

Solitaire "classique"
On joue sur une grille, avec des pions partout sauf sur une case. On peut manger un pion en le sautant avec un autre : ●●□ → □□● (dans les quatre directions)

Clobber
On joue sur une grille, avec des pions noirs et blancs. Si un pion blanc et un pion noir sont voisins, le pion blanc peut manger le noir en prenant sa place ou le pion noir peut manger le blanc en prenant sa place :
●○ → □● ou ●○ → ○□ (dans les quatre directions)

• Autres

Taquin
On se donne deux grilles colorées et un ensemble de règles qui sont des permutations d'éléments. Par exemple, on se donne les grilles :



Le problème se pose ainsi :

Instance : Deux grilles colorées et un ensemble de règles

Question : Peut-on passer d'une grille à l'autre avec les règles proposées ?

3 Quels types de situations pour l'algorithme ?

3.1 Objectifs

Notre but est de développer des situations pour l'algorithme. Nous l'avons vu, le concept algorithme est indissociable de la notion de problème. Toute situation pour l'algorithme se doit de mettre le problème au centre des questions.

Cependant, l'épistémologie de l'algorithme révèle d'autres éléments qui doivent guider la conception de situations. En particulier le lien privilégié à la preuve et la spécificité des problèmes de $\mathbb{P}$ doivent être pris en compte. Nous avons déjà abordé ces questions concernant le choix des problèmes à mettre en jeu dans les situations. Nous voulons évoquer ici les implications sur l'organisation des situations elles-mêmes.

Preuve La preuve intervient de diverses manières autour du concept algorithme et nous pensons qu'une situation pour l'algorithme doit porter un enjeu de preuve. Cela implique que la situation mette l'élève en position de rechercher et de construire la validation de son travail. En particulier, il nous semble cohérent que le problème $\mathfrak{p}_0$ puisse émerger. De tels enjeux nécessitent une activité sur un temps long et une gestion adaptée.

Dialectique outil-objet Pour que l'algorithme soit en jeu dans une situation, il nous semble indispensable qu'il ne soit pas uniquement présent en tant qu'outil. La question de preuve dont nous venons de discuter contribue à cela, ainsi que les problèmes de $\mathbb{P}$ (de complexité en particulier). Des situations mettant en jeu une dialectique outil-objet demandent à être menées sur plusieurs séances et organisées de manière à favoriser cette dialectique. Les allers-retours entre un même algorithme tour à tour outil et objet doivent être favorisés par le milieu.

Paradigmes AM et PA En nous concentrant sur les aspects de preuve et de dialectique outil-objet, il nous semble que des situations pour l'algorithme doivent pouvoir se détacher de la programmation et du paradigme AI. L'utilisation de l'outil informatique dans une telle situation constitue un paramètre qu'il nous semble préférable de délaisser dans un premier temps. Bien entendu, nous ne négligeons pas son rôle dans la construction du concept algorithme et nous n'excluons pas de mettre en jeu l'informatique dans des situations futures.

3.2 Situations de Recherche en Classe (SiRC)

Les points que nous venons de soulever concernant les situations pour l'algorithme nous amènent à proposer des situations qui mettent les élèves en situation de résolution de problème en groupe, face à un problème pour lequel ils peuvent adopter une démarche proche de celle du chercheur (essais-erreurs, conjectures, preuve, modification du problème, etc.). Les *Situations de Recherche en Classe* (SiRC), se prêtent particulièrement bien à ce type d'activité.

Les travaux sur les SiRC proposent la caractérisation suivante (Grenier & Payan, 2003 ; Godot & Grenier, 2004 ; Cartier, Godot, Knoll, & Ouvrier-Buffet, 2006) :

(a) Le problème abordé doit être proche de questions de recherche vivantes.

(b) La question initiale doit être facile d'accès pour l'élève.

(c) Des stratégies d'approche simples doivent exister pour permettre d'entrer dans une démarche de recherche.

(d) De nombreuses stratégies de recherche peuvent être mises en place et les méthodes de résolution ne sont pas désignées.

(e) Une question résolue peut amener à se poser d'autres questions : il n'y a que des critères de fin "locaux".

Les SiRC correspondent aussi à un type de situations particulier qui implique un travail sur plusieurs séances et une gestion spécifique.

Ces critères répondent à plusieurs de nos objectifs. Plus précisément, la multiplicité des stratégies possibles nous semble importante pour que la construction d'algorithmes, leur validation, leur comparaison ou leur étude puissent vivre. Un autre élément, transversal aux critères ci-dessus, nous semble important : l'accessibilité du problème. Il nous semble qu'une situation pour l'algorithme doit être proposée dans un cadre bien connu de l'élève afin de favoriser l'émergence de questions concernant l'algorithme, sans que des notions trop avancées ne fassent obstacle à celles visées.

3.3 Remarques sur les problèmes d'optimisation

Nous avons souligné l'intérêt d'une dialectique outil-objet dans une situation pour l'algorithme. Nous souhaitons évoquer un certain type de problèmes qui nous semble favorable à cette dialectique : les problèmes de recherche d'algorithmes optimaux. De manière générale, ils se présentent ainsi :

Instance : Un problème $p \in \mathscr{P}_{\text{A}}$

Question : Quel est l'algorithme le plus efficace pour répondre à p.

Selon p, il est nécessaire de préciser quel est le critère d'efficacité que l'on prend en compte.

Il nous semble que l'enjeu d'optimisation permet de mettre en œuvre une dialectique outil-objet, puisqu'il s'agit à la fois de construire un algorithme répondant au problème, mais aussi de garantir certaines propriétés :

– L'enjeu de validation de l'algorithme est renforcé. Si l'on propose un algorithme meilleur qu'un précédent, il faut être sûr qu'il répond bien à toutes les instances.

– La complexité est au cœur du problème sans que son calcul doive nécessairement être explicité.
– Les algorithmes deviennent objets, lorsqu'on les comparent entre eux.
– Prouver qu'un algorithme est optimal pour un problème donné demande souvent un changement de point de vue (passage au problème dual, généralisation, étude d'un algorithme générique qui résout le problème, etc.) par rapport à la preuve de sa validité.

Le problème que nous allons présenter est un problème d'optimisation. Nous verrons dans l'analyse mathématique tous ces éléments à l'œuvre.

4 Une proposition de situation pour l'algorithme

4.1 Le problème des pesées

Énoncé

Voici le problème que nous allons étudier :

Dans un ensemble de pièces indiscernables à la vue et au toucher, se trouvent des fausses pièces. Les vraies pièces pèsent toutes le même poids, les fausses aussi mais leur poids est différent de celui des vraies. À l'aide d'une balance Roberval à deux plateaux et sans poids, peut-on retrouver les fausses pièces ? Si oui, quelle est la méthode qui permet de les retrouver en effectuant le moins de pesées possible ?

Deux problèmes sont énoncés ici :
Le problème $\mathcal{P}_{\text{pesées}}$ de recherche des fausses pièces qui est dans $\mathscr{P}_{\text{A}}$:

Instance : Un ensemble de pièces contenant des fausses pièces

Question : Peut-on retrouver les fausses pièces ?

et celui de recherche de la meilleure méthode qui est dans $\mathbb{P}$:

Instance : Un problème de $\mathscr{P}_{\text{A}}$

Question : Quel est l'algorithme optimal pour le résoudre ?

Ici, le problème d'optimalité est instancié sur $\mathcal{P}_{\text{pesées}}$.

On peut définir plusieurs variantes de ce problème en jouant sur les variables suivantes :

- Poids des fausses pièces :
  - On sait au départ que les fausses pièces sont plus lourdes que les vraies (ou plus légères).
  - On ne sait rien.
- Le nombre de fausses pièces :
  - Fixé à l'avance.
  - Compris dans un intervalle donné.
  - Non connu à l'avance.

Ce problème, tout comme la plupart de ses variantes, n'est pas encore complétement résolu pour la recherche en mathématiques (voir l'analyse mathématique pour plus de détails). Nous limiterons notre étude à quelques variantes de ce problème.

Un problème adapté à une SiRC ?

L'énoncé du problème ne fait appel à aucune notion mathématique avancée mise à part la notion d'optimalité, mais qui peut être comprise dans un sens intuitif au départ. L'appropriation de ce problème nous paraît donc relativement facile.

De plus la construction d'une méthode, même grossière du point de vue de la complexité, peut être mise en place. Pour prouver qu'une méthode de recherche donnée trouve toujours les fausses pièces, des raisonnements simples suffisent la plupart du temps (disjonction de cas par exemple).

La question la plus délicate semble donc être celle de l'optimalité. En effet elle soulève le problème du choix d'un critère d'optimalité. Deux critères peuvent être utilisés : la *complexité au pire* et la *complexité en moyenne*.
La complexité au pire est le nombre de pesées qu'effectue un algorithme dans le pire des cas. Autrement dit, la complexité au pire pour un algorithme donné est le nombre maximum d'étapes (ici les pesées) qu'il effectue, maximum calculé sur l'ensemble des instances d'une taille donnée (ici pour un nombre fixé de pièces).
La complexité au pire peut être introduite sous forme d'un jeu à deux joueurs. Un joueur cherche les fausses pièces en proposant des pesées et l'adversaire décide du résultat de ces pesées (sans être en contradiction avec les résultats des pesées précédentes). Le joueur doit trouver les fausses pièces en utilisant le moins de pesées possible, tandis que son adversaire essaie de maximiser le nombre de pesées.Nous ne nous intéresserons pas ici au problème de l'optimalité pour la complexité en moyenne. Cela ne veut pas dire que cette question n'est pas intéressante du point de vue mathématique ou didactique.

Nous le verrons, la complexité au pire soulève déjà des problèmes riches. Elle demande moins d'outils mathématiques (de probabilité notamment) pour être manipulée. Dans la recherche actuelle, le problème est traité pour la complexité au pire.

On peut déjà affirmer que le problème des pesées répond aux critères (a) et (b) des SiRC. Nous avons aussi vu qu'il existe plusieurs variantes au problème, cela va dans le sens du caractère (e). Enfin, l'analyse mathématique mettra en avant de nombreuses stratégies de recherche envisageables. Cela montrera que ce problème répond aux critères (c) et (d) et confirmera le point (e).
Ce problème est donc adapté à être proposé sous la forme d'une SiRC.

4.2 Analyse mathématique du problème

On s'intéresse maintenant aux différentes stratégies envisageables en fonction des variables suivantes : le nombre de fausses pièces et les informations sur leur poids :

- Le nombre de fausses pièces peut être connu (1, 3, compris entre 0 et 2...) ou non.
- Le poids de la fausse pièce peut être connu (plus lourd par exemple) ou non.

Dans notre étude on se limitera aux cas où le nombre de fausses pièces n'excède pas un : c'est-à-dire lorsqu'il y a exactement une fausse pièce ou au plus une fausse pièce.
Pour des résultats avec un plus grand nombre de fausses pièces on pourra consulter par exemple (Tosic, 1983) qui donne un encadrement du nombre maximum de pesées pour trouver 2 fausses pièces plus lourdes parmi n, (Pyber, 1986) qui fait la même chose pour un nombre de pièces m quelconque ou (Aigner & Li, 1997) qui affinent les résultats des

deux articles précédents. Rappelons que de nombreuses variantes sont encore des problèmes ouverts.

Le problème des pesées est un problème d'optimisation. La résolution d'un tel problème repose sur l'étude de deux sous-problèmes :

- P_M : Construire une méthode de recherche de la fausse pièce pour un problème donné.
- P_O : Prouver l'optimalité de la méthode. On distinguera parmi les stratégies de résolution de P_O celles qui relèvent d'un argument d'optimalité locale et celles qui relèvent d'un argument d'optimalité globale.

P_M permet de travailler sur la condition suffisante et P_O sur la condition nécessaire.
Parfois les deux sous-problèmes sont traités simultanément : on construit une méthode et sa construction prouve son optimalité. Nécessairement cela s'appuiera sur un argument d'optimalité locale.

Pour le problème d'optimalité P_O on peut définir le problème **dual** P_O^* : si l'on fixe un nombre de pesées p, combien de pièces peut-on traiter avec p pesées ?
Parfois le passage à P_O^* pourra s'avérer fructueux pour répondre à P_O.

On distingue les quatre problèmes suivants :

- $P_{1,+}$: 1 fausse pièce, plus lourde.
- $P_{1,+/-}$: 1 fausse pièce, plus lourde ou plus légère.
- $P_{0/1,+}$: 0 ou 1 fausse pièce, plus lourde.
- $P_{0/1,+/-}$: 0 ou 1 fausse pièce, plus lourde ou plus légère.

Stratégies pour $\mathbf{P_{1,+}}$:

Dans une démarche de recherche on est souvent amené à étudier des petits cas afin d'essayer de mieux comprendre l'enjeu du problème et de pouvoir, par exemple, émettre des conjectures. Cela donne lieu à la stratégie suivante :

Stratégie expérimentale $\mathbf{S_{exp}}$: on traite les petits nombres de pièces (par exemple par énumération de cas). Cela peut aboutir à des généralisations formulées comme conjectures. D'autre part, pour un nombre de pièces fixé, si l'on a déjà traité tous les cas plus petits, on peut énumérer toutes les pesées possibles (de manière plus ou moins organisée) et déduire la méthode optimale pour ce cas. Cette méthode s'appuie sur un argument d'optimalité locale. Elle devient très vite fastidieuse mais permet d'avoir un grand nombre d'exemples sur lesquels s'appuyer pour faire des conjectures ou les invalider.

Pour répondre à ces conjectures il convient de formaliser ce qu'est une pesée. Un premier point de vue est qu'effectuer une pesée c'est prendre deux ensembles de pièces de même taille et de les comparer pour savoir lequel des deux est le plus lourd. Cette vision peut donner lieu à la stratégie :

Stratégie de dichotomie $\mathbf{S^1_{dicho}}$: pour traiter n pièces, on compare $\lfloor \frac{n}{2} \rfloor$ pièces sur chaque plateau. Le déséquilibre permet de garder $\lfloor \frac{n}{2} \rfloor$ pièces à tester, on réitère ensuite le processus. Si n est impair, un équilibre de la balance est possible et indique que la fausse pièce est celle laissée de côté. Cela donne un algorithme en $\lfloor \log_2(n) \rfloor$ pesées au pire.

Cette stratégie ne conduit pas à un algorithme optimal, il suffit pour s'en convaincre de regarder le problème pour 9 pièces. Avec S^1_{dicho} il faudra 3 pesées alors que S_{exp} donne une méthode en 2 pesées. L'optimalité de cet algorithme peut reposer sur la conception erronée : plus on pèse de pièces à la fois plus on est efficace (argument d'optimalité locale). En fait cette méthode ne tient pas assez compte du fait qu'une pesée donne aussi de l'information sur l'ensemble des pièces non pesées.
En effet, effectuer une pesée c'est faire 3 tas : 2 de même taille A et B que l'on compare et 1 tas C que l'on laisse de côté. Alors 3 résultats sont possibles :

- A est plus léger que B et alors B contient la fausse pièce.
- A est plus lourd que B et alors A contient la fausse pièce.
- A et B sont de même poids et c'est le tas C qui contient la fausse pièce.

En s'appuyant sur cette observation, on peut mettre en œuvre la stratégie suivante :

Stratégie de trichotomie $\mathbf{S_{tricho}}$: on divise le tas de pièces restantes en trois paquets de « même taille », i.e. tels que $|A| = |B|$ et $||A| - |C|| \leqslant 1$.
Appelons l'algorithme ainsi obtenu $\mathcal{A}_{1,+}$.
Pour n pièces, avec $3^{k-1} < n \leqslant 3^k$, cet algorithme demande k pesées, autrement dit $\lceil \log_3(n) \rceil$ pesées. Cette méthode permet donc de traiter avec k pesées jusqu'à 3^k pièces.

Proposition 1. *L'algorithme $\mathcal{A}_{1,+}$ est optimal pour la complexité au pire.*

On propose 3 preuves de ce résultat :

Preuve 1 : À chaque étape, si l'on découpe l'ensemble des k pièces restantes en 3 suivant un autre découpage, il y aura un des paquets de taille supérieure à $\lceil \frac{k}{3} \rceil$. Selon le résultat de la pesée, on peut se retrouver à chercher la fausse pièce dans ce paquet. Il faudra alors autant ou plus de pesées pour la trouver.
On peut formaliser cela avec une formule de récurrence. En notant $\mu(n)$ le nombre de pesées nécessaires pour retrouver la fausse pièce parmi n, on a :

$$\mu(n) = 1 + \min_{2i+j=n} (\max(\mu(i), \mu(j)))$$

où chaque couple (i, j) représente une décomposition possible de n en 2 tas de i pièces que l'on pèse et un tas de j pièces laissé de côté. □

Preuve 2 : Soit $\mathcal{A}$ un algorithme de recherche de la fausse pièce pour $P_{1,+}$. Construisons l'arbre de décision de $\mathcal{A}$ sur la donnée de n pièces : chaque nœud représente une pesée, chaque branche représente un résultat possible de la pesée et relie cette pesée à la pesée suivante à effectuer selon le résultat obtenu. À chaque feuille correspond une position de la fausse pièce parmi les n. La figure 9.1 donne une représentation d'un tel arbre. Comme $\mathcal{A}$ est un algorithme pour $P_{1,+}$, chaque position de la fausse pièce correspond à au moins une feuille, i.e. en posant f le nombre de feuilles, on a $n \leqslant f$.
Le nombre de pesées au pire pour cet algorithme sur n pièces correspond à la profondeur p de l'arbre. Comme chaque nœud a au plus 3 fils, on a la formule suivante : $f \leqslant 3^p$ d'où $n \leqslant 3^p$. Donc $\mathcal{A}$ ne peut traiter plus de 3^p pièces en p pesées, autrement dit on ne peut traiter n pièces en moins de $\lceil \log_3 n \rceil$ pesées.
Donc l'algorithme construit par S_{tricho} est optimal. □

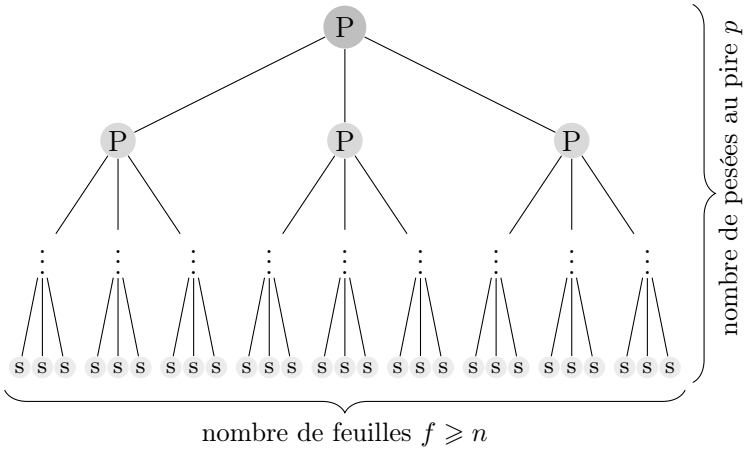


FIGURE 9.1 – L'arbre de décision de l'algorithme $\mathcal{A}$. Les nœuds notés P représentent des pesées, les nœuds notés s les résultats en sortie de l'algorithme.

Preuve 3 : On passe au problème dual : combien de pièces peut-on traiter avec p pesées ? Notons $\eta(p)$ le nombre maximal de pièces que l'on peut traiter avec au plus n pesées. Montrons que $\eta(p) = 3^p$ par récurrence.
On peut vérifier que $\eta(0) = 1$ et $\eta(1) = 3$.
Supposons $\eta(p) = 3^p$ et intéressons-nous à $\eta(p+1)$: lors de la première pesée on va partager les pièces en 3 tas A, B et C, la pièce pouvant être dans chacun d'eux, il faut pouvoir l'y retrouver en p pesées. Donc il faut $|A|, |B|, |C| \leqslant 3^n$ par hypothèse de récurrence et donc $\eta(p+1) = 3 \times 3^p = 3^{p+1}$. □

Notons que la première preuve s'appuie sur un argument d'optimalité locale alors que les deux autres s'appuient sur un argument d'optimalité globale.

Une autre façon de concevoir une pesée, est de n'interpréter le résultat que comme un équilibre ou un déséquilibre, sans tenir compte de quelles pièces sont sur le plateau le plus lourd. Cela peut conduire à la stratégie :

Stratégie par étalon $\mathbf{S_{etal}}$: on utilise un ensemble E de pièces identifiées comme vraies pour tester un ensemble A de pièces inconnues par comparaison. Soit les pièces de A sont vraies et elles s'ajoutent à l'ensemble E des pièces étalons, soit on sait que A contient la fausse pièce et il ne reste plus qu'à la retrouver. Pour ce faire on peut, par exemple, identifier des bonnes pièces dans A grâce à E et les enlever jusqu'à n'en avoir plus qu'une.

Stratégie de passage par un autre problème $\mathbf{S_{prob}}$: on se ramène à des problèmes déjà traités ou qui paraissent plus simples (voire on déduit de leur optimalité celle de la méthode construite).

Stratégies pour $\mathbf{P_{1,+/-}}$:

On retrouve les stratégies $\mathbf{S_{exp}}$ et $\mathbf{S_{prob}}$.

On retrouve $\mathbf{S_{tricho}}$ avec une légère variante : on divise le nombre de pièces par 3 à chaque pesée et l'on pèse les deux premiers tas. S'il y a équilibre, la fausse pièce est dans le troisième tas et on recommence l'opération. Dès le premier déséquilibre de la balance, on

effectue une pesée supplémentaire d'un des tas avec des vraies pièces pour savoir si la fausse pièce est plus légère ou plus lourde puis on poursuit comme pour $P_{1,+}$. Pour un nombre de pièces de départ de la forme $n = 3k + 2$, si ce déséquilibre survient à la première pesée, une vraie pièce supplémentaire peut être nécessaire, ou bien alors on peut comparer des paquets de k pièces à la première pesée.
Si l'on dispose d'une pièce supplémentaire, on obtient un algorithme en $\lceil \log_3(n) \rceil + 1$ pesées.
Si l'on s'interdit une pièce supplémentaire, on a un algorithme en $\lceil \log_3(n) \rceil + 2$ pesées si $n = 3k + 2$, et en $\lceil \log_3(n) \rceil + 1$ sinon.

On peut appliquer $\mathbf{S^1_{dicho}}$ avec le même genre de variante : en 2 ou 3 pesées on cherche à savoir si la fausse pièce est plus légère ou plus lourde. Puis on applique la stratégie de dichotomie décrite pour $P_{1,+}$.

Il est nécessaire, ici aussi, de se demander quelle information apporte une pesée. Les deux stratégies précédentes s'appuient sur la conception erronée qu'une pesée donne plus d'information une fois que l'on connait le poids de la fausse pièce et qu'il faut déterminer cette information le plus tôt possible. Si au contraire on ne cherche pas à déterminer le poids de la fausse pièce, une pesée peut être vue comme délivrant uniquement l'information : la fausse pièce est sur l'un des plateaux (déséquilibre) ou elle est parmi les pièces non pesées (équilibre).Cela justifie alors les deux stratégies suivantes :

La stratégie $\mathbf{S_{etal}}$ déjà présentée : on utilise un ensemble E de pièces identifiées comme vraies pour tester des ensembles de pièces inconnues par comparaison. Cette stratégie est plus légitime pour $P_{1,+/-}$ car on n'oublie pas l'information du poids de la pièce comme c'est le cas pour $P_{1,+}$.

Stratégie de dichotomie $\mathbf{S^2_{dicho}}$: on pèse $\frac{1}{4}$ des pièces sur chaque plateau. S'il y a déséquilibre, la fausse pièce est sur la balance, s'il y a équilibre elle est dans la moitié non pesée. S'il ne reste que deux pièces on utilise une pièce identifiée comme vraie pour la dernière étape. On a ainsi un algorithme qui se comporte en $\log_2(n)$.
Un argument erroné d'optimalité locale pourrait être utilisé pour justifier cette méthode comme la meilleure : en une pesée on ne peut pas faire mieux que de diviser les pièces en 2 ensembles.

Toutes les stratégies présentées jusque-là pour $P_{1,+/-}$ n'utilisent pas toutes les informations délivrées par une pesée : lorsque l'on effectue une pesée on partage les pièces en 3 ensembles A, B et C avec $|A| = |B|$ et on pèse A et B. En cas d'équilibre c'est C qui contient la fausse pièce et on ne sait rien de plus. Par contre, en cas de déséquilibre on sait bien sûr que la fausse pièce est dans $A \cup B$, mais on sait aussi lequel de A ou de B pèse le plus lourd. Cette information peut avoir son utilité car une pièce qui a été parmi les plus lourdes lors d'une pesée et parmi les plus légères dans une autre est nécessairement une vraie pièce. Conserver toutes ces informations après chaque pesée peut donner lieu à la modélisation suivante :

Modélisation par marquage des pièces : pour conserver l'information on utilise un marquage des pièces après une pesée déséquilibrée : $\ominus$ pour les pièces du côté le plus léger et $\oplus$ pour les pièces du côté le plus lourd. En fait le marquage $\oplus$ (respectivement $\ominus$) signifie que la pièce est soit vraie, soit fausse et plus lourde (respectivement soit vraie soit fausse et plus légère). Après une première pesée déséquilibrée, on peut enlever les pièces

de C et marquer les pièces de $A \cup B$. Se pose alors un nouveau problème :

$\mathbf{P^M_{1,+/-}}$: Comment retrouver en utilisant le moins de pesées possible, une fausse pièce parmi un ensemble de pièces marquées ?

Si au contraire la première pesée est équilibrée, alors la fausse pièce est dans C, et on se pose le même problème qu'au départ avec une légère différence : on a maintenant à disposition un ensemble de vraies pièces. Cela conduit à s'intéresser au problème :

$\mathbf{P^B_{1,+/-}}$: Comment retrouver en utilisant le moins de pesées possible, une fausse pièce de poids inconnu, en ayant une bonne pièce à disposition.

Pour appréhender ces nouveaux problèmes on peut employer des stratégies du type S_{exp}, S^1_{dicho}, S^2_{dicho}, S_{etal} ou S_{prob}. Mais c'est en combinant la stratégie de trichotomie S_{tricho} et la modélisation par marquage que l'on peut obtenir l'algorithme optimal pour le problème $P_{1,+/-}$:

Voyons d'abord comment trouver la fausse pièce parmi un ensemble de pièces marquées $\oplus$ ou $\ominus$. Passons au problème dual : combien de pièces au maximum peut-on traiter à coup sûr avec p pesées ? Posons $f^M_{1,+/-}(p)$ ce nombre de pièces.

Proposition 2. $f^M_{1,+/-}(p) = 3^p$.

Autrement dit, on peut retrouver la fausse pièce en $\lceil \log_3(n) \rceil$ pesées.

Preuve 4 : Montrons $f^M_{1,+/-}(p) \leqslant 3^p$:
Il y a 3^p résultats différents pour une suite de p pesées. Donc si l'on traitait $m > 3^p$ pièces, on aurait 2 entiers i et j tels que les cas où la fausse pièce est la i^{e} ou la j^{e} donneront la même suite de pesées avec les mêmes résultats. Ces deux cas ne seraient pas discernables.

Montrons $f^M_{1,+/-}(p) \geqslant 3^p$ en construisant une méthode qui peut traiter jusqu'à 3^p pièces en p pesées.
Raisonnons par récurrence sur p : avec une pesée on peut traiter 2 et 3 pièces donc $f(1) = 3$.
Supposons que $f(p-1) = 3^{p-1}$. Si l'on a $m \leqslant 3^p$ pièces, alors on peut les partager en 3 ensembles A, B et C tels que l'on ait, avec les notations $|X|_{\ominus}$ et $|X|_{\oplus}$ pour les nombres de pièces marquées $\ominus$ et $\oplus$ dans l'ensemble X :
$|A| = |B|$, $|A| - 1 \leqslant |C| \leqslant |A| + 1$ et $|A|_{\ominus} = |B|_{\ominus}$ (et donc $|A|_{\oplus} = |B|_{\oplus}$).
On pèse les ensembles A et B :
S'il y a équilibre, la fausse pièce se trouve dans C et on peut la retrouver en $p-1$ pesées car $|C| \leqslant 3^{p-1}$.
S'il y a déséquilibre, par exemple si A est plus léger que B, la fausse pièce se trouve parmi les pièces marquées $\ominus$ de A ou parmi les pièces marquées $\oplus$ de B. Cela représente un ensemble de $|A|_{\ominus} + |B|_{\oplus} = |A| \leqslant 3^{p-1}$ pièces que l'on sait traiter en $p-1$ pesées.
Donc on sait traiter nos m pièces en p pesées. Cela conclut la récurrence et prouve la proposition. □

Remarque : Dans le cas présent, lorsque l'on trouve la fausse pièce, on sait si elle est plus lourde ou plus légère.

Revenons au problème général où les pièces ne sont pas marquées au départ et passons au problème dual en posant :

• $f_{1,+/-}(p)$: le nombre maximum de pièces, non forcément marquées, parmi lesquelles on peut trouver la fausse pièce en p pesées.
• $f^B_{1,+/-}(p)$: le nombre maximum de pièces, non forcément marquées, parmi lesquelles on peut trouver la fausse pièce en p pesées en ayant à disposition une bonne pièce supplémentaire.

Proposition 3. $f^B_{1,+/-}(p) = \dfrac{3^p+1}{2}$ *et* $f_{1,+/-}(p) = \dfrac{3^p-1}{2}$.

Autrement dit, une méthode optimale pour $P_{1,+/-}$ avec n pièces, utilise $\lceil \log_3(2n+1) \rceil$ pesées.

Preuve 5 : Pour faciliter la lecture de cette preuve, on pose $f_{1,+/-} = g$ et $f^B_{1,+/-} = h$. On a $g(1) = 1$ et $h(1) = 2$.
Soit un ensemble de m pièces que l'on sait traiter en k pesées. Faire une première pesée, c'est partager les pièces en 3 ensembles A, B et C tels que $|A| = |B|$ et comparer A et B. S'il y a équilibre, la fausse pièce est dans C et on dispose de bonnes pièces dans $A \cup B$, donc on sait traiter C en $p-1$ pesées *ssi* $|C| \leqslant h(p-1)$.
S'il y a déséquilibre, par exemple si A est plus lourd que B, on marque $\oplus$ les pièces de A et $\ominus$ les pièces de B et la fausse pièce se trouve dans $A \cup B$. On peut alors traiter $A \cup B$ en $p-1$ pesées *ssi* $|A| + |B| \leqslant f(p-1) = 3^{p-1}$.
Donc $m = |A| + |B| + |C| \leqslant h(p-1) + f(p-1)$
Mais $f(p-1)$ est un nombre impair donc on ne peut jamais partager $f(k-1)$ pièces en 2 ensembles A et B de même taille. Donc sans bonne pièce à disposition on a : $m \leqslant h(p-1) + f(p-1) - 1$ et donc $g(p) = h(p-1) + f(p-1) - 1$.
Par contre, si l'on dispose d'une bonne pièce supplémentaire, on peut la rajouter aux $f(p-1)$ autres pour égaliser les tailles des ensembles sur la balance.
Alors on a $m + 1 \leqslant h(p-1) + g(p-1) + 1$ et donc $h(p) = h(p-1) + f(p-1)$.
On obtient : $h(p) = f(p-1)+f(p-2)+...+f(1)+h(1) = 3^{p-1}+3^{p-2}+...+1+1 = \frac{1}{2}(3^p+1)$.
On en déduit $g(p) = h(p-1) + f(p-1) - 1 = h(p) - 1 = \frac{1}{2}(3^p - 1)$. □

Cela donne l'algorithme suivant, pour n pièces non marquées :

On partage les pièces en trois ensembles A, B, C tels que $|A| = |B|$, $|A| - 1 \leqslant |C| \leqslant |A| + 1$ et on pèse A et B.
- S'il y a déséquilibre on marque les pièces $\ominus$ et $\oplus$ et on applique la trichotomie sur $A \cup B$.
- Sinon on pose $|C| = m$.
  - Si $m = 3k+2$ on partage C en 3 ensembles A', B', C' avec $|A'| = k$, $|B'| = |C'| = k+1$. On pèse alors B' et A' avec une bonne pièce.
  - Sinon on partage C en 3 ensembles A', B', C' avec $|A'| = |B'|$, $|A'| - 1 \leqslant |C'| \leqslant |A'| + 1$ et on pèse A' et B'.
    - S'il y a déséquilibre on marque les pièces $\ominus$ et $\oplus$ et on applique la trichotomie sur $A' \cup B'$.
    - Sinon on réitère sur C' le procédé appliqué à C.

$\mathcal{A}_{1,+/-}$: algorithme optimal pour $P_{1,+/-}$

Remarque : avec 3 pesées on peut traiter jusqu'à 13 pièces et, dès 5 pièces, 3 pesées sont nécessaires. En observant l'arbre de décision de cet algorithme sur 13 pièces (voir figure

9.2) on constate que si l'on a un équilibre à la première pesée on se ramène à chercher une fausse pièce parmi 5 pièces non marquées (sous-arbre encadré en pointillés). Il faut 2 pesées supplémentaires pour la trouver, alors que pour le problème à 5 pièces il faut 3 pesées (car on ne dispose pas d'une vraie pièce supplémentaire).

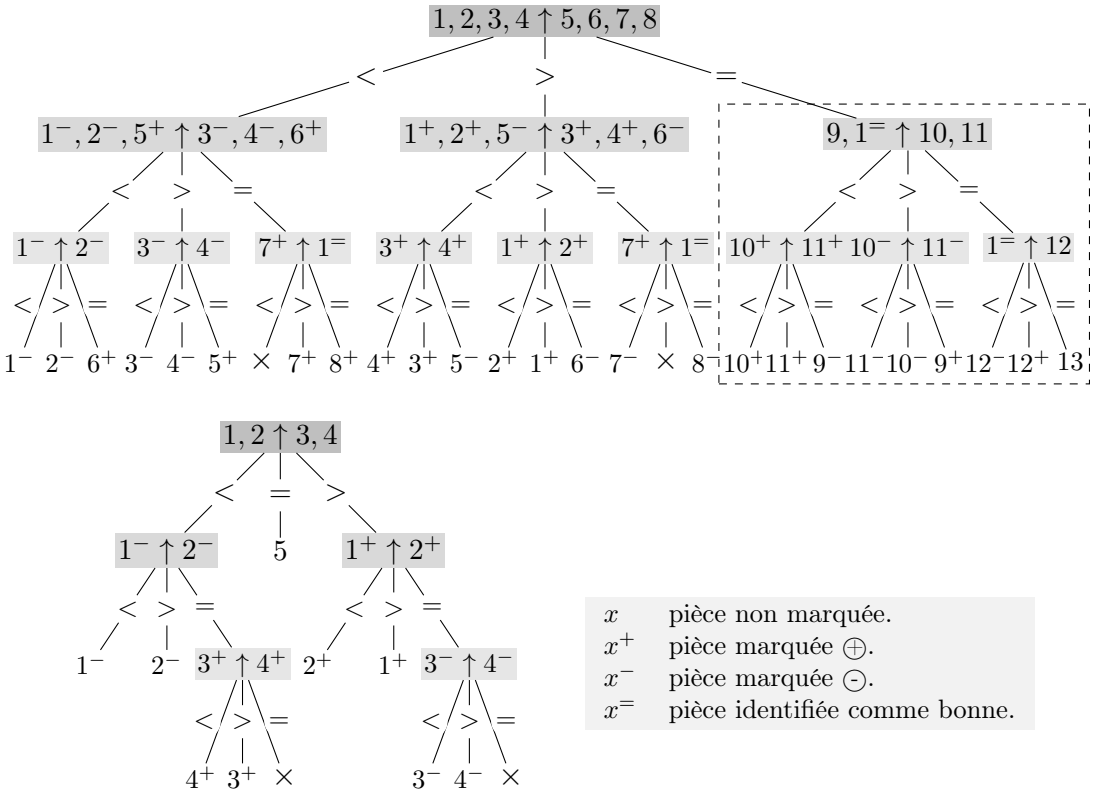


FIGURE 9.2 – L'arbre de décision de l'algorithme optimal pour $P_{1,+/-}$, pour 13 pièces et pour 5 pièces.

Sans passer par le problème dual, il est difficile de résoudre le sous-problème d'optimalité P_O. Cependant on peut tout de même obtenir des résultats partiels intéressants :

Proposition 4. *Soit $p(n)$ le nombre de pesées minimum pour traiter n pièces. On a l'encadrement suivant :*

$$\lceil \log_3(2n-1) \rceil \leqslant p(n) \leqslant \lceil \log_3(n) \rceil + 1$$

Preuve 6 : Soit $\mathcal{A}$ un algorithme pour le problème $P_{1,+/-}$. Lorsque $\mathcal{A}$ identifie une pièce comme étant la fausse, il détermine nécessairement si elle est plus lourde ou plus légère, à l'exception éventuellement de l'une des pièces dont le poids restera inconnu. Autrement dit, il existe au plus une pièce qui peut être repérée comme étant la fausse par $\mathcal{A}$ sans être passée sur la balance (on peut le voir sur la figure 9.2).
En effet, supposons le contraire et imaginons que c'est le cas pour au moins 2 pièces i et j. Si l'une de ces 2 pièces est la fausse, toute pesée effectuée par $\mathcal{A}$ donnera un équilibre car ni i ni j ne sera sur la balance. $\mathcal{A}$ ne pourra pas déterminer qui de i ou de j est la fausse, ce qui est absurde.
Sur n pièces, un algorithme pour $P_{1,+/-}$ discerne donc au moins $2n-1$ possibilités. Nécessairement il doit utiliser au moins $\lceil \log_3(2n-1) \rceil$ pesées (voir la preuve 2 de la proposition 1), ce qui donne la première inégalité.

La deuxième inégalité provient d'une majoration de la complexité de l'algorithme décrit plus haut. Si l'on note $C(n)$ la complexité au pire de l'algorithme pour n pièces on montre :

$$C(n) \leqslant 1 + \max\left(\left\lceil \log_3\left(\left\lfloor 2\frac{n}{3}\right\rfloor\right)\right\rceil, C\left(\left\lceil \frac{n}{3}\right\rceil\right)\right)$$

Soit $3^{p-1} < n \leqslant 3^p$, montrons par récurrence sur p que $C(n) \leqslant \lceil \log_3(n) \rceil + 1$.
Si $p = 1$, $n = 3$ et la propriété est vraie.
Supposons-la vraie jusqu'au rang p et soit $3^{p-1} < n \leqslant 3^p$. D'après la formule ci-dessus on a :
$C(n) \leqslant C(3^p) \leqslant 1 + \max(\lceil \log_3(2.3^{p-1}) \rceil, C(3^{p-1}))$
Donc $C(n) \leqslant 1 + \max(p, p - 1 + 1) = p + 1 = \lceil \log_3(n) \rceil + 1$. D'où le résultat. □

Remarque : $(\lceil \log_3(n) \rceil + 1) - \lceil \log_3(2n - 1) \rceil \in \{0, 1\}$, ce qui est assez satisfaisant.

Stratégies pour $P_{0/1,+}$:

On peut utiliser les stratégies S_{exp}, S_{etal} et S_{prob}.

On peut aussi utiliser des variantes des stratégies S_{tricho} et S^1_{dicho} : si l'on n'a eu que des équilibres jusqu'à la fin et qu'il restait 1 pièce à la dernière pesée, une pesée supplémentaire est nécessaire pour déterminer si la pièce non pesée est fausse ou s'il n'y avait aucune fausse pièce.

Stratégies pour $P_{0/1,+/-}$:

On peut utiliser les stratégies S_{exp}, S_{etal} et S_{prob}.

On peut adapter les stratégies S_{tricho}, S^1_{dicho}, S^2_{dicho} et S_{marq} : si l'on n'a eu que des équilibres jusqu'à la fin et qu'il restait 1 pièce à la dernière pesée, une pesée supplémentaire est nécessaire pour déterminer si la pièce non pesée est fausse ou s'il n'y avait aucune fausse pièce.

Stratégies pour P_O :

Certaines stratégies pour la preuve de l'optimalité ont déjà été présentées. On les reprend ici avec d'autres. Certaines peuvent se combiner. On distinguera si elles sont du type P^L_O ou P^G_O.

- P^L_O : Preuve de l'optimalité par un argument local sur chaque étape de la méthode.
- P^G_O : Preuve de l'optimalité par un argument global (borne inf, ...).

Parfois la construction de la méthode et la preuve de son optimalité peuvent être simultanées.

Stratégie d'énumération de tous les cas : elle s'applique à de petits nombres de pièces. L'énumération peut être faite de manière plus ou moins fine. Selon les cas on peut être de type P^L_O ou P^G_O.

Stratégie optimum local ⇒ optimum global : on prouve que chaque étape est optimale et on en déduit que l'algorithme est optimal. Elle est formellement incorrecte car

pour certains problèmes faire le mieux à une étape peut mener à une configuration qui nécessite plus d'opérations. Elle est du type P_O^L.

Stratégie par récurrence : on établit une relation de récurrence sur le nombre de pièces au départ. Par exemple, on passe de $P_{0/1,+/-}$ à $P_{1,+/-}$ en faisant le minimum de pesées puis on applique un algorithme optimal pour $P_{1,+/-}$. On conclut que cette nouvelle méthode est optimale. Elle est incorrecte pour les mêmes raisons que la précédente. L'argument est de type P_O^L.

Stratégie par borne inférieure atteinte : on montre que le nombre de pesées nécessaires dans notre méthode est un minorant du nombre de pesées nécessaires (par exemple en construisant un arbre de choix de l'algorithme) : elle est du type P_O^G.

Stratégie du sous-problème optimal : on s'est ramené à un autre problème dont on connaît une solution optimale et on en déduit l'optimalité. On peut être du type P_O^L ou P_O^G.

Stratégie de passage au dual : on cherche le plus grand nombre de pièces qu'on peut traiter avec un nombre fixé de pesées. On peut ensuite faire appel aux différentes stratégies d'optimalité ci-dessus, de type P_O^L ou P_O^G.

4.3 Un problème fondamental

Critères de $\mathscr{P}_{\text{FA}}$

Le problème des pesées répond bien aux critères que nous avons proposés pour $\mathscr{P}_{\text{FA}}$:

- Il est résoluble algorithmiquement.
- Le concept d'algorithme est indispensable à sa résolution.
- Il soulève des questions dans $\mathbb{P}$.

Ces points sont nécessairement vérifiés du fait que l'on est dans un problème d'optimisation d'algorithme.

De plus, le problème vérifie les deux points supplémentaires :

- Il met en jeu plusieurs paradigmes. Ici, AM et PA sont en jeu comme nous allons le voir plus loin.
- Il soulève de nouveaux problèmes dans $\mathscr{P}_{\text{A}}$: de nombreuses variantes du problème peuvent être étudiées, certaines s'avèrent nécessaires à la résolution du problème (par exemple la résolution de $P_{1,+/-}$ demande de résoudre $P^M_{1,+/-}$).

μ-conceptions

Le modèle de μ-conceptions permet, du point de vue du savoir savant, d'analyser un problème et son lien à l'algorithme. Cela constitue un outil efficace pour savoir si un problème est fondamental.
Dans notre cas, l'analyse mathématique du problème met en avant les conceptions AM-outil, PA-outil, AM-objet et PA-objet. Nous détaillons ici un ou deux exemples pour chacune.

AM-outil

P	$P_{1,+}$	$P_{1,+/-}$
R	Algorithmes donnés par les stratégies S^1_{dicho}, S_{tricho}, S_{etal} ou S_{prob}.	Algorithme $\mathcal{A}_{1,+/-}$.
L	Langage courant et langage mathématique, exprimant les manipulations sur les pièces et leur organisation.	
Σ	Justification que l'on retrouve la fausse pièce à tout les coups : on élimine un ensemble de pièces à chaque pesée qui ne contient pas la fausse.	La preuve 5 garantit que l'algorithme trouve toujours la fausse pièce.

PA-outil

P	$P_{1,+}$	$P^M_{1,+/-}$
R	Éléments constructifs de la preuve 3.	Éléments constructifs de la preuve 4 (deuxième partie « $\geqslant$ »).
L	Langage mathématique et opérations constructives.	
Σ	Preuve 3.	Preuve 4 (deuxième partie « $\geqslant$ »).

AM-objet

P	$P_O(P_{1,+}, \mathcal{A}_{1,+})$: $\mathcal{A}_{1,+}$ est-il optimal pour $P_{1,+}$?	$C(\mathcal{A}_{1,+/-})$: quelle es la complexité de $\mathcal{A}_{1,+/-}$?
R	Preuve 1 ou preuve 3 (opérant sur $P_{1,+}$ et $\mathcal{A}_{1,+}$).	Preuve 6 (borne supérieure).
L	Langage mathématique.	
Σ	Logique mathématique.	

AM-objet

P	$C_{min}(P_{1,+})$: Donner un minorant de la complexité de tout algorithme pour $P_{1,+}$.
R	Preuve 2, opérant sur un algorithme générique pour $P_{1,+}$.
L	Langage mathématique.
Σ	Logique mathématique.

Ces conceptions mettent bien en évidence la présence de l'algorithme-outil et de l'algorithme-objet, des deux paradigmes AM et PA et de divers problèmes de $\mathscr{P}_{\text{A}}$ et de $\mathbb{P}$. Cela soutient l'appartenance du problème des pesées à $\mathscr{P}_{\text{FA}}$.

4.4 Analyse a priori de la situation

Nous proposons une brève analyse a priori de la situation, en explicitant les différentes variables en jeu et en exposant quelques hypothèses sur le déroulement de l'activité de recherche des élèves.

Variables de la situation

On s'intéresse ici aux différentes variables de la situation, et notamment à celles des deux types suivants :

- les variables de recherche,
- les variables didactiques.

Les variables de recherche sont les variables de la situation qui sont à disposition de l'élève pour organiser son travail de recherche.

> L'élève se retrouve en effet en position de « chercheur » mais aussi en situation de « gestionnaire » de sa propre recherche : c'est lui qui choisit et modifie les valeurs des variables de recherche. (Cartier et al., 2006)
>
> Elles déterminent la compréhension et l'intérêt de la question, son ouverture à de nouvelles questions, l'élargissement des stratégies de recherche, les possibilités de transformation du problème (modélisation).(Grenier & Payan, 2003)

Les variables didactiques, parmi les variables de la situation, sont celles dont les modifications peuvent provoquer un changement de stratégie chez l'élève. D'après Brousseau :

> Seules les modifications qui affectent la hiérarchie des stratégies sont à considérer (variables pertinentes) et parmi les variables pertinentes, celles que peut manipuler un professeur sont particulièrement intéressantes : ce sont les variables didactiques. (Brousseau, 1982, p. 23)

Dans la situation étudiée, nous avons repéré les variables didactiques suivantes.

Parmi les variables du problème :

Le nombre de fausses pièces : le problème peut devenir relativement complexe. Si l'on s'autorise plus d'une fausse pièce, par exemple, le problème avec 2 fausses pièces n'est à ce jour pas complètement résolu.
Pour le problème où l'on sait qu'il y a au maximum une fausse pièce, pour n'importe quelles valeurs des autres variables, on peut savoir si le nombre de fausses pièces est 0 ou 1 en une ou deux pesées ce qui favorisera une stratégie du type S_{prob}.
Au contraire, si l'on sait qu'il y a exactement une fausse pièce, la question de rechercher une fausse pièce qui n'existe peut-être pas n'interfère plus et des stratégies comme S_{dicho} ou S_{tricho} pourront être mises en place directement.

Le poids des fausses pièces ou plutôt les informations sur leur poids : si l'on sait qu'elles sont plus lourdes (ou plus légères, c'est le même problème), ou si l'on ne le sait pas, les stratégies envisageables vont sensiblement différer.
Dans le problème $P_{1,+/-}$ par exemple, ne pas connaître le poids de la fausse pièce pousse à rechercher cette information avant de rechercher la fausse pièce. Cela renforce la stratégie S_{prob}. Ou au contraire, on peut être amené à penser que le poids de la fausse pièce ne peut être déterminé et se tourner vers une stratégie de type S_{etal}, qui n'utiliserait par exemple que l'information que le poids de la fausse pièce est différent de celui d'une vraie.
Si l'on connaît à l'avance le poids de la fausse pièce ($P_{1,+}$), la première de ces stratégies n'a pas de sens. La deuxième, bien que pouvant aboutir à une méthode de recherche acceptable, sera effacée par des stratégies moins complexes et utilisant toute l'information sur le poids de la fausse pièce (S_{dicho}, S_{tricho}, etc.).

Le nombre de pièces : lorsque l'on veut résoudre un cas particulier du problème avec un nombre de pièces fixé, ce nombre peut influencer les stratégies.
Par exemple, un petit nombre de pièces permet la stratégie S_{exp} mais dès que l'on s'interroge sur un grand nombre de pièces, ce type de stratégie devient impossible à mettre en œuvre (notamment une énumération de tous les cas devient très vite fastidieuse). Ou encore pour les problèmes $P_{1,+}$ ou $P_{0/1,+}$, un nombre pair de pièces poussera à partager les pièces en 2 tas et à toutes les peser ensemble et favorisera S_{dicho}. De même si l'on cherche sur un cas à $3k+1$ ou $3k+2$ pièces, S_{tricho} peinera à apparaître face à d'autres stratégies.

Ces trois variables, nombre de fausses pièces, poids des fausses pièces et nombre de pièces sont des **variables de recherche** de la situation. Ce sont ces variables, en particulier les deux premières, qui permettent à l'élève de manipuler les diverses variantes du problème selon son choix où selon les problèmes qui apparaissent dans la résolution (notamment avec la stratégie S_{prob}). Le passage d'un problème à un autre et les relations entre différents problèmes sont importants en algorithmique. Fixer ces variables pourrait dénaturer l'activité de recherche et l'enjeu algorithmique.

Parmi les autres variables de la situation :

L'énoncé du problème : la présentation du problème et l'ordre dans lequel sont présentées les différentes variantes peut avoir une influence sur les méthodes de résolution choisies, de même que le choix des exemples préliminaires qui peuvent être proposés.
Par exemple si l'on a cherché sur $P_{1,+/-}$ et qu'ensuite on regarde $P_{0/1,+/-}$, cela pourra favoriser une stratégie de type S_{prob}. Il en sera de même si l'on regarde le problème $P_{1,+/-}$ après avoir traité $P_{1,+}$.
S'il est proposé de traiter des cas particuliers avant le cas général, et que pour ces cas particuliers plusieurs méthodes sont optimales, ou conduisent à la même succession de pesées, cela peut entraver la mise en œuvre de certaines stratégies. Par exemple, la trichotomie et le découpage qui laisse 2 ou 3 pièces dans le troisième tas selon la parité du nombre total de pièces, donnent les mêmes découpages pour 4, 5, 6, 7, 8 et 9 pièces.

Le matériel disponible : la présence d'une balance Roberval et de « pièces » peut avoir une influence sur la dévolution de l'énoncé du problème et sur la méthodologie de recherche. Cela peut favoriser les stratégies de type S_{exp}, mais cela peut aussi entraver le passage à des grands nombres de pièces et créer des difficultés à énumérer tous les cas possibles pour un algorithme donné.

Stratégies attendues

Il nous semble que la stratégie la plus courante pour répondre au problème $P_{1,+}$ sera S_{dicho}. Une bonne méthode pour la rejeter comme stratégie optimale peut être la stratégie S_{exp} qui par exemple pour 9 pièces permet de voir qu'une solution à 2 pesées existe. Alors d'autres stratégies, comme S_{tricho}, pourront apparaître.
Nous faisons l'hypothèse que l'optimalité de l'algorithme de trichotomie ne va pas apparaître facilement et que si c'est le cas ce sera par un argument d'optimalité locale.

Concernant les autres variantes du problème, les stratégies principales seront, selon nous, de se ramener au problème $P_{1,+}$. La hiérarchie des stratégies employées sera alors la même que pour $P_{1,+}$. Les autres stratégies identifiées dans l'analyse mathématique seront, pour nous, minoritaires.

L'optimalité des méthodes proposées risque de rester souvent sans preuve. Une preuve qui pourra émerger sera celle, erronée, qui consiste à déduire l'optimalité de celle obtenue pour $P_{1,+}$. C'est encore un argument d'optimalité locale.

Le calcul de la complexité des algorithmes proposés sera, d'après nous, souvent interrogé mais peut constituer un point délicat.

Il nous semble qu'un temps de recherche long est nécessaire pour dépasser les obstacles ou les difficultés que nous venons de mentionner. Ces obstacles sont notamment liés au passage de problèmes de $\mathscr{P}_{\text{A}}$ à des problèmes de $\mathbb{P}$, caractéristique de la dialectique outil-objet, et au passage d'arguments d'optimalité locale à des arguments d'optimalité globale.

En résumé nous émettons les hypothèses suivantes :

- H_1 : pour $P_{1,+}$, la stratégie privilégiée est S_{dicho} ; les stratégies S_{exp} et S_{tricho} viennent plus tard.
- H_2 : l'optimalité de S_{tricho} est délicate et sera majoritairement abordée avec un argument d'optimalité locale.
- H_3 : pour les autres variantes, la stratégie privilégiée est S_{prob} pour revenir à $P_{1,+}$.
- H_4 : l'optimalité dans les variantes aura du mal à émerger ; lorsqu'elle apparaîtra, elle sera déduite de celle de $P_{1,+}$.
- H_5 : un temps long est nécessaire à l'apparition de problèmes de $\mathbb{P}$ et à l'émergence d'arguments d'optimalité globale.

Pré-expérimentations

Des pré-expérimentations de cette situation ont déjà été menées auprès de deux publics différents sur des temps courts (une heure à une heure et demi).

Elles confirment cependant que l'entrée dans le problème est facile : tous les groupes ont construit des algorithmes et ont prouvé qu'ils permettaient bien de retrouver la fausse pièce. De plus, les questions de $\mathscr{P}_{\text{A}}$ et $\mathbb{P}$ sont soulevées (complexité et optimalité) et l'enjeu de preuve est présent. Cela nous permet déjà d'affirmer que ces problématiques peuvent être dévolues au travers de cette situation.

Ces pré-expérimentations valident aussi les hypothèses de l'analyse a priori. Les stratégies privilégiées sont celles attendues et l'optimalité n'est abordée que par l'argument local.
Il ressort aussi que le rapport de l'élève à la preuve a une influence sur la dévolution des problématiques de la situation et les stratégies adoptées.

Il est clair qu'un temps plus long sera nécessaire à faire émerger des stratégies plus avancées.

Les résultats des analyses des pré-expérimentations, développés dans un travail précédent (mémoire de Master 2) sont présentés en annexe D. On y trouve des précisions sur les stratégies abordées qui montrent que le potentiel de la situation est réel pour l'algorithmique, pour l'activité de recherche et pour la preuve.

Des outils pour permettre l'analyse de futures expérimentations en classe sont en développement. En particulier, étant donné le type d'activité attendu, il est nécessaire de proposer des outils permettant d'étudier finement l'activité de preuve et de résolution de problèmes.

Conclusion et perspectives

Bilan

Le travail épistémologique de la première partie a permis de proposer des critères pour caractériser les problèmes propres à fournir des situations didactiques. Nous les avons appelés problèmes fondamentaux pour l'algorithme.

Cela nous a permis de repérer un ensemble de problèmes qui sont de bons candidats pour cela. Nous avons étudié en détail l'un de ces problèmes : le problème des pesées.Nous avons montré en quoi le modèle de μ-conceptions permet d'analyser un problème et d'étudier son potentiel. L'analyse montre que le problème des pesées peut être un problème fondamental pour l'algorithme.

Le problème des pesées fournit une situation riche pour l'algorithme. Une SiRC est adaptée pour proposer ce problème et mettre en jeu l'algorithme dans une activité de recherche de l'élève. Les résultats d'expérimentations courtes de ce problème nous poussent à poursuivre l'étude, en expérimentant ce problème sur un temps plus long en classe.

Perspectives

Ce travail est encore en chantier et les poursuites et perspectives sont nombreuses.

Nous l'avons dit, un de nos objectifs à court terme est l'expérimentation de la situation des pesées dans des classes. Nous souhaitons aussi poursuivre le développement d'autres situations à partir des problèmes proposés.

L'objectif à long terme est de proposer une situation ou un ensemble de situations fondamentales pour l'algorithmique. La situation des pesées tends à faire partie de telles situations, avec ses variantes ou d'autres situations. La construction d'une telle situation sera soutenue par le cadre théorique construit dans les premiers chapitres.

Du côté théorique, l'utilisation du modèle de μ-conceptions pour l'étude des problèmes et des situations pour l'algorithmique est à poursuivre. Le concept-problème (Giroud, 2011) nous semble être une piste sérieuse pour compléter cette approche et permettre de mieux appréhender la dialectique outil-objet dans l'activité algorithmique et dans d'autres activités.

Conclusions des recherches Perspectives et nouvelles questions

La problématique de l'enseignement de l'algorithme, avec l'introduction de l'algorithmique au lycée, mais avant tout pour son lien étroit à la preuve, nous a amené à soulever plusieurs questions à l'origine de cette thèse. Ces questions concernent l'épistémologie de l'algorithme, la transposition de l'algorithmique au lycée, et la construction de situations.

Nous allons revenir point par point sur ces questions et sur les réponses que nous leur avons apportées. Étudier ces questions nous a demandé d'introduire des outils théoriques ou d'en adapter d'autres. L'utilisation de certains de ces outils peut aussi constituer un résultat sur lequel nous reviendrons.

Nous détaillerons ensuite quelles nouvelles questions sont soulevées et quelles perspectives peuvent être envisagées.

1 Résultats

1.1 Épistémologie de l'algorithme

Un premier modèle

Peu d'études épistémologiques sur l'algorithme existent, en particulier dans une perspective d'enseignement. En nous appuyant sur le texte du savoir savant, nous avons d'abord étudié la question :

Question Q1. *Quels sont les éléments constitutifs spécifiques du concept d'algorithme ? Quels sont l'objet et les préoccupations fondamentales de la discipline algorithmique ? Quel lien entretiennent algorithme et preuve ?*

En réponse, nous avons proposé un premier modèle qui permet de mettre en valeur les aspects fondamentaux du concept. Cela nous a permis de dégager les cinq ASPECTS suivants que nous avons classifiés selon une dualité outil-objet :

Outil	**Objet**
PROBLÈME EFFECTIVITÉ	PREUVE COMPLEXITÉ MODÈLES THÉORIQUES

Grâce à cela, nous avons pu, en particulier, détailler le lien entretenu par algorithme et preuve.

Pensée algorithmique

Une autre question fondamentale pour l'algorithmique est son lien aux mathématique et à l'informatique. Nous nous sommes posé la question de savoir si une pensée spécifique était en jeu en algorithmique :

Question Q2. *Peut-on parler d'activité ou de pensée algorithmique dans les mathématiques ? Et dans l'informatique ? Quelles perspectives cela peut-il ouvrir pour une approche didactique ?*

En nous appuyant sur les recherches sur l'*advanced mathematical thinking* nous avons proposé une étude de la *pensée* algorithmique. Nous avons présenté ses particularités et montré l'importance de ne pas la considérer uniquement comme une pensée mathématique.

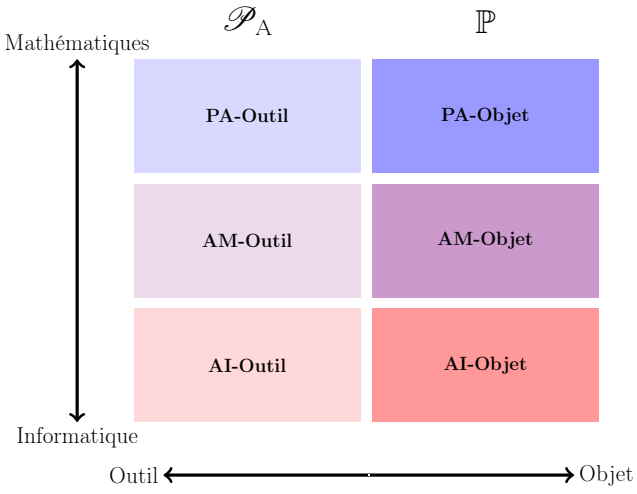
Un modèle de conception

Les limites du modèle par ASPECTS et les éléments mis en avant dans l'analyse de la *pensée algorithmique* nous ont amené à concevoir un nouveau modèle pour répondre à la question :

Question Q3. *Peut-on construire un modèle de l'activité algorithmique permettant de questionner la transposition didactique ?*

Le modèle obtenu se base sur le modèle théorique cK¢ et sur une définition spécifique de la notion de problème comme couple *(Instances, Question)*. Cela nous a permis de décrire des conceptions pour l'algorithme, du point de vue du savoir-savant : des μ-conceptions.

Ces μ-conceptions s'articulent selon différents paradigmes dans lesquels l'algorithme peut être exprimé (Preuve Algorithmique, Algorithme Mathématique et Algorithme Informatique) et deux familles de problèmes ($\mathscr{P}_{\text{A}}$ et $\mathbb{P}$) permettant de mettre en jeu une dualité outil-objet.



Le choix de décrire des conceptions dans ce modèle permet de l'utiliser en tant qu'outil pour étudier le rapport au concept algorithme. En particulier cela répond à l'objectif d'étudier les transpositions didactiques.

1.2 Conceptions de chercheurs

Nous avons confronté nos résultats épistémologiques avec les discours de chercheurs, afin de les éprouver et de les légitimer. Proposant des modèles étendus, il nous a semblé bon de les

valider expérimentalement par des entretiens avec des chercheurs en mathématiques mais aussi en informatique. Cela nous a permis de répondre à la première partie des questions issues de Q4 :

Question $\mathbf{Q4_{aspects}}$ - $\mathbf{Q4_{conceptions}}$. *Nos modèles d'*ASPECTS *et de μ-conceptions de l'algorithme s'accordent-ils avec la perception qu'ont les chercheurs ? Le modèle peut-il nous informer sur les variations que l'on pourrait entrevoir dans les discours de chercheurs ?*

La deuxième partie de la question a été l'occasion de tester la capacité du modèle à décrire des conceptions individuelles. Nous avons mis en évidence différents rapports à l'algorithme chez les chercheurs. La discipline de recherche semble intervenir dans la conception de l'algorithme. Cela peut avoir une influence sur la façon dont le concept est transposé.

1.3 Transposition au lycée

Ce modèle nous a ensuite permis de développer plusieurs analyses de la transposition didactique au lycée. En proposant un outil adaptable à diverses ressources nous avons étudié la transposition du savoir à enseigner au travers des instructions officielles et de manuels. De manière complémentaire, nous avons réalisé une étude de ressources en ligne, qui peuvent influencer le savoir enseigné et qui peuvent être révélatrices de certaines conceptions.

Instructions officielles

Il s'agissait ici de répondre aux questions dérivées de Q5 :

Question $\mathbf{Q5_{aspects}}$ - $\mathbf{Q5_{conceptions}}$. *Quels* ASPECTS *de l'algorithme et quelles* conceptions *sont présents dans les programmes de mathématiques et documents ressources du lycée ? Les* ASPECTS *et* conceptions *outil et objet sont-ils représentés ?*

L'étude des programmes de mathématiques du lycée et du document ressource pour la seconde révèlent une conception de l'algorithme assez éloignée de celle du savoir-savant.

En particulier, nous avons montré que l'algorithme n'est pas considéré en tant qu'objet dans ces documents. Il est réduit à un rôle de mise en œuvre des procédures rencontrées dans les différents chapitres (EFFECTIVITÉ). La notion de problème, avec différentes instances à résoudre, n'est pas toujours mise en jeu (ALGORITHMES INSTANCIÉS) et les algorithmes proposés ne sont pas toujours représentatifs du concept algorithme (PROGRAMMES DE MODÉLISATION-SIMULATION par exemple). De plus, l'algorithme se voit attribuer une écriture extrêmement formelle mettant en jeu des élément de programmation (PROGRAMME-PAPIER). Cela révèle une conception de l'algorithme comme étant orienté vers l'écriture de programmes (AI-outil).

Afin d'étudier une autre transposition de l'algorithme, nous avons analysé les programmes de la spécialité *Informatique et Sciences du Numérique* de T^e S qui comprend une forte composante d'algorithmique. Ces programmes montrent une conception de l'algorithme en tant qu'outil mais aussi en tant qu'objet, notamment en mettant en jeu les questions de complexité. Bien que fortement tourné vers la programmation (paradigme AI) les algorithmes, ici, sont très nettement différenciés des programmes (paradigme AM).

Manuels

Les mêmes questions se posent concernant les manuels :

Question Q6$_{\text{aspects}}$ - Q6$_{\text{conceptions}}$. *Quels* ASPECTS *et quelles* conceptions *de l'algorithme sont présents dans les manuels de mathématiques ? Les* ASPECTS *et* conceptions *outil et objet sont-ils représentés ?*

Les outils dérivés de nos modèles ont pu être adaptés à une étude quantitative des exercices et algorithmes présents dans les manuels. On peut constater que les manuels se placent dans une continuité totale des instructions officielles et les conclusions que nous avons faites pour les programmes sont les mêmes ici [9].

En particulier, les questions majoritaires dans les exercices de ces manuels font état d'une activité algorithmique essentiellement langagière, centrée sur l'interprétation, l'écriture et la traduction.

Ressources en ligne

Le corpus que nous avons choisi d'étudier est celui des ressources en algorithmique proposées par le site des IREM. Les questions étaient les suivantes :

Question Q7$_{\text{aspects}}$-Q7$_{\text{conceptions}}$. *Quels* ASPECTS *et quelles* conceptions *de l'algorithme sont présents dans les ressources en ligne ? Les* ASPECTS *et* conceptions *outil et objet sont-ils représentés ? Y-a-t-il une différence entre les documents de formation et ceux pour la classe ?*

La diversité de formes et de contenus des ressources proposées n'a pas constitué une difficulté pour appliquer nos outils d'analyse. Nous avons alors pu mettre en lumière de grandes variations de conceptions dans les ressources.

Les ressources pour la classe ne mettent jamais en jeu l'algorithme en tant qu'objet. En ce sens on retrouve les mêmes types d'algorithmes que dans le document ressource de seconde. Notamment le lien à la programmation est très présent dans ces ressources (AI-outil et PROGRAMMES-PAPIER).

Pour rencontrer l'algorithme en tant qu'objet et d'autres paradigmes, il faut se tourner vers les ressources à destination des enseignants. On peut alors trouver des algorithmes plus riches, pouvant être questionnés en tant qu'objets (validité, complexité, comparaison, etc.). Les algorithmes présentés n'ont pas nécessairement pour objectif d'être programmés et leur expression est moins formalisée, utilisant les éléments du langage mathématique et du langage courant (paradigme AM).

L'analyse de ces ressources nous a permis d'avancer des hypothèses sur les contraintes qui accompagnent l'introduction du concept algorithme au lycée et les conditions permettant son développement.

1.4 Situations pour la classe

L'analyse épistémologique apporte aussi des éléments pour la construction de situations didactiques pour l'algorithme (la construction de situations est d'ailleurs à l'origine du développement de ce modèle). Nous avions formulé les questions suivantes :

Question Q8. *Comment caractériser les problèmes à fort potentiel pour l'algorithmique à l'aide du modèle épistémologique développé ? Quels critères doivent-il vérifier ? Quels problèmes répondant à ces critères peut-on proposer ?*

9. À l'exception de la spécialité mathématique de T$^{\text{e}}$ S. Voir le chapitre concerné.

Question Q9. *Quels éléments épistémologiques sont à prendre en compte pour construire des situations didactiques autour de l'algorithme ? Quelles situations peut-on proposer ?*

En nous appuyant sur les liens privilégiés qu'entretient l'algorithme à la notion de problème, nous avons défendu la construction de situations pour l'algorithme basées sur la résolution de problème et l'activité de recherche. Pour cela nous avons introduit la notion de *problème fondamental pour l'algorithme* et précisé des caractéristiques de ces problèmes. Nous avons proposé une liste de problèmes qui devraient permettre de mettre en jeu l'algorithme dans une situation et de le questionner en tant qu'objet.

Un problème, le *problème des pesées*, a été étudié de manière détaillée et une situation précise en a été tirée. Des pré-expérimentations de cette situation ont été menées et confirment son potentiel. Cette partie de la recherche est encore en développement, nous reviendrons sur cette question dans la section *perspectives*.

Enfin, le lien fort entre preuve et algorithme nous laisse penser qu'une situation didactique pour l'algorithme doit mettre en jeu l'activité de preuve. Les Situations de Recherche en Classe sont un moyen pertinent pour atteindre un tel objectif.

1.5 Résultats théoriques

Quelques points théoriques ou méthodologiques développés dans ce travail méritent d'être soulignés.

μ-conceptions

La description de conceptions du point de vue du savoir savant, telle que proposée par Balacheff dans l'idée de μ-conception, a été ici mise en œuvre. Le modèle cK$\not$ nous a permis de décrire plusieurs conceptions dans le savoir savant et leurs relations.

Cela permet de proposer une analyse "descendante" des conceptions individuelles ou institutionnelles, au sens où non avons utilisé les μ-conceptions comme outil d'analyse. Les μ-conceptions (par définition) permettent de porter un regard sur toute autre conception de l'algorithme.

Problème

La notion de problème, reprise de Giroud, s'est avérée très adaptée à l'étude des conceptions de l'algorithme et en particulier à celle des μ-conceptions, où la notion de problème comme *perturbation du système sujet-milieu* n'est plus adaptée. Il nous semble qu'une telle définition de problème peut être réutilisée pour décrire les μ-conceptions d'autres concepts.

Les classes $\mathscr{P}_{\text{A}}$ et $\mathbb{P}$ issues de cette définition de problème, couplées à la notion de conception, nous ont permis de décrire la dualité outil-objet du concept. Cette dualité prend alors sens par un "décalage" où les opérateurs deviennent objet des problèmes et les structures de contrôle deviennent opérateurs.

Transposition

Enfin, l'utilisation du modèle cK$\not$ pour l'étude de la transposition didactique est suffisamment peu courante pour que nous soulignions son utilisation ici. En particulier, dans

l'analyse des ressources et des manuels, le modèle de conception s'est avéré assez souple d'utilisation et a permis de tenir compte de l'hétérogénéité des corpus.

2 Perspectives

2.1 Situations pour l'algorithme

La principale perspective de ce travail est le développement de situations didactiques pour l'algorithme.

L'étude de la situation des pesées doit être poursuivie par son expérimentation dans un cadre scolaire. Les autres problèmes proposés pourraient donner lieu à des situations. Il convient d'analyser en détail leur potentiel afin de rechercher ceux qui pourrait mettre en jeu des éléments complémentaires à ceux présents dans le problème des pesées.

Un objectif pourrait être de proposer un ensemble de situations fondamentales pour l'algorithme.

Certaines situations déjà développées et étudiées parmi les SiRC mettent en jeu l'algorithme. Il nous semble pertinent d'étudier ces situations sous cet angle nouveau et de voir en quoi elles peuvent contribuer à l'apprentissage de l'algorithmique.

Enfin, le lien entre algorithme et preuve est fort et nous pensons que l'algorithmique peut être un moyen de mettre en jeu la preuve dans la classe de mathématiques. Il nous semble que des situations dans ce but peuvent aussi être développées.

2.2 Contraintes et conditions

L'algorithme ne vit pas en tant qu'objet dans l'enseignement du lycée en France. Nous l'avons montré au travers de l'étude de sa transposition. Les analyses réalisées donnent des pistes sur les raisons de cette absence.

Certaines contraintes pèsent sur la transposition et sur l'enseignement, et le fait que l'algorithme soit un objet mal connu (des enseignants mais aussi des mathématiciens) a une influence. De même, la demande de l'usage des TICE joue un rôle dans la domination du paradigme AI. Cependant, il nous semble que deux pistes peuvent être formulées en termes de conditions d'existence de l'algorithme objet.

La première est que l'algorithme est un objet qui apparaît beaucoup plus dans certains champs des mathématiques et pour répondre à certains types de questions. Pour que l'algorithme vive pleinement, il faut que ces champs et ces questions soient présents. Cela relève de l'écologie des savoirs. Comprendre quelles sont les niches et habitats de l'algorithme permettrait de connaître les conditions favorables à son enseignement. Notre travail est déjà en lien avec cette problématique puisque nous avons soulevé la question du rôle de l'algorithme et formulé des hypothèses concernant les champs privilégiés [10].

L'autre point de vue est de s'intéresser au rôle donné à l'algorithme, non pas au rôle mathématique mais à son rôle pour l'enseignement. Il nous semble que dans la transposition

10. Nous avons évoqué ces champs lors de l'étude des conceptions des chercheurs et de l'analyse de la transposition, en particulier de l'analyse des ressources IREM.

didactique en jeu au lycée, l'algorithme a plus un rôle "didactique" que mathématique. Ce rôle didactique peut être un obstacle à l'existence de l'algorithme en tant qu'objet mathématique : l'algorithme est "outil d'enseignement".

Comprendre ces phénomènes peut permettre de préciser l'analyse de la transposition en jeu dans l'enseignement du lycée.

2.3 Conceptions sur l'algorithme

L'analyse de la transposition, via les conceptions des institutions, nous a amené a construire des outils spécifiques. Nous souhaitons poursuivre l'utilisation et le développement de ces outils afin d'étudier d'autres transpositions. Les phénomènes de transpositions sont parfois difficiles à observer mais il nous semble que l'introduction d'un nouveau concept tel que l'algorithme est l'occasion de les étudier en détail.

Nous n'avons étudié la transposition au lycée qu'à travers les instructions et des ressources pour l'enseignant. L'étape suivante pour l'étude du savoir enseigné est de faire des observations en classe. En particulier, il serait intéressant de pouvoir observer des enseignants ayant des conceptions différentes lors de séances sur l'algorithmique et de voir comment elles influencent leurs pratiques.

L'étude de la transposition du concept algorithme dans des enseignements d'informatique (notamment dans la classe d'ISN) nous paraît un bon moyen de comprendre les phénomènes de transposition d'un même concept dans deux disciplines différentes. L'étude des choix de transposition en place dans d'autres enseignements de mathématiques (en France ou à l'étranger) ou dans des enseignements de mathématiques-informatique nous semble être une piste d'intérêt.

Enfin, nous avons pu étudier les conceptions des chercheurs individuellement en nous appuyant sur des entretiens. La méthodologie utilisée est à systématiser si l'on veut construire des outils d'analyse des conceptions plus efficaces. Cela permettrait de poursuivre l'analyse des conceptions des chercheurs à plus grande échelle mais surtout d'étudier celles des enseignants, voire celles des élèves (qui peuvent être révélatrices du savoir transposé).

2.4 Sémiotique

Le modèle de paradigmes et de μ-conceptions, au travers des systèmes de représentation, nous a permis de tenir compte des diverses formes que peuvent prendre les algorithmes. La notion de conception du modèle cK¢, grâce à aux systèmes de représentation Σ permet de prendre en charge des aspects sémiotiques avec une certaine souplesse et en interaction avec les autres composantes de la conception (problèmes, opérateur, contrôles).

Au vu de l'analyse épistémologique, il nous semble important de développer l'étude des questions sémiotiques en lien avec nos modèles. Le cadre des conceptions nous paraît donc adapté pour développer des éléments de sémiotique. Balacheff et Margolinas (2005) propose l'utilisation des registres sémiotiques de Duval, pour décrire les systèmes de représentation. Cela nous semble être une piste à explorer.

2.5 Modèles et épistémologie

L'analyse épistémologique du concept peut être encore affinée. Il nous semble que l'étude et la recherche de *problèmes fondamentaux pour l'algorithme* peut participer au développement du modèle. Les expérimentations de situations en classe sont aussi un moyen de questionner le modèle et de le faire évoluer.

La notion de problème introduite ici s'inspire de la notion utilisée par Giroud pour décrire le concept-problème. Cette définition d'un problème permet de décrire la dualité outil-objet de l'algorithme. Nous pensons qu'il est possible de modéliser la dualité outil-objet de cette manière pour d'autres concepts. Cela pourrait aussi permettre de caractériser la dialectique outil-objet dans l'activité.

Enfin, l'utilisation des μ-conceptions pour construire un modèle épistémologique et développer un outil d'analyse de la transposition pour d'autres concepts nous semble être un point à questionner.

Références

Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1989). *Structures de données et algorithmes*. Paris : Inter Éd, D.L. (Trad. de : Data structures and algorithms)

Aigner, M., & Li, A. (1997). Searching for counterfeit coins. *Graphs and Combinatorics*, *13*(1), 9-20.

Artigue, M. (1990). Épistémologie et Didactique. *Recherches en Didactique des Mathématiques*, *10*(2.3), 241-285.

Balacheff, N., & Margolinas, C. (2005). cK¢ Modèle de connaissances pour le calcul de situations didactiques. In A. Mercier & C. Margolinas (Eds.), *Balises pour la didactique des mathématiques* (p. 1-32). Grenoble : La Pensée Sauvage.

Beauquier, D., Berstel, J., & Chrétienne, P. (1992). *Éléments d'algorithmique*. Masson.

Bouvier, A., George, M., & Le Lionnais, F. (2005). *Dictionnaire des mathématiques*. Puf.

Brousseau, G. (1982). Les Objets de la Didactique des Mathématiques. In *Contributions à la seconde école d'été de didactique des mathématiques* (p. 10-33). IREM d'Orléans.

Brousseau, G. (1998). *Théorie des Situations Didactiques*. La Pensée Sauvage.

Brousseau, G. (2010). *Glossaire de quelques concepts de la théorie des situations didactiques en mathématiques*. Disponible sur `http://guy-brousseau.com/wp-content/uploads/2010/09/Glossaire_V5.pdf`

Burton, L. (2004). *Mathematicians as inquirers. Learning about learning mathematics*. Kluwer Academic Publishers.

Cartier, L. (2008). *Le graphe comme outil pour enseigner la preuve et la modélisation*. Thèse de doctorat non publiée, Université Joseph Fourier, Grenoble. Disponible sur `http://tel.archives-ouvertes.fr/`

Cartier, L., Godot, K., Knoll, E., & Ouvrier-Buffet, C. (2006). Les Situations-Recherche : Apprendre à chercher en mathématiques. In *Colloque EMF et AMQ*. Canada : Université de Sherbrooke.

Chabert, J.-L. (2010). *Histoire d'algorithmes - du caillou à la puce* (2e éd.). Belin.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Cazin, X. (1994). *Introduction à l'algorithmique*. Paris : Dunod. Disponible sur `http://opac.inria.fr/record=b1077377` (Trad. de : Introduction to algorithms)

Douady, R. (1986). Jeux de cadres et dialectique outil-objet. *Recherches en Didactique des Mathématiques*, *7*(2), 5-31.

Dreyfus, T. (1991). Advanced Mathematical Thinking Processes. In D. O. Tall (Ed.), *Advanced Mathematical Thinking* (p. 25-41). Holland : Kluwer.

Fleischner, H. (1991). *Eulerian Graphs and Related Topics* (Vol. 1.2) (No n° 50). Elsevier Science.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Giroud, N. (2011). *Étude de la démarche expérimentale dans les situations de recherche pour la classe*. Thèse de doctorat non publiée, Université de Grenoble. Disponible

sur http://tel.archives-ouvertes.fr

Godot, K., & Grenier, D. (2004). Research Situations for teaching : a modelization proposal and examples. In *Proceedings of ICME 10.* Roskilde University : IMFUFA.

Grenier, D., & Payan, C. (2003). Situations de recherche en « classe », essai de caractérisation et proposition de modélisation. *Les cahiers du laboratoire Leibniz*, *92*.

Gueudet, G., & Trouche, L. (2010). *Ressources vives. Le travail documentaire des professeurs de mathématiques* (INRP, Ed.). Presses Universitaires de Rennes.

Harel, G., & Sowder, L. (2005). Advanced Mathematical-Thinking at Any Age : Its Nature and Its Development. *Mathematical Thinking and Learning*, *7*, 27-50.

Hart, E. W. (1998). Algorithmic Problem Solving in Discrete Mathematics. In L. J. Morrow & M. J. Kenney (Eds.), *The teaching and learning of Algorithm in school mathematics, 1998 NCTM Yearbook* (p. 251-267). Reston, VA : National Council of Teachers of Mathematics.

Kahane, J.-P. (2002). *Enseignement des sciences mathématiques : Commission de réflexion sur l'enseignement des mathématiques : Rapport au ministre de l'éducation nationale* (O. Jacob, Ed.). Paris : CNDP. Disponible sur http://smf.emath.fr/Enseignement/CommissionKahane/

Knuth, D. E. (1973). *The Art of Computer Programming, Volume I : Fundamental Algorithms* (2e éd.). Addison-Wesley.

Knuth, D. E. (1974). Computer Science and its Relation to Mathematics. *The American Mathematical Monthly*, *81*(4), 323-343. (Rééd. avec corrections (1996) In *Selected Papers on Computer Science*, *5-29*)

Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly*, *92*(1), 170-181. (Rééd. avec corrections (1996) Algorithms in Modern Mathematics and Computer Science. In *Selected Papers on Computer Science*, *87-114*)

Knuth, D. E. (1996). *Selected Papers on Computer Science.* Center for the Study of Language and Inf.

Lovász, L. (1988). Algorithmic mathematics : an old aspect with a new emphasis. In A. Hirst & K. Hirst (Eds.), *Proceedings of ICME 6* (p. 67-78). Budapest : J. Bolyai Mathematical Society.

Lovász, L. (2007). Trends in Mathematics : How they could Change Education ? In *Conférence européenne "The Future of Mathematics Education in Europe".* Lisbonne. Disponible sur http://www.cs.elte.hu/~lovasz/lisbon.pdf

Lovász, L., & Plummer, D. (2009). *Matching Theory.* AMS Chelsea Pub.

Maurer, S. B. (1998). What is an algorithm ? what is an answer. In L. J. Morrow & M. J. Kenney (Eds.), *The teaching and learning of Algorithm in school mathematics, 1998 NCTM Yearbook* (p. 21-31). Reston, VA : National Council of Teachers of Mathematics.

Mingus, T. T. Y., & Grassl, R. M. (1998). Algorithmic and Recursive Thinking - Current Beliefs and Their Implications for the Future. In L. J. Morrow & M. J. Kenney (Eds.), *The teaching and learning of Algorithm in school mathematics, 1998 NCTM Yearbook* (p. 32-43). Reston, VA : National Council of Teachers of Mathematics.

Mithalal, J. (2010). *Déconstruction instrumentale et déconstruction dimensionnelle dans le contexte de la géométrie dynamique tridimensionnelle.* Thèse de doctorat non publiée, Université de Grenoble. Disponible sur http://tel.archives-ouvertes.fr

Morrow, L. J., & Kenney, M. J. (Eds.). (1998). *The Teaching and Learning of Algorithms in School Mathematics.* NCTM. (Yearbook of the National Council of Teachers of Mathematics)

Ouvrier-Buffet, C. (2009). Mathématiques Discrètes : un champ d'expérimentation mais aussi un champ des mathématiques. In ARDM (Ed.), *Actes du séminaire national de didactique des mathématiques* (p. 31-45). Université Paris 7.

Pyber, L. (1986). How to find many counterfeit coins ? *Graphs and Combinatorics*, *2*(1), 173-177.

Pólya, G. (1945). *How to Solve It : A New Aspect of Mathematical Method*. Princeton, NJ : Princeton University Press.

Rasmussen, C., Zandieh, M., King, K., & Teppo, A. (2005). Advancing mathematical activity : A view of advanced mathematical thinking. *Mathematical Thinking and Learning*, *7*, 51-73.

Robert, A. (1992). Problèmes méthodologiques en didactique des mathématiques. *Recherches en Didactique des Mathématiques*, *12*(1), 33-58.

Schuster, A. (2004). About Traveling Salesmen and Telephone networks - Combinatorial Optimization Problems at High School. *ZDM*, *36*(2), 78-81.

Sedgewick, R. (1988). *Algorithms*. Addison Wesley.

Tall, D. O. (1991). The Psychology of Advanced Mathematical Thinking. In D. O. Tall (Ed.), *Advanced Mathematical Thinking* (p. 3-21). Holland : Kluwer.

Tosic, R. (1983). Two counterfeit coins. *Discrete Mathematics*, *46*(3), 295-298.

Vergnaud, G. (1990). La théorie des champs conceptuels. *Recherches en Didactique des Mathématiques*, *10*(2-3), 133-170.

Wilf, H. S. (1982). What is an answer ? *The American Mathematical Monthly*, *89*(5), 289-292.

Wilf, H. S. (2005). Mathematics : an experimental science. In W. T. Gowers (Ed.), *Princeton Companion to Mathematics.* Princeton University Press. Disponible sur `http://www.math.upenn.edu/~wilf/reprints.html`

Wolper, P. (2006). *Introduction à la calculabilité : cours et exercices corrigés*. Paris : Dunod.

Glossaire

Ce glossaire reprend des définitions générales liées à l'algorithmique ou aux théories didactiques utilisées. Il contient aussi les termes définis spécifiquement pour ce travail de thèse. Les pages citées correspondent à des références importantes aux concepts dans la discussion.

A

algorithme Procédure de résolution de problème, s'appliquant à une famille d'instances du problème et produisant, en un nombre fini d'étapes constructives, effectives, non-ambigües et organisées, la réponse au problème pour toute instance de cette famille. 22, 25, 58

algorithme informatique (AI) Dans notre modèle de μ-conceptions, c'est l'un de trois paradigmes pour l'algorithme. Il correspond à une activité de type Problème-Programme-Vérification. L'algorithme est ici exprimé dans un langage de programmation et décrit des manipulations formelles. 61, 63

algorithme mathématique (AM) Dans notre modèle de μ-conceptions, c'est l'un de trois paradigmes pour l'algorithme. Il correspond à une activité de type Problème-Algorithme-Preuve. L'expression de l'algorithme est dissociée de sa preuve et sa forme s'appuie sur un mélange de langage de langage mathématique et d'élément spécifique à l'algorithme et parfois issus des langages de programmation. 61, 63

algorithme-instancié Un algorithme-instancié est une représentation d'un algorithme, non pas générique et pour toute instance, mais au contraire exprimée sur une instance précise. Ce n'est pas un algorithme, au-sens où il n'exprime pas une méthode générique de résolution de problème. 141

algorithmique Discipline, à l'intersection entre mathématique et informatique, dont l'objet d'étude sont les algorithmes. L'algorithmique s'intéresse notamment à la création, la description et à l'analyse des algorithmes ainsi qu'à l'étude des problèmes algorithmiquement résolubles. 26, 29

aspect Les aspects, ou aspects fondamentaux de l'algorithme sont les éléments descriptifs du premier modèle épistémologique de l'algorithme proposé dans la thèse. 40

C

Caml Caml est un langage de programmation développé par l'INRIA depuis 1985. C'est un langage fortement typé qui permet différents types de programmation (fonctionnelle, impérative...). 31

cK¢ Cadre didactique développé par Nicolas Balacheff pour modéliser les relations entre les notions de conception (c), connaissance (K) et concept (¢) en didactique des mathématiques. 55, 59, 66, 69

complexité La complexité d'un algorithme est l'étude de son comportement en fonction de l'instance traitée. Plus précisément, la complexité s'exprime comme une fonction de la taille de l'instance et décrit le nombre d'étapes nécessaires à la résolution (complexité en temps) ou la quantité d'information à mémoriser (complexité en espace). Cette fonction peut exprimer la complexité en moyenne (valeur moyenne pour l'ensemble des instances d'une taille donnée) ou au pire (maximum des valeurs pour l'ensemble des instances d'une taille donnée). On utilise souvent les outils mathématiques de l'analyse pour décrire son comportement. Ainsi on peut parler de complexité exponentielle, logarithmique, en $\mathcal{O}(n^2)$, etc. Dans le premier modèle épistémologique proposé, la complexité est un des aspects fondamentaux de l'algorithme. 27, 28, 34, 36, 39, 42, 50

conception Au sens du modèle cK¢une conception décrit la relation d'un sujet ou d'une institution au concept. Elle se décrit par un quadruplet (P, R, L, Σ) où P est un ensemble de problèmes sur lequel la conception est opératoire, R est un ensemble d'opérateur agissant sur P, L est un système de représentation qui permet de décrire R et P, et Σ est une structure de contrôle de l'action de R sur P. 55, 56

μ-conception C'est une conception, au sens de cK¢, qui décrit le concept tel qu'il existe dans le savoir savant. 56, 57

correction La correction d'un algorithme est le fait que cet algorithme fournit une réponse correcte à toute instance du problème que l'on veut résoudre. 23, 30, 36, 41

D

diviser pour régner Un algorithme qui s'appuie sur le principe de diviser pour régner est un algorithme qui, pour résoudre une instance d'un problème, la réduit à la résolution d'une ou plusieurs instances plus petites. En général, on repère plusieurs phases dans un tel algorithme : On découpe l'instance du problème en plusieurs instances plus petites du même problème, on résout ces instances en faisant appel récursivement à l'algorithme puis on combine les solutions de ces instances pour produire la solution recherchée. 26, 38

E

effectivité C'est un des ASPECTS proposé dans le premier modèle théorique. L'EFFECTIVITÉ recouvre l'idée qu'un algorithme doit proposer une résolution systématique, qui doit être finie, non-ambigüe et transmissible à un opérateur quelconque. 40

G

glouton Un algorithme glouton est un algorithme basé sur le principe de faire des choix d'optimisation locale pour trouver un optimum global. 27

I

instance Un problème p étant l'expression d'une même question Q pour tout élément d'un ensemble I donné, on appelle instances les éléments de I. L'instanciation d'un problème est l'expression de la question Q pour une instance choisie. 40, 58, 59

L

langage mathématique Langage utilisé de manière usuelle dans les écrits mathématiques. 62

logique formelle Par opposition à ce que nous avons appelé la logique mathématique, la logique formelle fait référence à la branche des mathématiques et de l'informatique qui s'intéresse à la validité formelle des énoncés. 64

logique mathématique L'ensemble des règles de logique et de raisonnement utilisées implicitement dans l'activité mathématique. Ce terme est choisi ici par opposition à la logique formelle. 62

M

modèles théoriques Ce sont des modèles mathématiques de ce qui est calculable de manière effective, autrement dit de ce qui résoluble par algorithme. Les plus courants sont les machines de Turing et les fonctions récursives. Dans notre premier modèle épistémologique, les MODÈLES THÉORIQUES sont un des aspects fondamentaux de l'algorithme. 25, 42

O

opérateur Dans une conception, l'ensemble des opérateurs (noté R) décrit les actions applicables aux problèmes. 56, 59

outil-objet La dualité outil-objet fait ici référence aux travaux de Douady. L'aspect outil se réfère à l'usage d'un concept, l'aspect objet fait référence au concept comme objet d'étude. En ce qui concerne l'algorithme, nous avons définit l'ensemble $\mathscr{P}_{\text{A}}$ des problèmes pour lesquels il est outil et l'ensemble $\mathbb{P}$ de problème dont il est l'objet. Nous parlons de dialectique outil-objet concernant les relations entre algorithme-outil et algorithme-objet (en particulier au cours de l'activité). 42, 56, 59

P

P et NP Ce sont des classes de problèmes. P représente les problèmes pour lesquels il existe un algorithme de résolution en temps polynomial en fonction de la taille de l'instance. NP représente l'ensemble des problèmes pour lesquels étant donnée une instance et une solution au problème pour cette instance, on peut vérifier en temps polynomial que la solution est valide. On sait que P $\subset$ NP, mais P=NP reste un des grands problèmes mathématique actuels. 25

paradigme Ce sont les grands cadres dans lesquels peut exister l'algorithme selon notre modèle. Ils représentent les "modes de vie" de l'algorithme. Nous avons distingué trois paradigmes : la preuve algorithmique, l'algorithme mathématique et l'algorithme informatique. 60, 62

preuve La PREUVE est l'un des ASPECTS fondamentaux de notre premier modèle. Cet ASPECT recouvre autant les questions de preuve des algorithmes eux-mêmes que le rôle des algorithmes dans les preuves. 40

preuve algorithmique Preuve qui contient dans son expression, de manière plus ou moins explicite, un algorithme de résolution du problème. On peut considérer qu'il s'agit

d'une preuve constructive finie. Dans notre modèle de μ-conceptions, c'est l'un de trois paradigmes pour l'algorithme, noté PA. Il fait référence à une activité de type Problème-Théorème-Preuve, où la preuve est de type constructif et fini. L'algorithme et la preuve sont ici indissociables et la forme de l'algorithme est celle d'une preuve mathématique. 24, 27, 31, 38, 58, 61, 62

problème Dans cette thèse, nous appelons problème un couple (I, Q) où I est une famille d'instances et Q une question pouvant porter sur chacune de ces instances (l'ensemble de tous ces problèmes est noté $\mathscr{P}$). On dit qu'un algorithme résout un problème si pour chaque instance il apporte la réponse à Q. Dans le premier modèle épistémologique, la notion de PROBLÈME fait partie des aspects fondamentaux pour l'algorithme. Dans le modèle par μ-conceptions, la notion de problème permet de distinguer l'algorithme en tant qu'outil de l'algorithme en tant qu'objet (voir outil-objet) en permettant de décrire les familles $\mathscr{P}_{\text{A}}$ et $\mathbb{P}$. Cela permet aussi de définir la notion de problèmes fondamentaux pour l'algorithme ($\mathscr{P}_{\text{FA}}$). 40, 57, 59

programmation dynamique La programmation dynamique est un type d'algorithme (le terme de programmation est à prendre ici au sens de planification ou d'organisation et non de programmation informatique). Son principe est le suivant : pour trouver une solution optimale à une instance d'un problème, on peut s'appuyer sur la recherche de solutions optimales à des instances plus petites. Ainsi, on commence par résoudre de petites instances, puis de plus grandes et ainsi de suite jusqu'à obtention de la solution de l'instance étudiée. 27

programme de modélisation-simulation Expression d'une méthode systématique visant à simuler par informatique un phénomène ou un modèle donné de ce phénomène. Il peut s'agir de simuler un jeu, une machine, une expérience, un modèle de comportement d'un appareil, etc. Les programmes de modélisation-simulation ne sont pas des algorithmes au sens où ils ne résolvent pas un problème. 143

programme-papier Représentation des algorithmes comme des programmes (au niveau formel) hors d'un contexte de programmation mais en conservant de nombreux éléments liés à l'implémentation. Les programmes-papier sont caractéristique d'un ancrage dans AI et bien souvent d'un amalgame algorithme-programme. 129

pseudo-code Nous appelons pseudo-code, un mélange de langage mathématique et d'éléments de langage de programmation permettant d'exprimer des algorithmes sans entrer dans les spécificités d'un langage de programmation. 24, 63

S

savoir-savant Dans l'étude de la transposition didactique, le savoir-savant représente le savoir de référence, tel qu'il existe dans les institutions productrices du savoir. 56

structure de contrôle Dans une conception, l'ensemble des structures de contrôle (noté Σ) assure la non-contradiction de la conception et décrit la relation problèmes-opérateurs. 56, 59

système de représentation Dans une conception, le système de représentation (noté Σ) permet l'expression des problèmes et des opérateurs. 56

T

terminaison La terminaison d'un algorithme est le fait que pour toute instance du problème que l'on veut résoudre, l'exécution de l'algorithme demande un nombre fini d'étapes. 23, 31, 36, 41

V

variable informatique Variable qui joue le rôle d'un emplacement en mémoire en pouvant changer de valeur au cours du temps. Par une opération d'affectation, on peut associer une nouvelle valeur à la variable, l'ancienne valeur est alors perdue. La variable informatique est un modèle de la mémoire d'une machine. 27, 50, 63

variable mathématique Symbole utilisé en mathématiques pour représenter un élément non spécifié ou inconnu d'un ensemble et marquer sa place dans une expression mathématique. 62

Acronymes et symboles

Nous récapitulons ici les principaux acronymes et symboles du document. Les pages correspondent à leur première apparition et définition. Plus de précisions sur certains peuvent être trouvées dans le glossaire précédent.

AI	Algorithme Informatique	61
AM	Algorithme Mathématique	61
PA	Preuve Mathématique	61
ISN	Informatique et Sciences du Numérique	13
$\mathscr{P}$	Problèmes de la forme (Instances, Question)	58
$\mathscr{P}_{\text{A}}$	Problèmes algorithmiquement résolubles	58
$\mathbb{P}$	Problèmes d'algorithmique	58
$\mathfrak{p}_0$	Problème d'appartenance à $\mathscr{P}_{\text{A}}$	58
$\mathbb{P}_{\text{CA}}$	Problèmes de complexité d'algorithme	58
$\mathbb{P}_{\text{CP}}$	Problèmes de complexité de problèmes	58
$\mathscr{P}_{\text{FA}}$	Problèmes fondamentaux pour l'algorithme	217
$P_{1,+}$	Problème de la recherche d'une fausse pièce, plus lourde	227
$P_{1,+/-}$	Problème de la recherche d'une fausse pièce de poids différent	227
$P_{0/1,+}$	Problème de la recherche de zéro ou une fausse pièce, plus lourde	227
$P_{0/1,+/-}$	Problème de la recherche de zéro ou une fausse pièce, de poids différent	227
S_{exp}	Stratégie expérimentale	227
S_{prob}	Stratégie de changement de problème	229
S_{etal}	Stratégie par étalonnage	229
S_{dicho}	Stratégie de dichotomie	227
S_{tricho}	Stratégie de trichotomie	228

Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ?

Résumé

Récemment, l'*algorithme* a pris une place plus importante dans l'enseignement secondaire en France et à l'étranger. Ce concept, lié à l'informatique mais aussi aux mathématiques et à la preuve, soulève de nombreuses questions didactiques. Cette thèse propose une analyse épistémologique du concept dans le but d'étudier sa transposition et de construire des situations didactiques.

Dans un premier temps, nous présentons une analyse épistémologique détaillée du concept en mettant en avant ses aspects fondamentaux. Cela permet de proposer un modèle de *conceptions* pour l'algorithme du point de vue du savoir savant (en mathématiques et informatique) et tenant compte l'ensemble des formes que peut prendre l'algorithme.

Ces résultats, validés expérimentalement par les analyses d'entretiens avec des chercheurs, permettent de mener une étude de la transposition en jeu dans l'enseignement au lycée en France. Au travers de l'étude des instructions officielles, de manuels scolaires et de ressources en ligne, nous mettons en évidence une transposition partielle du concept principalement orientée vers la programmation et l'usage de l'algorithme comme un outil.

La dernière partie propose une caractérisation des *problèmes fondamentaux* pour l'algorithme et des perspectives pour la construction et l'étude de situations didactiques en algorithmique.

Mots-clés

didactique des mathématiques, algorithme, algorithmique, épistémologie, transposition didactique, situations didactiques, conceptions, outil-objet

Teaching algorithm, what for? Some new questions for mathematics and issues for proof learning.

Abstract

Recently, the notion of *algorithm* has gained in importance in secondary school curricula, in France and abroad. *Algorithm* is strongly linked with computer science, mathematics and proof and its teaching raises many didactical questions. In this thesis, we propose an *epistemological analysis* of the algorithm in order to study its *transposition* and to build *didactical situations*.

To begin with, we introduce a detailed epistemological analysis of the concept, highlighting its fundamental aspects. It leads us to construct a model of conceptions for algorithm regarding the academical knowledge (in mathematics and computer science) and taking into account the different forms an algorithm can take. Those results have been experimentally validated by the analysis of interviews of researchers.

This allows us to study the *didactical transposition* involved in the French high school. Through the study of the official curricula, selected textbooks and online resources, we emphasize a partial *transposition* of the concept, mainly *tool-oriented* and based on programming.

In the last part, we propose a characterization of *fundamental problems* for algorithm and perspectives for the design and the study of *didactical situations* in algorithmics.

Keywords

mathematics education, algorithm, algorithmics, epistemology, didactical transposition, didactical situations, conceptions, tool-object