Enseignement de l'informatique : panorama du cycle 4 au post-bac

Rémi Lajugie

Ecrit réflexif
DU : "Enserigner les mathématiques dans le scond degré"
2016-2017

Dans cet écrit réflexif, nous nous intéressons à l'informatique à travers sa place dans l'enseignement secondaire et les premières années du supérieur. L'enseignement de l'informatique prend en effet une place de plus en plus importante dans les programmes de mathématiques. Les nouveaux programmes du collège, et les aménagements proposés au lycée amènent les enseignants de mathématiques à avoir un rôle pivot dans l'enseignement de l'informatique (parfois sous l'appellation "algorithmique et programmation"). Cette place importante de l'informatique amène à se poser de nombreuses questions : l'informatique est elle une discipline à part entière ? doit on nécessairement parler d'ordinateurs ? que doit on enseigner et quand ? comment faire le lien entre le lycée où l'informatique est principalement incluse dans le cours de mathématiques et les premières années du supérieur [1] où l'informatique est une discipline à part entière ? Une première problématique globale pourrait être : "Quelle est la place de l'informatique dans l'enseignement secondaire français et les premières années de l'enseignement supérieur ? Quels sont les obstacles à son enseignement plein et entier ?"

La problématique que nous venons d'énoncer est assez large, elle se centrera donc sur les aspects suivants : "Quelle est la place faite par les programmes de mathématiques à l'enseignement de l'informatique ? Y a t il des obstacles épistémologiques ou historiques à l'enseignement de l'informatique qui sont distincts de ceux inhérents aux mathématiques ? Quel est le lien entre l'informatique au lycée et celle enseignée dans les classes supérieures ? "

Pour traiter ces questions, nous proposons d'adopter le plan décrit ci-dessous.

Dans une première partie nous rappellerons les enjeux de l'informatique, en tant que discipline à part entière. Pour convaincre le lecteur que l'informatique est bien plus qu'une sous branche des mathématiques qui serait apparue il y a quelques années, nous adopterons pour cela une perspective historique qui ne prétend bien entendu absolument pas à être exhaustive.

Par la suite nous nous attacherons à étudier les attendus des programmes du lycée en informatique, en faisant bien entendu le lien avec les programmes du cycle 4.

Dans une troisième partie, nous étudierons la place de l'informatique dans le supérieur en nous restreignant au cas de l'enseignement de l'informatique dans les classes préparatoires aux grandes écoles scientifiques.

Nous conclurons par une partie relevant plus de la prospective, en imaginant ce que pourrait devenir l'enseignement de l'informatique à court ou moyen terme.

1. Rappelons que l'institution scolaire souhaite donner au continuum Bac-3 Bac+3 une place centrale dans l'éducation nationale.

Chapitre 1

L'informatique : émergence d'une nouvelle discipline

Dans cette section, nous allons étudier l'histoire de la discipline informatique. Cela nous permettra d'en sentir les enjeux et donnera aussi des idées sur la difficulté de construire certaines notions propres à l'informatique (les obstacles épistémologiques de Brousseau [Bro89] et Bachelard [BAC38]). En effet, il est important de connaître l'histoire de la discipline, de savoir où se sont situés les écueils, quels concepts ont été lents à émerger. Cela permettra de mieux appréhender les difficultés liées à l'enseignement et à la didactique de la discipline. En effet, si l'informatique est généralement considérée par l'institution scolaire comme une sous-branche de la discipline mathématique, nous pensons que l'informatique mérite le statut de discipline à part entière et qu'il ne faut pas la confondre avec l'utilisation de l'ordinateur pour faire des mathématiques (sujet fort bien documenté [Arc92]).

Il nous est apparu nécessaire de séparer deux aspects dans cet exposé historique : le premier aspect concerne le savoir savant de l'informatique (ce que l'on qualifie souvent d'"informatique théorique"), le second est un aspect d'ingéniérie, technique, qui est lié à la structure des machines utilisées par l'informatique.

Nous pouvons désormais commencer notre voyage à travers le temps.

1.1 L'ingéniérie informatique

Nous allons parler ici de l'évolution des dispositifs de calcul au fil de l'histoire de l'humanité, ce que nous avons appelé la composante "ingéniérie" de l'informatique. Nous allons donc nous focaliser non pas sur l'aspect programmation et algorithmique mais sur l'existence des machines permettant de calculer.

Il est toujours possible de faire remonter ce genre d'historiques aux temps immémoriaux, mais pour les besoins de cet exposé nous nous bornerons à remonter à l'âge classique. Cela nous sera suffisant.

L'ère de la mécanisation. On fait généralement remonter à Pascal (en 1642) la première tentative réussie de créer une machine à calculer automatique : la Pascaline. Ce dispositif de calcul

reposait sur un habile jeu d'engrenages, mécanisant l'algorithme de propagation de retenue dans le système décimal. Quelques années plus tard, Leibniz améliore le dispositif de Pascal pour le rendre capable d'effectuer également des additions et des soustractions. D'autres raffinements de ces dispositifs ont lieu au cours de l'histoire, mais les dispositifs que nous pourrions qualifier d'informatique resteront balbutiants jusqu'à la révolution industrielle.

L'idée de la programmation. Jusqu'au début du XIXème siècle, l'informatique se fait assez discrètes. On peut voir plusieurs raisons à ce phénomène. Nous citerons les deux qui nous paraissent les plus fondamentales : tout d'abord, la seule énergie disponible est mécanique, et si l'on pense à nos ordinateurs actuels, cette énergie apparaît bien dérisoire [1] ; d'autre part on peut dire en termes modernes qu'une machine sert à exécuter un et un seul programme. En d'autres termes ces machines ne sont pas programmables, elles ne peuvent avoir qu'une fonction. L'idée qu'une machine puisse exécuter plusieurs programmes distincts (ce qui est différent d'avoir plusieurs fonctionnalités) n'avait pas encore émergée.

Un grand pas en avant est réalisé au XVIIIème siècle dans la capitale des Gaules, future terre des canuts. Dans ce carrefour du commerce du tissu des artisans (Bouchon, Vaucanson et le plus célèbre d'entre eux : Jacquart) donnent, à l'aide de rubans perforés puis des cartes perforées, à donner des instructions à un métier à tisser. La révolution tient au fait qu'un seul métier à tisser peut filer le tissu de différentes manières suivant le contenu de la carte perforée. Pour la première fois en occident, une machine pouvait exécuter plusieurs programmes. C'est une petite révolution si l'on prend la peine d'y réfléchir, une machine n'est plus associée à une seule et une unique fonction.

La machine de Charles Babbage. Ces cartes perforées, qu'avec nos yeux de modernes nous pouvons qualifier de langage de programmation, a servi de base à ce que la coutume a consacré comme le premier ordinateur de l'histoire : la machine analytique de Charles Babbage [Mor61]. Pour la première fois (1832-1834), une machine à calculer est conçue dans le but d'être programmable. On retrouve là une ébauche de ce que sera l'architecture des ordinateurs modernes (les fameuses machine de Von Neumann) : les registres, les unités de calcul, une forme de mémoire, des périphériques d'entrée, des dispositifs de sortie. Les cartes perforées des métiers à tisser de Jacquart servent à donner les instructions et programmes de calcul.

On considère généralement que cette machine de Babbage est le premier ordinateur de l'histoire. Ada Lovelace a notamment déclaré à propos de cette machine : "La machine analytique n'a nullement la prétention de créer quelque chose par elle-même. Elle peut exécuter tout ce que nous saurons lui ordonner d'exécuter. Son rôle est de nous aider à effectuer ce que nous savons déjà dominer." Cette description pourrait fort bien convenir à nos ordinateurs modernes.

Pour la petite histoire, notons que la machine, initialement commandée et soutenue financièrement par le parlement anglais, a finalement été laissée inachevée en 1834. Cependant, son architecture aboutie et les quelques calculs qu'elle a pu effectuer ont laissé une marque historique.

1. Les ordinateurs de bureau les plus classiques nécessitent environ une puissance de 400W pour fonctionner, ce qui correspond à la puissance déployée par le meilleur coureur cycliste du Tour de France au maximum de son effort. En d'autres termes, à supposer que l'on puisse convertir l'énergie mécanique intégralement en énergie électrique, pour faire fonctionner son ordinateur de bureau, il faudrait que le meilleur sportif du monde produise le maximum de sa puissance en permanence.

Le tournant du siècle. Après l'aventure très (trop ?) ambitieuse de Babbage, les inventeurs, dans le foisonnement inventif du XIXème siècle, reviennent à des objectifs plus modestes. Les années 1880-1900 constituent la période des calculatrices à cartes perforées. On passe définitivement de l'ère de la mécanique à celle de l'électromécanique. C'est à cette époque qu'est fondée, par un certain Hollerith, allemand installé aux états-unis une compagnie d'ingéniérie électromécanique, spécialisée dans la conception de machines de haute technologie et à usage bureautique. Cette compagnie a depuis changé de nom, elle s'appelle aujourd'hui IBM.

L'âge d'or de l'électromécanique. La première partie du XXème siècle, période où l'électricité est devenue une énergie relativement abondante, permet une explosion de l'approche électromécanique. Les machines (notamment contrairement à celle de Babbage) bénéficient désormais d'une énergie plus abondante. L'informatique électromécanique, souvent caricaturée comme étant l'âge des circuits avec des ampoules et des électroaimants, pose les bases de l'architecture moderne des ordinateurs. Les ingénieurs de cette époque cherchent à accomplir le rêve de Babbage, à savoir la construction d'une machine à calculer capable d'exécuter n'importe quel programme. En d'autres termes, l'objectif est de créer une machine qui soit universelle au sens où elle serait capable d'exécuter n'importe quel algorithme/programme de calcul qu'un être humain peut concevoir.

D'immenses calculateurs voient le jour (le Mark1 de Harvard, l'ENIAC dans les années 1940). Cependant, chaque machine avait son mode de fonctionnement propre. Cela posait des soucis que nous qualifierions aujourd'hui de soucis d'inter opérabilité (difficulté à porter un programme sur ruban perforé d'une machine à l'autre, certaines machines gérant les boucles, les instructions conditionnelles, d'autres non etc.). Durant la seconde guerre mondiale, les armées britanniques et américaines vont donner une impulsion décisive à l'élaboration d'une architecture normalisée. Deux brillants mathématiciens : Turing en Angleterre, et Von Neumann aux Etats Unis sont réquisitionnés par l'armée pour créer, à l'aide des meilleurs ingénieurs de leurs pays respectifs, d'immenses ordinateurs dont l'usage sera purement militaire (notamment décrypter les communications entre U-boots). De ces travaux naîtra l'architecture moderne des ordinateurs, que l'on appelle d'ailleurs souvent "architecture de Von Neumann".

La révolution des transistors et la miniaturisation. L'année 1947 marque un tournant décisif dans l'histoire des ordinateurs. C'est cette année là qu'est inventé le transistor dans les Bell labs. Les dispositifs de calcul et les microprocesseurs ont été chamboulé. Là où un ordinateur devait occuper une pièce entière, il n'a besoin maintenant que d'occuper un espace restreint, un petit boîtier. Cela ouvrira la voie dans les années 1970 à la micro informatique personnelle, et à ceux qui sont souvent vus à tort comme les pionniers de l'informatique.

Les ordinateurs deviennent alors des objets incontournables dans le milieu scientifique et dans la vie quotidienne (même si la massification de la micro informatique est relativement tardive). Tous les ordinateurs ayant adopté l'architecture standardisée de Von Neumann, les cartes perforées sont abandonnées au profit d'une programmation que nous qualifierions aujourd'hui de logicielle. Les premiers travaux de Shannon concernant la représentation numérique de l'information, effectués dans les années 1930, sont utilisés et les programmes sont désormais codés sous forme de 0 et de 1 dans des programmes écrits dans certains langages de programmation. La deuxième partie du XXème siècle est celle de l'invention de nombreux langages, chacun ayant un domaine d'appli-

cation différent. Le début du XXIeme siècle semble être l'âge de raison de l'informatique : moins de langages différents, plus de standardisation.

Notons également que dans cet historique nous ne parlons pas de l'essor de la micro-informatique personnelle (smartphones etc.) dont les héros : Bill Gates, Steve Jobs etc. sont connus et qui n'ont finalement fait qu'exploiter les travaux de leurs prédecesseurs sans réellement modifier en profondeur l'architecture des ordinateurs.

1.2 Des mathématiques à l'informatique

Intéressons nous désormais à l'informatique que l'on peut qualifier de théorique et qui est apparue comme sous-discipline des mathématiques. Nous verrons également quand est-ce que cette sous-discipline a rencontré l'ingéniérie pour donner naissance à l'informatique telle que nous la concevons aujourd'hui.

On pourrait faire remonter à très loin un "esprit informatique" dans la résolution de problèmes mathématiques. En effet, nombre de problèmes d'algèbre donnent naissance à des méthodes voire des algorithmes de résolution. En effet, si l'on regarde avec des yeux d'aujourd'hui et que l'on considère que les mathématiques et l'informatique sont des disciplines distinctes, on constate que l'un des liens les plus intéressants entre informatique et mathématiques réside dans la similarité des raisonnements permettant de résoudre un problème. En effet, en informatique et notamment dans la branche que l'on appelle algorithmique, il est très naturel de décomposer un problème en sous-problèmes (instructions) élémentaires dans le but de définir une méthode reposant sur une suite d'instructions atomiques pour le résoudre. Cette approche prévaut également en mathématiques au moins depuis Descartes [Des37], et figure d'ailleurs dans les manuels de résolution de problèmes les plus classiques [Pol57].

Plutôt que de remonter trop loin dans le temps, nous allons étudier comment l'informatique (que nous qualifierons d'informatique théorique afin de simplifier le vocabulaire, même s'il est fortement discutable) a émergé au sein des mathématiques comme une branche de celle-ci.

Revenons au XIXeme siècle, en pleine révolution industrielle. Les mathématiciens commencent à s'intéresser aux problèmes de logique et s'interrogent sur les fondations de leur discipline. Est il possible de tout démontrer à partir d'axiomes ? Est-il possible de résoudre les problèmes mathématiques uniquement à partir d'axiomes et en appliquant des règles de logique apparaissant clairement et distinctement à chacun ?

Les logiciens au XIXème siècle. Si la volonté d'inscrire les mathématiques dans un système axiomatique remonte à longtemps, au moins jusqu'à Euclide, la fin du XIXème siècle a été un moment décisif dans l'histoire.

La découverte des géométries non Euclidiennes par Gauss et Riemann a soulevé de nombreuses questions[DP]. Cette découverte s'inscrivait dans la recherche de la solution d'un très vieux problème : à savoir est ce que le cinquième postulat d'Euclide (équivalent au postulat que nous appelons "postulat des parallèles") est une conséquence logique des autres ou doit on vraiment l'inclure ? Les fameux mathématiciens que nous venons de citer ont démontré, en essayant une sorte de démarche par l'absurde, que ce postulat pouvait être remplacé par d'autres et que cela

donnait naissance à des théories qui semblaient cohérentes (en logique, on parle plutôt de théorie "consistante").

Pour la première fois, on se pose la question de la consistance logique des théories.

La géométrie Euclidienne classique n'amenait pas à se poser ce genre de question puisque sa consistance semblait assurée par l'évidence intuitive de la physique (un plan, quoi qu'étant un objet mathématique, semble avoir une réalité physique). En d'autres termes, puisque le monde qui nous entoure semble cohérent, la géométrie Euclidienne se devait également de l'être.

Les questions axiomatiques prennent alors une importance capitale. En effet, s'il est possible de changer les axiomes, qu'est ce qui garantit la cohérence interne (on parle de consistance logique) d'une théorie mathématique ?

Cela conduit les mathématiciens à s'intéresser de près à la logique. Cette discipline quitte le giron de la philosophie pour devenir l'apanage des seuls mathématiciens. C'est Frege [Fre79] qui marque le mieux ce tournant, en reprenant la logique de Kant et en la mathématisant. Il est le premier à distinguer trois notions qui seront centrales pour le développement de l'informatique : la cohérence, la complétude et la décidabilité, notions sur lesquelles nous reviendrons plus tard.

De nombreuses théories mathématiques fondées sur quelques axiomes voient le jour. Parmi les plus fameuses on peut citer la théorie des ensembles de Cantor, celle de l'arithmétique de Péano, celle de Zermelo et Frankel. Cependant la question de savoir si ces théories englobent bien tout l'univers des possibles et surtout si elles sont non-contradictoires est un problème relativement ouvert. Bertrand Russell et Ernst Zermelo montrent au début du XXeme siècle un paradoxe dans la théorie des ensembles : "il n'existe pas d'ensemble de tous les ensembles". Russell, avec sa théorie des types, colmate très brièvement la brèche. D'autres failles sont découvertes dans d'autres théories. Il se pose alors la question de savoir dans quel mesure on peut faire confiance aux mathématiques. Comment être certain de ne pas pouvoir démontrer une chose et son contraire ? Et même peut on vraiment tout démontrer ? Et peut on le faire de manière systématique ? C'est la crise des fondements.

Le programme de Hilbert. Au début du XXème siècle, le mathématicien David Hilbert, pour résoudre la crise des fondements, propose un programme de recherche [2]. Ce programme visait à trouver une théorie axiomatique des mathématiques qui soit à la fois

1. complète : tout énoncé mathématique peut s'y démontrer,
2. consistante : c'est à dire non contradictoire,
3. décidable : c'est à dire qu'il existe un algorithme universel pour résoudre n'importe quel problème formulé dans la théorie.

C'est ce programme et notamment le troisième point qui marque le début de l'informatique théorique moderne. En d'autres termes, la question est de savoir si les mathématiques sont réductibles au calcul et donc s'il est imaginable d'automatiser tous les processus mathématiques.

2. En Allemagne et dans les pays de tradition anglo-saxonne, il est traditionnel qu'un universitaire titulaire d'une chaire prestigieuse fasse un discours donnant les grandes orientations de la chaire pour les années à venir. Un tel discours est ce que l'on appelle un **programme** ; parmi les programmes fameux, on peut citer, outre celui de Hilbert, ceux de Langlands (ambitieux programme visant à unifier un grand nombre de résultats sur les structures algébriques) ou d'Erlangen (prononcé par Felix Klein et qui visait à lier définitivement géométries et groupes de transformations).

Une réponse positive au programme de Hilbert serait une incroyable révolution dans l'histoire des mathématiques.

Réponse au programme de Hilbert : la théorie de la calculabilité. Les premières réponses au programme de Hilbert sont encourageantes : l'arithmétique de Presburger (arithmétique sans multiplication) ou la géométrie Euclidienne (théorème de Tarski, voir par exemple [Dow13]) sont des théories tout à fait complètes, consistantes et décidables. En d'autres termes, pour ces théories il existe un algorithme (fondée sur une méthode d'élimination des quantificateurs) qui permet de démontrer n'importe quel problème formulé dans leur cadre. Ces théories sont réductibles au calcul.

Cependant, l'arithmétique usuelle et d'autres théories mathématiques résistent aux mathématiciens s'attaquant au programme de Hilbert. Finalement, c'est le logicien Autrichien Kurt Gödel [NN01] qui, en exploitant finement l'argument de l'extraction diagonale démontre que toute théorie de l'arithmétique usuelle est incomplète. Ce résultat de 1931 a mis du temps avant d'être pleinement assimilé par la communauté des mathématiciens.

En parallèle, Alonzo Church et son étudiant à Princeton, un certain Alan Turing, travaillent sur une nouvelle manière d'aborder les problèmes de mathématiques. Ils travaillent également sur le programme de Hilbert, mais s'intéressent plus à la partie décidabilité qu'à celle sur la complétude. Leur approche est une approche extrêmement importante pour l'informatique. Ils ont commencé par montrer que tous les problèmes formulables en mathématiques sont réductibles à un problème de décision (c'est à dire préciser si une phrase est vraie ou fausse). Ils démontrent en 1936 qu'il n'existe pas d'algorithme permettant de décider universellement si une phrase est vraie ou fausse. Une fois de plus, c'est l'usage de l'argument diagonal qui permet de montrer le théorème.

Ce théorème répond au programme de Hilbert par la négative : les mathématiques ne sont pas réductibles au seul calcul, et les raisonnements mathématiques permettent d'accéder à des connaissances nouvelles. Sur le plan philosophique, cela a également eu son importance : en effet, l'argument classique disant que les mathématiques sont une vaste tautologie tombe. Le raisonnement mathématique est une opération de dévoilement de la réalité.

Shannon et la théorie de l'information. Les théories de la décidabilité marquent vraiment le début de l'informatique théorique. Cependant, l'informatique théorique en tant que science autonome n'a pu émerger que grâce aux travaux de Claude Shannon sur la théorie de l'information [Sha48] et ceux, plus anciens, sur la possibilité de représenter l'algèbre de Boole sur un circuit imprimé [Sha38]. La théorie de l'information est d'abord celle de la représentation des données dans un circuit informatique. Shannon développe le concept d'entropie sur un canal de communication, et également celui de bit d'information. Ce concept étend l'entropie de Boltzmann pour les gaz. Ces concepts permettent de donner un fondement mathématique à des questions concernant la représentation de l'information : ce concept a désormais un sens théorique. C'est extrêmement important pour le développement de la science informatique car elle a désormais un objet d'étude : l'information. [3]

3. Mathématiquement, l'entropie d'une distribution discrète pouvant prendre n valeurs distinctes que l'on représente par un vecteur $(p_1; \ldots; p_n)$ est la fonction $H(p) = -\sum_{i=1}^{n} p_i \log p_i$.

Les ordinateurs théoriques. Le dernier ingrédient nécessaire à l'émergence de l'informatique théorique et à sa séparation des mathématiques est la fameuse création de Alan Turing [Tur37] : la machine de Turing (voir par exemple [Wol91] pour une introduction mathématique des concepts, nous nous contenterons ici d'une description informelle de l'objet). La machine de Turing est un objet théorique composée d'une tête de lecture, d'un ruban infini composé de cases dans lesquelles se trouvent des symboles (une case peut éventuellement être blanche). A chaque pas de temps, la tête de lecture lit ce qui est marqué sur le ruban et effectue une action (par exemple avancer le ruban d'une case vers la droite, ou encore s'arrêter). La machine change alors d'état (les états sont en nombre finis). Il a été démontré par la suite par Church et Turing que ce modèle est universel dans un certain sens et que toute machine à calculer peut être remplacé par une machine de Turing équivalente. Cette machine théorique dont les théoriciens, au moment de la publication de l'article fondateur de Turing, doutaient qu'elle puisse jamais être approchée est un modèle absolument essentiel puisque c'est celui d'une méta-machine, celle qui peut calculer absolument tout ce qui est calculable, de traiter toutes les informations ; en d'autres termes la machine capable de simuler toutes les autres machines jamais créées par l'homme [4].

Une discipline autonome : quand les ordinateurs théoriques rencontrent les vraies machines. La première partie du XXème siècle pose toutes les bases de ce que nous appelons l'informatique théorique. Cependant, la discipline est encore considérée comme une branche des mathématiques.

C'est l'histoire qui force finalement l'informatique à devenir autonome et fait se rencontrer l'informatique théorique et les ordinateurs. La tragique seconde guerre mondiale a forcé les grands esprits de l'époque à mettre leurs concepts de calculabilité au service de la construction effective de véritables machines de Turing, que nous avons pris l'habitude de désigner sous le nom d'ordinateurs. Durant les années 1940, Turing, Church, Von Neumann, Shannon, Kleene et les autres sont réquisitionnés par les armées (et travaillent notamment du côté de Princeton). C'est une des premières fois que ces théoriciens sont confrontés aux praticiens des dispositifs de calcul dont nous avons parlé dans la section précédente. L'Histoire force les ingénieurs et les mathématiciens à travailler ensemble. C'est à ce moment là que l'informatique devient une discipline, indépendante des mathématiques, tout en lui étant liée.

L'informatique théorique dans la deuxième moitié du XXème siècle. La deuxième partie du XXème siècle a été celle de l'essor de l'informatique théorique, des théories des langages de programmation et des automates. L'informatique théorique (computer science dans la langue de Shakespeare) est devenue une discipline autonome. Des départements universitaires sont créés, des chaires d'informatiques également.

Cependant, l'union des théoriciens et des praticiens permise par la seconde guerre mondiale s'est quelque peu affaiblie, même si ingénieurs et universitaires sont convaincus de la nécessité de cette union. La discipline informatique reste une construction fragile et est encore trop cloisonnée entre les deux branches dont nous venons de brièvement faire le survol historique.

4. A ce sujet, songeons que les bombes atomiques ne sont plus testées en grandeur nature mais simulées à l'aide d'une machine de Turing.

1.3 Difficulté à concilier les points de vue et enseignement de la discipline

Cette longue introduction historique a pour objectif de soulever la difficile émergence de l'informatique comme une discipline unifiée (et cela va rendre difficile son enseignement, ce qui est quand même ce qui nous intéresse ici). Il y a, dans la discipline informatique une dualité entre une partie que l'on pourrait qualifier de théorique et l'autre de pratique. Ces deux aspects ne sont pas toujours mis en relation alors qu'ils constituent les deux faces d'une même médaille. Aujourd'hui encore de nombreux ingénieurs informatiques ne connaissent que peu la partie mathématique de la discipline et vice-versa. On a vu précédemment que, pour que les ingénieurs et les informaticiens théoriciens se parlent et fondent véritablement une discipline autonome, il a fallu des circonstances exceptionnelles.

On comprend alors d'où viennent certains problèmes que nous retrouvons quand il s'agit d'enseigner l'informatique aujourd'hui par exemple au collège : d'un côté le professeur de technologie enseigne à se servir de l'outil ordinateur et le professeur de mathématiques parle d'algorithmique, le tout sans réelle concertation mutuelle, comme si l'ingénieur et l'informaticien théoricien devaient impérativement être séparés. La dualité que nous avons soulevé en retraçant l'histoire de la discipline a créé dans l'institution scolaire une dualité pour son enseignement.

Un enseignement de l'informatique à l'intérieur des mathématiques et de la technologie soulève-t-il des questions ? On peut bien entendu défendre une solution pluridisciplinaire à ce problème, notamment grâce aux EPIs permis par la récente réforme du collège. Cependant, recourir à de telles solutions reviendrait à nier le caractère disciplinaire de l'informatique et à n'y voir que l'intersection de diverses disciplines. Ce n'est pas vraiment le point de vue que nous souhaitons défendre ici.

Différence entre mathématiques et informatique : la notion de variable. En effet, la didactique de l'informatique n'est pas tout à fait la même que celle des mathématiques. Les notions de fonctions, variables n'ont pas tout à fait la même signification et surtout appellent à des représentations mentales différentes. On sait à quel point la représentation mentale en informatique et mathématiques est cruciale pour la bonne compréhension des concepts et démonstrations (voir par exemple [AF10] pour une exploitation fort à propos de représentations mentales pour démontrer l'inégalité de Bienaymé-Chebyshev).

Examinons ce qu'est une variable informatique dans le contexte d'une fonction informatique. Nous nous intéresserons ensuite à la question de la variable mathématique dans une fonction mathématique.

La variable informatique peut généralement se représenter mentalement comme étant une certaine case dans la mémoire de l'ordinateur dans laquelle on écrit une certaine valeur. L'opération informatique de base sur la variable est l'*affectation*, toutes les autres opérations (incrémentation, effacement) sont fondées sur cette opération fondamentale. D'autre part, en informatique, la représentation de la variable comme une case mémoire autorise la création de pointeurs ou de références

vers la variable [5]. La variable mathématique n'a pas du tout le même statut épistémologique et didactique. Pour simplifier, supposons que la fonction mathématique que nous étudions soit donné par une expression explicite $f(x)$, la variable (que l'on appelle en logique mathématique une variable libre) prend un statut différent puisqu'elle ne représente à la fois aucun nombre et tous ceux d'un ensemble potentiel (ensemble de définition). La variable libre mathématique appelle nécessairement la notion de quantification [Fre79]. La variable informatique, si elle peut changer de valeur par le truchement de l'opération d'affectation, a toujours a priori une valeur (dans la case mémoire, il se trouve toujours une certaine séquence de 0 et de 1).

Récit d'expérience. Nous pouvons également mentionner à ce propos une expérience personnellement rencontrée. Lors de certaines de nos séances TICE, nous avons pu constater que les élèves questionnent la légitimité de l'enseignant en mathématiques à enseigner l'informatique. Nous avons notamment entendu, de la bouche d'élèves dans une classe de seconde au lycée Murat d'Issoire des phrases comme : "Mais c'est pas des maths !" ou encore, lors d'une séance d'informatique débranchée [6] "Bah l'informatique ça se fait qu'avec des ordinateurs". Nous avons été assez désemparés face à ces questions, mais une fois que l'on retrace l'histoire de la discipline, il devient évident que ces questions sont légitimes. Les deux jambes de l'informatique sont trop souvent vues comme étant indépendantes et le lien entre elles n'est pas fait. Sans compter que, très souvent on a tendance à ne voir en l'informatique que l'un ou l'autre des aspects. La confusion est encore plus grande lorsque l'on sait que dans le langage courant (et également dans les intitulés de certaines matières scolaires), tout usage des ordinateurs est considéré comme étant "de l'informatique", chose qui devrait nous paraître aussi étrange que d'affirmer que l'usage d'un crayon de couleur dans une discipline fait de la discipline de l'"art plastique". La définition de ce qu'est la discipline informatique est difficile et cela contribue à rendre très délicat son enseignement au sein du cours de mathématiques.

5. Pour faire simple, une référence ou un pointeur (nous ne ferons pas la nuance entre les deux dans ce texte) vers une variable est une variable dont la valeur est l'adresse d'une autre variable.

6. Il s'agissait de faire sortir un robot d'un labyrinthe en lui donnant des instructions élémentaires.

Chapitre 2

L'enseignement de l'informatique dans le secondaire

Dans ce chapitre, nous nous intéressons à la place de l'informatique dans les programmes de collège et de lycée. L'informatique n'étant pas (encore) reconnue comme une discipline scolaire à part entière, son enseignement est réparti dans ceux de mathématiques et de technologie. Nous allons nous focaliser sur les programmes de mathématiques.

Remarque historique préliminaire. L'histoire de l'enseignement de l'informatique a en effet été assez chaotique. Dans les années 1980, une grande vague de l'enseignement de l'informatique a atteint les systèmes éducatifs anglo-saxons et français, on voulait alors initier les élèves à la programmation (en LISP et son dérivé éducatif : le fameux langage LOGO dont la tortue a acquis une grande notoriété dans les milieux éducatifs et informatiques). Les thèses de Papert [Pap80] étaient alors en vogue : la programmation informatique devait absolument faire partie des connaissances et savoir-faire maîtrisés par les élèves à la fin de leur curriculum. Les années 1990 et 2000 ont été marquées par un recul de la vague programmation au profit de la vague Multimédia/utilisation des TIC en milieu scolaire [Bar94].

Les nouveaux programmes de lycée de 2008, et la création de la filière ISN marquent le début d'un retour en force de l'aspect "programmation informatique" dans les programmes scolaires. La récente réforme du collège consacre ce retour.

Comme remarqué par les didacticiens de l'informatique [BB11], un des enjeux de l'enseignement actuel de l'informatique est de réussir à donner aux élèves une représentation mentale du fonctionnement d'un ordinateur (et a fortiori du fonctionnement de la programmation informatique). En effet, les élèves sont désormais des usagers de l'informatique mais, en grande partie grâce aux efforts des ingénieurs, l'ergonomie des systèmes est tellement simple que l'outil n'a pas besoin d'être maîtrisé pour être utilisé. Il s'agit en quelque sorte de renoncer à une conception primitive de l'ordinateur comme étant une machine dotée de réactions parfois aléatoire voire à une conception animiste du fonctionnement de la machine [1]

1. Au sens de Piaget [Pia26], même si ce dernier utilisait le terme d'animisme pour de plus jeunes enfants que ceux que l'on peut rencontrer dans l'enseignement secondaire. Il nous est arrivé en classe de seconde de détecter une forme d'animisme dans la conception de certains objets pour les élèves (et parfois même chez certains adultes), notamment en ce qui concerne l'informatique.

2.1 Dans les programmes de mathématiques au collège (cycle 4)

La récente réforme du collège a fait la part belle à l'enseignement de l'informatique dans le cycle 4. La maîtrise de la programmation est un attendu de fin de cycle.

La place de l'algorithmique et de la programmation. La programmation est une activité difficile puisqu'elle consiste à faire résoudre un problème par un ordinateur comme le fait fort bien remarquer [Ars88]. Il faut donc au préalable avoir posé le problème et savoir le résoudre par une méthode générale que l'on peut appliquer à des variables informatiques. En d'autres termes, on ne peut faire programmer à un élève que la résolution d'un problème sur lequel il a suffisamment de recul pour en avoir une représentation mentale abstraite et sur lequel il sait qu'il peut jouer mentalement avec les variables (longueur d'un segment par exemple). Il faut ensuite ajouter une autre difficulté qui est le prolongement des difficultés liées aux problèmes de communication : une fois la méthode dégagée, il faudra trouver un moyen de la communiquer à l'ordinateur sous la forme d'un programme. La sélection des exercices de programmation devra donc bien être gouvernée par le souci de présenter aux élèves des problèmes dont ils sont capables de dresser une méthode de résolution abstraite fondée sur l'assemblage de blocs atomiques [Des37] (en d'autres termes s'ils sont capables de trouver un algorithme pour résoudre le problème, l'algorithmique précédant alors la programmation).

Il est à noter que, pour la première fois dans le secondaire, un langage de programmation (ici Scratch) est explicitement recommandé par les documents d'accompagnement du programme de mathématiques [DGE]. Ce choix de langage est fort indiqué pour le collège à cause de sa simplicité d'utilisation, qui évitera d'ajouter des soucis de syntaxe informatique à la difficile abstraction nécessaire pour intégrer un algorithme à programmer.

Programmation avec Scratch. Scratch est un langage par blocs orienté objets (un paradigme de programmation très répandu, puisque c'est celui adopté par Java, C++ et dans une moindre mesure par Python). Il a été développé par le MIT afin de proposer à des enfants un langage de programmation à la fois ludique et fonctionnel. Les blocs scratch, de diverses couleurs permettent de séparer de manière visuelle les différentes composantes d'un programme informatique. Ainsi les instructions conditionnelles, les boucles, les affectations ont des couleurs différentes et demandent aux élèves de cliquer dans différents menus. Cela permet aux élèves de ne pas tout mélanger et de bien comprendre qu'un programme informatique n'est pas qu'une suite de symboles sans signification mais plutôt que les instructions ont des statuts logiques différents. Une variable n'est pas une instruction conditionnelle etc.

L'intention pédagogique des créateurs de Scratch [RMMH$^+$09] était de libérer les enfants des problèmes inhérents à la rigidité de certains langages de programmation pour limiter au maximum la possibilité de bugs de syntaxe dont la répétition peut être très décourageante pour le débutant en programmation. Un autre souci des concepteurs de scratch était de libérer l'informatique du problème du typage. Scratch ne nécessite en effet aucune connaissance en structure de données (listes tableaux etc.). S'affranchir de ces problèmes permet aux élèves/enfants de se focaliser sur l'intérêt premier du langage qui est de programmer en donnant des instructions. Scratch permet

notamment de bien appréhender une épineuse question de la didactique de l'informatique : celle de l'affectation d'une variable (appelée opération d'attribution dans Scratch). Scratch écrit l'affectation en toute lettre et ne propose pas de symbole ou de notation pour cette opération. Pour des élèves de collège qui n'ont pas forcément totalement bien assimilé le sens du symbole égal et pas encore compris que ce n'était qu'un symbole conventionnel dont la sémantique est donnée par le contexte, il est très prudent de ne pas utiliser le symbole "$=$" pour désigner l'opération d'affectation de variable. Cette ambiguïté sur l'usage de l'égalité comme symbole d'affectation est un souci fréquemment soulevé par les collègues enseignants (notamment ceux qui utilisent Python au lycée), et il est difficile de prévenir les erreurs des élèves. En effet une égalité $i = i + 1$ dans un code Python n'a pas du tout le même statut que l'équation mathématique $i = i+1$ et la variable informatique n'a pas le même rôle que la variable mathématique. La variable informatique appelle une représentation mentale très différente de celle de la variable mathématique.

Les concepteurs de Scratch se sont inspiré des travaux de Pappert dans les années 1980 sur la didactique de l'informatique [Pap80]. Il préconisait qu'un langage de programmation soit extrêmement simple de prime abord, de sorte que l'investissement pour faire des programmes spectaculaires (et donc valorisants) soit minimal et en même temps extrêmement complet pour permettre, dans le cadre du langage et sans passer à un autre, réaliser des programmes répondant à de complexes cahiers des charges. Le rêve de Pappert était donc la création d'un langage de programmation qui puisse suivre l'élève tout au long de sa scolarité. Scratch répond partiellement à cet ambitieux projet. En effet, si Scratch est suffisant pour une utilisation du primaire jusqu'à la première année du lycée, il est délicat de l'utiliser dans le cadre de la spécialité ISN de Terminale S ou même en première ES ou S.

L'utilisation de Scratch, si elle résout un grand nombre de problèmes laisse donc entier le problème du passage ultérieur à un langage de programmation plus "classique" (Python par exemple qui est, à raison, très utilisé dans les lycées). Les problèmes didactiques liés à la syntaxe (par exemple l'opérateur d'affection noté avec le symbole égal) sont laissés pour plus tard mais devront quand même être considérés.

Algorithmique. Ni les documents d'accompagnement [DGE], ni les nouveaux programmes ne proposent explicitement de séparer l'enseignement de la programmation et celui de l'algorithmique. L'accent est mis sur quelques difficultés didactiques inhérentes à l'algorithmique : notamment la notion de variable informatique dont le sens est différent de celui qu'il peut revêtir en mathématiques. A ce sujet, on peut mentionner que l'opération d'affectation conduit souvent dans les langages de programmation à écrire des expressions du type $i \leftarrow i + 1$ ou, ce qui est encore plus source de confusion pour les élèves : $i = i + 1$. On rajoute encore un type d'égalités pour les élèves. Il faudra être très vigilant à limiter les confusions dans l'esprit des élèves. Comme mentionné dans le paragraphe précédent, la manière dont le langage Scratch résout le problème mérite qu'on s'y attarde.

Le programme de collège ne se limite cependant pas qu'aux aspects que nous avons mentionnés plus haut, qui sont des aspects techniques ou des connaissances (instructions conditionnelles etc.). Il est également attendu que les élèves développent un esprit que l'on pourrait qualifier d'esprit informatique. Ils doivent devenir capables, non seulement de produire quelques programmes élémentaires mais également de "Décomposer un problème en sous-problèmes afin de structurer

un programme ; reconnaître des schémas." Il s'agit donc d'être capable de créer une démarche algorithmique dans le but de résoudre un problème donné puis de le faire fonctionner dans un ordinateur.

N'enseignant pas cette année en collège, nous n'avons pas pu mesurer tous les enjeux didactiques de l'enseignement de l'algorithmique et de la programmation dont la place est centrale dans les nouveaux programmes du cycle 4.

2.2 Dans les programmes de mathématiques du lycée

Au lycée, les programmes de mathématiques comportent une partie informatique sous le nom d'"algorithmique et programmation".

La place de l'algorithmique. Les programmes soulignent l'aspect fondamental de la composante algorithmique de la démarche mathématique. En effet, pour rentrer dans une démarche algorithmique en vue de résoudre un problème, il faut dégager une suite de sous problèmes élémentaires ou atomiques qu'il convient ensuite de réassembler. Il s'agit en somme de prolonger la démarche cartésienne [Des37] qui disait : "Je cherche à diviser chacune des difficultés que j'examinerois, en autant de parcelles qu'il se pourroit, et qu'il seroit requis pour les mieux résoudre."

Il est explicitement commandé par les programmes de lycées de proposer des algorithmes en vue de résoudre certaines questions. Le programme de seconde met notamment en valeur la place des algorithmes pour toutes les questions de calcul (les élèves voient souvent au collège des problèmes de type "création d'un programme de calcul"), dans le domaine de la géométrie avec des programmes de construction notamment, dans les questions de probabilités et de statistiques (algorithmes de simulation) et également pour toutes les questions liées à l'approximation numérique (l'algorithme de dichotomie est explicitement évoqué par le programme de seconde, et il est possible en terminale d'évoquer les méthodes de Newton, Euler ou encore de Héron pour approcher les solutions de certaines équations).

La place de la programmation. Les programmes du lycée pas plus que les documents d'accompagnement, n'imposent de langage de programmation. Cependant, il est mentionné que les élèves doivent dans la mesure du possible programmer les algorithmes vus en classe que ce soit à l'aide d'une calculatrice, d'un tableur ou d'un langage de programmation usuel.

Il est explicitement indiqué que les élèves doivent connaître les bases de la programmation impérative : ils doivent être capables de réaliser de petits programmes avec des boucles bornées (boucle POUR) ou non bornées (boucle TANT QUE) et des instructions conditionnelles.

Notons que les objectifs en algorithmique et en programmation sont des objectifs globaux pour le lycée et que l'enseignant est laissé libre de choisir les moments auxquels il souhaite introduire telle ou telle notion. Certains lycées, comme celui où nous enseignons, font le choix de ne pas mettre en place de progressivité commune sur ces objectifs. D'autres collègues nous ont mentionné l'existence de progressivité unifiée au sein de leur établissement. Il apparaît relativement évident que le dernier cas évoqué est le plus à même d'assurer la cohérence pédagogique au sein d'un établissement.

Les objectifs en programmation sont certes globaux, mais si nous regardons attentivement le programme de seconde, nous voyons la présence de l'algorithme de dichotomie. Son implémentation requiert la mobilisation de toutes les compétences attendues en programmation pour le lycée. Il faut donc avoir travaillé sur les boucles non bornées, les instructions conditionnelles dès la classe de seconde.

Récit d'expérience : Nous avons, dans nos classes de première ES et de seconde tenté diverses approches pour l'enseignement de l'informatique. Comme le long historique en première partie peut en témoigner, nous sommes convaincus qu'il faut impérativement faire sentir aux élèves (dans le respect des programmes, bien entendu) que l'informatique est à la fois une discipline appliquée au sens où elle est la science des ordinateurs mais également une discipline théorique et est un mode de pensée. Cette pratique peut se faire de manière "débranchée", c'est à dire uniquement avec un papier et un crayon. Les manuels scolaires regorgent d'exercices de type algorithmique permettant de faire de l'informatique de cette manière. Cependant, les exercices proposés sont très souvent artificiels. Nous avons souvent essayé de proposer des exercices de type "algorithmique" avec des objectifs plus élaborés que ceux que l'on retrouve parfois dans les manuels.

En seconde, l'introduction de l'outil calculatrice programmable a été l'occasion de s'interroger sur le fonctionnement de la calculatrice. La calculatrice a un avantage très intéressant : son langage (un langage de type Basic) est extrêmement différent du langage naturel (auquel la plupart des langages de blocs utilisés en lycée comme Algobox ont recours). Il permet ainsi de bien séparer l'activité algorithmique sur papier de l'activité de programmation qui est une activité de traduction en un langage intelligible par une machine. Il est cependant important de noter que cet avantage peut se transformer en inconvénient. En effet lorsque le parc de calculatrice n'est pas homogène, l'enseignant doit gérer plusieurs langages de calculatrices (Basic Casio et Basic TI par exemple) en parallèle. Or ces langages ont des syntaxes distinctes. Cela peut consommer beaucoup de ressources attentionnelles de l'enseignant. Nous avons été confronté à ce problème et n'avons pas vraiment trouvé de solution satisfaisante mis à part de demander une homogénéisation du parc de calculatrices en vigueur dans l'établissement (homogénéisation prônée par toute l'équipe pédagogique du lycée où nous exerçons).

En première ES, nous avons également étudié une boucle "tant que" de manière à la fois pratique et théorique. Cette étude s'est faite durant le chapitre de probabilités, lors d'un travail pratique concernant le problème de la ruine du joueur. Un joueur compulsif jouait aux dés et nous étudions les cas où il finissait ruiné. Durant le TP, nous étudions un cas dégénéré où le joueur gagnait soit 1 euro quand sortait un nombre pair, soit 0 euro. Les élèves avaient un petit programme algobox fondé sur une boucle "TANT QUE" qui simulait le comportement du joueur. Et forcément, avec cette règle du jeu le programme tournait en permanence sans s'arrêter. Ce fut l'occasion de faire un point sur la nuance entre programme (morceau de code informatique qui peut ne pas terminer) et algorithme (procédure qui termine). Les élèves également furent confrontés à la "bêtise" de l'ordinateur puisque les élèves pouvaient très bien voir que le programme tournerait indéfiniment alors que l'ordinateur n'en est pas capable.

Cette distinction entre programme et algorithme n'est certes pas explicitement au programme, mais elle permet de voir une nuance fondamentale et de donner aux élèves une culture informatique qui fait souvent défaut. Elle permet de voir que l'esprit humain est capable de choses que la machine

n'est pas capable de faire : à savoir dire *a priori* par simple lecture du programme que l'on va boucler de manière indéfinie [2].

Mieux motiver l'algorithmique. Une des difficultés de l'enseignement de la partie algorithmique du programme et que notre pratique soulève réside dans le peu d'intérêt qu'ont les exercices d'algorithmique. Souvent, les exercices d'algorithmique n'ont pas vraiment raison d'être. Rares sont les problèmes dits d'algorithmique amenant une véritable résolution de problème. Il est pourtant toute une classe de problèmes accessibles au lycée amenant à de vraies démonstrations algorithmiques. Pour que l'enseignant soit convaincant, il faut qu'il soit convaincu que son exercice a de l'intérêt. Or, il nous est plusieurs fois arrivé de donner des exercices d'algorithmique ou des programmes de calcul sans véritable intérêt, au sens où la démarche algorithmique n'apporte pas vraiment un éclairage nouveau sur le programme.

Il nous faut rechercher des exercices nécessitant véritablement une résolution algorithmique. L'histoire des mathématiques regorge d'exemples que nous avons trop souvent délaissé pour l'enseignement. Comme fait par [Dow10], on peut par exemple considérer la résolution d'une équation diophantienne du type :

$$X^n = \sum_{i=0}^{n-1} a_i X^i.$$

Comme X^n croît plus vite que tous les termes du membre de droite de l'égalité, un raisonnement analytique permet de limiter les entiers possibles à ceux inférieurs à une certaine borne [3] donc de n'avoir qu'un nombre fini de solutions potentielles possibles. On peut alors tester de manière exhaustive ces possibilités par ordinateur.

Cet exemple est une belle application du principe d'élimination de l'infini dont l'application a conduit à la démonstration du célèbre théorème des quatre couleurs [AH$^+$77] ou encore résolu le problème des sphères empilées de manière compacte [Hal05]. Il s'agit, d'éliminer l'infinité de possibilité et ensuite d'utiliser un ordinateur, un algorithme informatique pour résoudre le problème. A l'avenir, nous souhaiterions "tester" des algorithmes reposant sur cette méthode dans nos classes. Le programme de seconde ne nous a pas permis de trouver une situation reposant sur ce principe, cependant les programmes de première et surtout de Terminale offrent un certain nombre de possibilités (à partir du moment où l'on fait des suites ou des mathématiques discrètes).

2.3 Au lycée : de ICN à ISN

L'informatique trouve également sa place au lycée désormais dans une forme semi-autonome des programmes de mathématiques. Un enseignement d'exploration Informatique et Création Numérique (ICN) a été créé en seconde et un enseignement de spécialité de terminale S : Informatique et Sciences du Numérique (ISN) a également été créé (il existe également des filières ICN

2. On dirait en termes informatiques que le problème de l'analyse statique des programmes n'est pas décidable. Cela signifie simplement qu'il n'est pas possible pour un ordinateur de dire *a priori* s'il va rentrer dans une boucle infinie ou non. En d'autres termes, tant que l'on n'a pas essayé le programme, on ne peut pas être totalement certain qu'il n'y a pas de bug.

3. Ceux plus petits que $n \max a_0; \ldots; a_n$.

en première ES et première L mais ces filières sont encore rares et nous n'avons pas pu avoir de renseignements autres que ceux délivrés directement par l'institution scolaire.).

Ces enseignements sont faits par des enseignants disposant d'une habilitation spécifique. Généralement, les enseignants habilités sont ceux de mathématiques ou plus généralement de science ; par exemple au lycée Murat d'Issoire un enseignant de sciences de la vie et de la terre ainsi qu'un enseignant de mathématiques sont habilités à enseigner ICN et ISN.

L'enseignement d'exploration Informatique et Création Numérique. Au lycée Murat d'Issoire, un enseignement d'exploration de seconde d'informatique et de création numérique, regroupant une quarantaine d'élèves est présent depuis la rentrée 2015. Dans cet enseignement d'exploration, les élèves sont initiés aux problèmes de la création numérique (droit d'auteur etc.) mais aussi au fonctionnement d'un ordinateur (principaux composants). Les élèves sont initiés à la programmation au travers de projets divers (traitement d'images, de sons, rédaction de programmes en scratch ou en python).

N'ayant pas pu assister cette année à des cours de cet enseignement, nous n'en parlerons pas beaucoup plus.

L'enseignement de spécialité de Terminale S : Informatique et Sciences du Numérique (ISN). Les élèves voient dans cette spécialité un panorama des divers aspects de l'ingéniérie informatique : la création de sites internet, le traitement d'images, la programmation impérative. Le choix des logiciels n'est pas imposé par le programme mais pour les pages web, les enseignants se limitent à présenter HTML et CSS (les élèves les plus volontaires peuvent se lancer dans du javascript ou du PHP). Concernant la programmation, le choix se porte généralement sur Python qui dispose d'une grande souplesse dans la syntaxe et se prête bien à l'enseignement. Le programme est très ambitieux et recommande d'adopter un point de vue global sur le monde de l'informatique (réseaux, programmation, traitement des signaux) sans oublier l'aspect sociétal en s'interrogeant sur les mutations de la société liées au numérique, notamment le côté juridique

Une caractéristique importante de cet enseignement de spécialité est qu'il donne lieu à une évaluation comptant pour le baccalauréat (les oraux se passent en mai). Cet oral prend la forme (au lycée Murat) de la présentation d'un projet préparé durant la deuxième moitié de l'année. Les élèves ont, généralement par groupes de deux, réalisé un projet informatique (site internet, mini jeu vidéo, programme utilisable sur tablette etc.). Devant le jury, ils doivent défendre ce projet en présentant :

1. la part de travail de chacun,
2. le cahier des charges initial,
3. les difficultés rencontrées,
4. une production finalisée et fonctionnelle.

Une grande partie de l'année est donc consacrée au travail sur le projet. Les élèves de Terminale S disposent généralement de suffisamment de maturité pour cela. La pédagogie par projet étant nouvelle pour eux, il faut les guider. Il s'agit non seulement d'être acteur de son savoir et de le construire, mais également de proposer des solutions techniques pour obtenir un résultat. Il faut aussi être capable de bâtir un cahier des charges réaliste et se projeter sur le long terme. Ils doivent

planifier sur une longue période et imaginer diverses solutions de repli dans le cas où le projet n'aboutirait pas.

Cette pédagogie par projet [BSM+91] présente l'avantage de donner un élan de motivation aux élèves (nous avons souvent pu observer dans nos classes un manque de motivation chronique). En contrepartie, en faisant se focaliser les élèves sur leurs projets, elle limite la possibilité d'explorer avec tous et de manière exhaustive le programme de la discipline. Il faut donc réussir à trouver un compromis entre la pédagogie par projet pure et une pédagogie plus classique dont l'objectif premier est de finir un programme.

Une des difficultés de la pédagogie de projet du point de vue de l'enseignant est celle de l'évaluation. Une approche d'évaluation par les pairs se justifie souvent dans ce cadre et peut revêtir un caractère éducatif [Rev13]. Il faut cependant se garder de n'évaluer qu'une production finale car la conduite de projet est avant tout une démarche. Il faut évaluer le respect d'un cahier des charges, une capacité à surmonter des difficultés, en bref à évaluer l'écart entre ce qui était planifié et ce qui s'est effectivement produit. Une évaluation fondée sur des compétences et capacités peut prendre toute sa place dans ce cadre.

L'enseignement ISN résout la question de l'évaluation en faisant une évaluation du projet par un jury de professeurs qualifiés, à qui l'élève présente le projet en respectant les critères énoncés plus haut.

Récit d'expérience : Nous avons eu l'occasion, au cours de cette année d'aller en observation dans les cours de la spécialité de terminale S d'ISN et l'enseignant visité nous a permis d'intervenir auprès des élèves pour les aider dans leurs projets informatiques. De cette expérience on peut ressortir plusieurs choses intéressantes :

— Tout d'abord la pédagogie de projet est très adaptée pour ce genre d'enseignement. Les élèves sont en effet acteurs de leur savoir et construisent eux mêmes leurs connaissances de manière naturelle. Le fait d'avoir un projet filé semble donner une grande motivation aux élèves. Certains, dont il paraît qu'ils sont en difficulté dans certains cours plus classiques, s'investissent beaucoup dans leur projet et produisent des choses intéressantes. Ils sont très friands de conseils techniques alors que, si l'on en croit leur enseignant de mathématiques, ils peuvent avoir une attitude très passive en cours. La plupart des élèves travaillent beaucoup sur leur projet en dehors de la salle de classe. Il est fréquent qu'une fois la sonnerie passée, des élèves restent dans la salle car "ils veulent finir ce qu'ils ont commencé" (bien entendu ce n'est pas le cas de tous mais le phénomène est suffisamment fréquent pour être signalé et nous semble démontrer une grande motivation et un grand investissement de la part des élèves).
— Au delà des connaissances, les élèves acquièrent une autonomie dans la recherche de solutions techniques aux problèmes qu'ils se posent. Partant de leur objectif, ils doivent mettre au point une stratégie pour y arriver. En informatique il y a souvent de nombreux chemins menant à une même solution. Il est en revanche dommage que les élèves n'essayent pas plusieurs solutions techniques et aient tendance à trop souvent se précipiter sur la première solution trouvée sur internet.
— L'accent est mis sur la composante ingéniérie et création numérique de l'informatique, le programme traite assez peu d'informatique théorique (dont nous retraçons l'historique dans

la deuxième partie de l'introduction).

— La grande autonomie laissée aux élèves cause cependant quelques soucis. En effet, le programme d'ISN n'évoque pas les structures de données et ne sont donc pas l'objet de TPs en début d'année. Cela cause un problème : par exemple les élèves mélangent allègrement listes chaînées et tableaux. Dès Lorsqu'ils cherchent des ressources pour leur projet de fin d'année sur internet, ils tombent sur des solutions techniques reposant sur un usage fin de certaines structures de données. L'enseignant, personne ressource dans ce projet, ne peut cependant pas apporter une explication suffisamment détaillée du rôle fondamental des structures de données en informatique et doit se contenter de présenter le choix technique comme quelque peu magique. Un programme moins rigide serait peut être plus à même de laisser du temps pour travailler certains points comme les structures de données, en laissant de côté certains points de détails concernant le routage dans un réseau.

Chapitre 3

Dans le supérieur : l'exemple de l'enseignement dans les CPGE

Pour saisir les enjeux de la discipline informatique, il est bon de voir comment elle est enseignée dans les classes post-bac. On retrouve de nombreuses filières à forte dominante informatique (en IUT informatique, en IUT Réseaux et Télécoms, à l'Université, en classes préparatoires aux grandes écoles). L'exhaustivité dans l'étude de ces diverses filières nous prendrait trop de temps. C'est pourquoi nous allons nous concentrer sur le cas particulier des classes préparatoires scientifiques.

L'enseignement de l'informatique en classes préparatoires prend deux formes : la première est celle d'une discipline à part entière commune à tous (l'"informatique pour tous", 2h par semaine en filière scientifique), et la deuxième est celle d'une spécialisation proposée aux élèves de MPSI/MP, les sciences industrielles sont alors remplacées par un enseignement de l'informatique théorique.

Pour ce chapitre nous avons interrogé un collègue de mathématiques en charge de l'enseignement d'informatique pour tous en classes préparatoires dans un lycée parisien plutôt réputé.

3.1 L'option informatique en MPSI/MP

Les élèves de MPSI ont, à la fin du premier semestre le choix entre une option sciences de l'ingénieur (qui permet de passer en fin d'année soit en MP option sciences industrielles ou en filière PSI) et une option informatique (à raison de 2h par semaine, une heure de cours et une heure sur machines, qui permet seulement de passer en MP option informatique). L'informatique de cette option est une informatique très axée sur la théorie mathématique de la représentation de l'information et sur la complexité algorithmique.

Cette option étant très orientée vers les mathématiques, elle est exclusivement enseignée par des enseignants de mathématiques.

Le langage de programmation choisi pour cette option informatique est le langage CamL (le langage Pascal est encore autorisé pour passer les concours, mais la plupart des enseignants utilisent CamL). CamL est un langage fonctionnel, ce qui renforce sa proximité avec les mathématiques. Les langages fonctionnels sont des langages reposant sur le concept fondamental de fonction (une fonction en langage CamL n'a par exemple qu'une seule sortie). Ces langages se prêtent bien à une analyse formelle théorique et permettent de définir facilement des types et des structures de données.

D'un point de vue didactique et pédagogique, recourir à un langage fonctionnel permet de montrer aux étudiants le côté extrêmement mathématique derrière l'informatique théorique. Les enseignants de CPGE louent souvent la rigidité de CamL et le fait que ce langage "implémente directement la pratique des mathématiques". A l'aide de ce langage, les élèves vont étudier les structures de données classiques de l'informatique (arbres et plus généralement graphes, piles), les algorithmes usuels (plus courts chemins dans des graphes, algorithmes de tri). L'idée est vraiment de mettre les mains au plus profond de la structure de l'informatique et des procédés de traitement de l'information.

3.2 L'informatique pour tous en filière scientifique

Depuis la refonte des programmes de 2013, on trouve l'informatique en tant que discipline à part entière dans toutes les filières scientifiques de CPGE. Cette matière (informatique commune ou informatique pour tous) est évaluée lors des principaux concours d'admission aux grandes écoles soit sous la forme d'une épreuve de mathématiques requérant de produire des programmes en Python (c'est le cas pour le concours Centrale ou le concours Mines/Ponts/Télécom), soit sous la forme de questions dans les sujets de mathématiques nécessitant de rédiger un petit programme (c'est le cas par exemple aux concours communs polytechnique).

Les langages de programmation commandés par les programmes sont le langage Python ainsi que le langage Scilab. Ces deux langages sont des langages interprétés (ils ne donnent pas lieu à la création d'exécutables binaires susceptibles d'être lancé sur une machine ne disposant pas de l'interpréteur du langage). Leur avantage est d'être extrêmement souples du point de vue de la syntaxe et également d'être énormément utilisés à la fois dans le monde professionnel et dans les écoles d'ingénieur qui dispensent pour la plupart leurs enseignements d'informatique en utilisant Python comme support.

L'enseignement de l'informatique commune se déroule sur trois semestres (toute l'année de mathématiques supérieures et la première moitié de l'année de spéciales) à raison de deux heures élèves par semaine (2h de TDs sur machine par groupes d'une vingtaine d'élève). L'enseignement est effectué généralement par des enseignants de sciences industrielles ou de mathématiques, l'enseignement ne nécessitant pas une habilitation spéciale.

Contenus. L'objectif affiché par le chapeau du programme est de donner aux étudiants une connaissance suffisante des concepts fondamentaux de l'informatique pour pouvoir communiquer avec les informaticiens. Le but est de donner une culture et une compétence suffisante aux futurs ingénieurs pour qu'ils soient capables de comprendre ce dont les informaticiens parlent. Il faut aussi construire avec les étudiants une représentation mentale de l'architecture interne d'un ordinateur (architecture de Neumann). La compétence de communication est donc très naturellement travaillée et est d'ailleurs un enjeu important de l'évaluation lors des concours terminaux.

Au niveau des contenus, contrairement à ce qui est fait dans les classes du lycée, l'accent est mis sur la notion de structure et la nécessité d'organiser un programme dans le souci de le rendre intelligible par les autres. Les étudiants doivent connaître les différentes structures élémentaires et les problèmes qu'il convient de résoudre à leur aide.

Le travail sur l'algorithmique vu dans le secondaire est poursuivi et est complété par une définition formelle de ce qu'est un algorithme (et donc un travail sur les notions de terminaison et de correction) ainsi que la notion afférente de complexité (en temps et en espace), qui permet de réinvestir les connaissances sur les suites. La complexité est une notion fondamentale en algorithmique. Elle est à la base de la plupart des grands problèmes de l'informatique moderne.[1] Il est attendu que les élèves soient capables d'évaluer le caractère "raisonnable" de leur algorithme. En effet, l'informatique doit gérer le temps, et la question de savoir si une réponse peut être apportée dans une échelle de temps acceptable est cruciale.

Une autre partie importante du programme porte sur les problèmes liés à la précision numérique et la représentation des nombres en mémoire. Contrairement aux mathématiques, l'informatique est une science de la finitude. Les nombres, s'ils peuvent parfois donner l'impression du contraire, ne sont jamais représentés que sur un nombre fini de bits.

Enfin une autre importante partie reprend une partie du programme de simulation numérique des anciens programmes de mathématiques des classes préparatoires. Cette partie implémente (et démontre la convergence et la complexité) de quelques fameuses méthodes : celle de Newton, celle d'Euler, celle de Runge-Kutta.

Une autre partie du programme est consacré aux bases de données, omniprésentes à l'ère des "big data". Les élèves doivent pouvoir interroger un serveur fonctionnant sur le modèle SQL.

Enjeux pédagogiques. L'enseignement de classes préparatoires comporte un programme assez dense comme nous l'avons vu plus haut. Cependant, l'enjeu principal est d'amener les étudiants à construire des solutions élaborées à des situations complexes qu'ils n'ont jamais rencontrées avant. Le programme recommande de partir de problèmes concrets, que l'on trouve dans l'industrie, pour construire les solutions informatiques de manière pragmatique. La pédagogie par mini-projets a donc ici toute sa place. De surcroît, les écoles d'ingénieurs auxquelles se destinent les étudiants de classes préparatoires seront naturellement confrontés à la pédagogie par projet qui est un mode tout à fait classique de travail dans les écoles d'ingénieur où le point culminant des études est la réalisation d'un projet de fin d'étude (PFE) [Rev13].

Souvent, les enseignants choisissent de consacrer un semestre entier à la réalisation par les élèves d'un projet long (souvent en lien avec le TIPE des élèves).

Les questions de précision numérique doivent être abordées à partir de situations concrètes donnant lieu à des problèmes. Il est intéressant de mettre les élèves face à une situation où l'enseignant sait qu'un problème de précision numérique va survenir et laisser les élèves chercher un bug logiciel là où le problème vient de la représentation en mémoire. Il est ensuite intéressant de faire le lien avec les effets potentiels de ces erreurs numériques (explosion d'Ariane 5 par exemple).

La question de l'évaluation est un enjeu crucial des classes préparatoires. Les concours terminaux proposent généralement une évaluation à l'écrit et à l'oral (l'épreuve de Mathématiques II du concours Centrale). Il faut donc entraîner les élèves, non seulement à programmer, mais aussi à écrire des algorithmes et programmes sur papier, de sorte à ce que ces programmes soient intelligibles de leur correcteur.

1. La question à un million de dollars de savoir si la classe de complexité P est égale à la classe de complexité NP cherche à établir s'il existe un algorithme polynômial capable de résoudre des problèmes dont on estime qu'ils sont difficiles (parmi lesquels 3-SAT ou le problème du voyageur de commerce).

Écrire des programmes sur papier est un difficile exercice de communication. Il faut en effet avoir atteint un degré d'abstraction très supérieur à celui du problème posé et être capable, non seulement de proposer un algorithme atomique pour le résoudre mais également le structurer pour en faire un programme cohérent et intelligible.

Eclairage sur les programmes et compétences développées dans le secondaire. Le programme des classes préparatoires aux grandes écoles a l'avantage de ressembler aux programmes que nous avons l'habitude de rencontrer dans le secondaire (c'est en grande partie ce qui nous a poussé à étudier le programme des classes préparatoires plutôt que les PPN des DUT par exemple).

Le programme des classes préparatoires distingue huit grandes compétences informatiques. Lorsque nous enseignons dans le secondaire, il nous fait chercher à travailler ce même genre de compétences. Les compétences sont les suivantes :

— Analyser et modéliser une situation,
— Concevoir une solution efficace au moyen de méthodes de programmation et de structures de données,
— Programmer un algorithme,
— Contrôler et valider les programmes,
— Communiquer autour de la solution informatique proposée à un problème.

Toutes ces compétences sont transposables dans le secondaire et peuvent nous inciter à repenser notre progressivité en algorithmique et programmation. Dans les classes d'ISN où nous avons pu observer le comportement des élèves mais aussi intervenir, nous avons vu comment la pédagogie par projet permet de travailler toutes ces compétences à la fois. Par exemple, un élève réalisant un site internet doit d'abord faire une phase d'analyse de la situation, puis doit concevoir son site avant de programmer puis de valider sa production. Enfin il doit communiquer autour de sa création, en faire la publicité et justifier certains choix d'organisation vis à vis de son public (où placer telle ou telle barre de menu etc.).

Détaillons désormais comment ces compétences peuvent être développées dans le cadre des programmes de lycée. L'enseignement de l'informatique doit constamment chercher à développer ces compétences. Leur lecture peut nous donner des idées quant aux tâches à donner aux élèves.

L'analyse de situation et la construction d'algorithmes pour modéliser une situation requiert le développement d'un grand recul sur sa pratique mathématique. Cette compétence ne peut probablement pas être travaillée trop abruptement. Il faut guider les problèmes de modélisation proposés aux élèves. Par exemple, partant d'un programme de construction qu'ils auront réalisé à la main, on essaiera d'abstraire une méthode générale.

La compétence de programmation est explicite et requiert de nombreuses capacités (traduire un algorithme en programme, débugger son programme etc.).

Trop souvent, nous avons tendance à négliger l'importance de la compétence de "validation d'un programme". Il faut absolument présenter aux élèves des exemples de programmes (ou à la rigueur d'algorithmes) qui sont faux et qu'il faut corriger. Nous avons essayé un tel exemple avec un programme de calcul erroné en classe de seconde.

Chapitre 4

Perspectives

Ce long balayage des programmes et des méthodes d'enseignement de l'informatique nous laisse quelque peu sur notre faim : les multiples réformes n'ont pas encore réussi à donner sa place à l'informatique au sein de l'enseignement scolaire français. Il est tout à fait logique que cette place soit difficile à faire. Dans cette partie nous allons nous interroger sur la suite, c'est à dire ce qui risque de se passer à court terme dans le domaine de l'enseignement de l'informatique.

4.1 Au collège et au lycée : mouvement de dissociation des mathématiques et de la technologie ?

Orientation actuelle : vers une bivalence des professeurs de mathématiques. Il semblerait que l'orientation actuellement choisie par l'institution,[1] soit de laisser l'enseignement de l'informatique aux professeurs de technologie (en collège) et aux professeurs de mathématiques (en lycée et en collège).

Il semble donc qu'il faille attendre des professeurs de mathématiques une certaine bivalence : mathématiques et informatique (comme on attend des professeurs de physique qu'ils soient également professeurs de chimie). Cette bivalence se marque notamment au niveau des concours de recrutement des enseignants de mathématiques. L'agrégation dispose depuis longtemps d'une option "informatique" dans laquelle les candidats ont une épreuve orale spécifique se fondant sur des questions d'informatique théorique (calculabilité, complexité, décidabilité, théorie des langages, algorithmique). Le CAPES 2017 [dlna] est le premier à présenter une option "Informatique". Une des épreuves écrites d'admissibilité du CAPES de mathématiques est remplacée par une épreuve écrite d'informatique dans laquelle les candidats doivent résoudre des problèmes de manière algorithmique et en proposant des programmes Python (tous les programmes étant écrits sur papier, l'épreuve restant une épreuve écrite et non pas une épreuve sur machine. Une des épreuves d'admission est également remplacée par une épreuve d'informatique pour les candidats choisissant cette option à compter de la session 2017 du CAPES de mathématiques.

1. Pour des raisons diverses comprenant la difficulté de dégager des horaires spécifiquement dédiés à l'informatique, la difficulté de monter une formation en informatique pour de nouveaux enseignants, et des raisons financières.

Arguments en faveur d'un enseignement à part entière. Nous pensons cependant que la voie la plus pérenne à suivre pour l'institution scolaire est celle de l'autonomisation de la discipline informatique en lui donnant une place à part entière. Le premier argument en faveur d'un enseignement disciplinaire à part entière vient peut être du malaise de bon nombre d'enseignants de mathématiques vis-à-vis de l'informatique : les témoignages de professeurs de mathématiques avouant ne pas s'intéresser à la discipline ou disant n'avoir pas la formation nécessaire pour l'enseigner correctement sont nombreux.

Un deuxième argument vient de l'enseignement à l'université : les universités et différentes écoles de l'enseignement supérieur séparent les mathématiques de l'informatique, les universités ayant toutes des laboratoires d'informatique distincts des laboratoires de mathématiques. Dès lors que le cloisonnement disciplinaire existe dans le supérieur, son inexistence dans le secondaire peut être questionné : qu'est ce qui justifie que l'informatique soit mêlé aux mathématiques au lycée mais plus dans l'enseignement supérieur ? La discipline est certes jeune mais elle est mûre pour être détachée des mathématiques.

Un autre argument intéressant concerne le point de vue des élèves. Les mathématiques peuvent causer des blocages psychologiques et dissocier dans l'esprit des élèves. La littérature de psychologie cognitive sur les blocages psychologiques liés aux mathématiques est très riche et documentée (voir par exemple [Mig13]). Dissocier dans les emplois du temps des élèves l'informatique des mathématiques serait probablement un moyen de ne pas créer de blocages chez les élèves ou d'éviter des transferts de "craintes" liées à l'histoire personnelle des élèves avec les mathématiques.

4.2 Aménagements du programme de seconde générale et technologique

Suite à la réforme du collège, et probablement en préalable à une réforme du lycée (au moins en ce qui concerne les programmes), le conseil supérieur des programmes a diffusé en novembre 2016 une proposition d'amendement au programme de seconde [dlnb]. Ces programmes prennent en compte la mise en place des nouveaux programmes de cycle 4. En toute cohérence avec ce qui se fait désormais au collège, un thème entier du programme est consacré à l'algorithmique et à la programmation. Notons que, contrairement au précédent programme qui ne mentionnait que des objectifs généraux pour le lycée en entier, les objectifs sont désormais ciblés sur un niveau de classe en particulier.

Globalement les attendus en informatique sont explicités. Si le langage de programmation n'est pas encore explicitement commandé, il devient évident que les concepteurs de programme attendent que les enseignants du lycée se mettent à enseigner le langage Python ("Un langage de programmation simple d'usage est nécessaire pour l'écriture des programmes. Le choix du langage se fera parmi les langages interprétés, concis, largement répandus, et pouvant fonctionner dans une diversité d'environnements. ").

Le programme aménagé propose explicitement de faire découvrir aux élèves la notion de fonction informatique. Une chose intéressante est que les fonctions informatiques prennent naturellement comme ensemble de départ des ensembles différents de celui des réels, voire même peuvent avoir plusieurs arguments. Cependant, si comme cela est préconisé par le programme, le langage

choisi est le langage Python, l'analogie avec les fonctions mathématiques s'arrêtera là. En effet, en Python [2] une fonction peut retourner plusieurs sorties (en d'autres termes, plusieurs images). Cela pourrait assurément prêter à confusion dans l'esprit des élèves.

Nous comprenons fort bien que les concepteurs des programmes ont voulu donner à la partie informatique une dimension concrète, d'utilité pour la formation professionnelle. Cependant, préconiser un langage fonctionnel comme c'est le cas dans l'enseignement de l'informatique en MPSI/MP aurait probablement eu un effet encore plus bénéfique en classe de seconde. En effet, si l'on adopte une approche de construction ascendante du savoir [Bro89], utiliser un tel langage en informatique aurait conduit à construire la même notion de fonction en informatique et en mathématique (c'est d'ailleurs car leurs fonctions sont des fonctions au sens mathématique que les langages fonctionnels sont appelés ainsi).

La nouvelle rédaction du programme pourrait inciter certains collègues à faire des séquences intégrales "algorithmique et programmation", comme il y a des séquences intégrales de probabilités, analyse, algèbre etc. Cependant, il faut se garder de cette lecture hâtive. Cette interprétation ne semble pas conforme à la volonté du rédacteur des programmes qui a stipulé dans l'en tête de la proposition d'aménagement du programme de mathématiques que "l'algorithmique a une place naturelle dans tous les champs des mathématiques et les problèmes ainsi posés doivent être en relation avec les autres parties d u programme (fonctions, géométrie, statistiques et probabilité, logique) mais aussi avec les autres disciplines ou la vie courante. "

2. Et contrairement à ce qui se passe dans certains autres langages ayant un paradigme dit "fonctionnel" comme le langage CamL.

Chapitre 5

Conclusion : comment enseigner l'informatique au lycée ?

Le panorama que nous avons dressé s'est fait en deux grands temps : nous avons tout d'abord brossé un long, mais très incomplet, historique de l'informatique puis nous avons analysé les modalités d'enseignement de l'informatique à l'heure actuelle dans l'enseignement scolaire français. De cette double analyse, on peut tirer plusieurs enseignements :

— l'informatique, bien que parfois considérée comme relevant du champ des mathématiques, est une discipline à part entière dont l'histoire que nous avons retracé montre bien le caractère distinct des mathématiques ;

— la discipline informatique souffre de problèmes de jeunesse : ses contours ne sont pas forcément définitivement fixés, et l'institution scolaire n'a pas encore réussi à trouver une place pleinement satisfaisante pour enseigner la matière ;

— l'informatique comporte deux composantes : l'une d'ingéniérie et l'autre théorique. L'enseignement ne peut pas oublier un aspect au profit de l'autre, il faut conserver le souci de présenter à la fois des résultats théoriques et également faire véritablement de la programmation ;

— la discipline présente des obstacles didactiques différents de ceux des mathématiques et son enseignement se prête bien à l'innovation pédagogique ;

— l'informatique mérite définitivement un enseignement dédié. Cependant des contraintes, liées à la structure des emplois du temps des élèves, à l'organisation de l'institution scolaire font que cet enseignement repose en grande partie sur l'enseignant de mathématiques. Ce dernier doit donc impérativement se sensibiliser encore plus qu'il ne l'est actuellement aux enjeux de cette discipline.

Bibliographie

[AF10] Pierre Arnoux and Alain Finkel. Using mental imagery processes for teaching and research in mathematics and computer science. *International Journal of Mathematical Education in Science and Technology*, 41(2) :229–242, 2010.

[AH+77] Kenneth Appel, Wolfgang Haken, et al. Every planar map is four colorable. *Illinois Journal of Mathematics*, 21(3) :429–490, 1977.

[Arc92] Jean-Pierre Archambault. L'ordinateur pour enseigner les mathématiques. *Bulletin de l'EPI (Enseignement Public et Informatique)*, (67) :125–133, 1992.

[Ars88] Jacques Arsac. La didactique de l'informatique : un problème ouvert ? In *Colloque francophone sur la didactique de l'informatique*, pages 9–18. Association EPI, 1988.

[BAC38] GASTON BACHELARD. La formation de l'esprit scientifique. 1938.

[Bar94] Georges-Louis Baron. *L'informatique et ses usagers dans l'éducation*. Habilitation à diriger des recherches, Université René Descartes - Paris V, September 1994.

[BB11] Georges-Louis Baron and Éric Bruillard. L'informatique et son enseignement dans l'enseignement scolaire général français : enjeux de pouvoir et de savoirs. In *Recherches et expertises pour l'enseignement scientifique*, volume 1, pages 79–90. De Boeck Supérieur, 2011.

[Bro89] Guy Brousseau. Les obstacles épistémologiques et la didactique des mathématiques, 1989.

[BSM+91] Phyllis C Blumenfeld, Elliot Soloway, Ronald W Marx, Joseph S Krajcik, Mark Guzdial, and Annemarie Palincsar. Motivating project-based learning : Sustaining the doing, supporting the learning. *Educational psychologist*, 26(3-4) :369–398, 1991.

[Des37] René Descartes. *Discours de la méthode*. 1637.

[DGE] DGESCO. Document d'accompagnement au programme de mathématiques cycle 4 : Algorithmique et programmation.

[dlna] Ministère de l'éducation nationale. Programme officiel du capes de mathématiques de la session 2017.

[dlnb] Ministère de l'éducation nationale. Proposition d'aménagement du programme de mathématiques de la classe de seconde.

[Dow10] Gilles Dowek. Diophante, l'infini et les ordinateurs. *http ://www.les-ernest.fr/diophante-linfini-et-les-ordinateurs*, 2010.

[Dow13] Gilles Dowek. *Les métamorphoses du calcul.* Le Pommier, 2013.

[DP] Amy Dahan and Jeanne Peiffer. *Une histoire des mathématiques : routes et dédales*. Editions du Seuil.

[Fre79] Gottlob Frege. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. *From Frege to Gödel : A source book in mathematical logic*, 1931 :1–82, 1879.

[Hal05] Thomas C Hales. A proof of the kepler conjecture. *Annals of mathematics*, 162(3) :1065–1185, 2005.

[Mig13] John Mighton. For the love of maths. *Scientific american, MIND*, 24 :60–64, 2013.

[Mor61] Philip Morrison. *Charles Babbage and his calculating engines ; selected writings by Charles Babbage and others.* Dover Publications, 1961.

[NN01] Ernest Nagel and James R. Newman. *Godel's Proof*. NYU Press, 2001.

[Pap80] Seymour Papert. *Mindstorms : Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[Pia26] Jean Piaget. *La représentation du monde chez l'enfant*. 1926.

[Pol57] George Polyà. *How to solve it.* Garden City, 1957.

[Rev13] Catherine Reverdy. Des projets pour mieux apprendre ? *Veille et analyse*, 82, 2013.

[RMMH+09] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch : programming for all. *Communications of the ACM*, 52(11) :60–67, 2009.

[Sha38] Claude E Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12) :713–723, 1938.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, 1948.

[Tur37] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1) :230–265, 1937.

[Wol91] Pierre Wolper. *Introduction à la calculabilité*. InterEditions, Collection iia, 1991.