Analyse didactique d'un jeu de recherche : vers une situation fondamentale pour la complexité d'algorithmes et de problèmes ⋆

Antoine Meyer[1] and Simon Modeste[2]

[1] LIGM (UMR 8049), UPEM, CNRS, ESIEE, ENPC, Université Paris-Est, 77454 Marne-la-Vallée, France
antoine.meyer@u-pem.fr
[2] IMAG, Université de Montpellier, CNRS, Montpellier – France
simon.modeste@umontpellier.fr

Résumé En analyse d'algorithmes, on s'intéresse souvent à la notion de complexité en temps et dans le pire cas : étant donné un algorithme résolvant un certain problème, peut-on estimer le nombre d'opérations effectuées pour résoudre la pire instance d'une certaine taille ? Une autre notion importante s'intéresse à la complexité du problème lui-même, indépendamment d'un algorithme : peut-on démontrer que tout algorithme résolvant ce problème effectue au moins un certain nombre d'opérations ? Nous questionnons la possibilité d'aborder à divers niveaux d'enseignement (scolaire ou supérieur) les notions de complexité au pire d'un algorithme et de complexité intrinsèque d'un problème. Après une brève présentation des notions visées, nous présentons une famille d'activités de type « débranché » inspirées du classique « jeu de la devinette », dont nous faisons l'hypothèse qu'elles sont de nature à faire émerger ces notions, voire de constituer une situation fondamentale à leur égard. Ces activités reposent en particulier sur la notion d'argument d'adversaire, utilisée en théorie de la complexité. Nous fournissons une analyse a priori de cette famille d'activités et présentons un plan d'expérimentation.

Keywords: complexité · algorithme · problème · ingénierie didactique · jeu · informatique débranchée

1 Introduction

Selon Modeste [10], qui s'appuie sur une revue de littérature informatique et didactique, 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 ». Les algorithmes occupent en France une place croissante dans les programmes scolaires. Ils interviennent en mathématiques et en technologie au cycle 4 (fin du collège, 12 à 15 ans), en

⋆. Travail réalisé avec le soutien financier de l'ANR (projet DEMaIn, référence <ANR-16-CE38-0006-01>).

mathématiques au lycée (seconde, première et terminale, 15 à 18 ans), et surtout dans les nouvelles disciplines SNT [3] et NSI [4], qui identifient les algorithmes comme l'un des quatre « concepts fondamentaux » et « universels » de la science informatique. Déjà au cycle 3 (9 à 12 ans), les élèves y sont familiarisés par le biais d'algorithmes simples de déplacement, de construction de figures, etc.

Suite à la popularisation par Wing [13] du terme *pensée algorithmique*, divers travaux ont tenté d'apporter des précisions sur les compétences exactes que recouvre ce terme. S'appuyant sur une revue de la littérature parue entre 2006 et 2013, Selby et Woolard [11] proposent de retenir cinq compétences principales, que l'on pourrait traduire par (1) la capacité à penser par abstractions, (2) à décomposer un problème, (3) à planifier et exécuter une procédure algorithmique, (4) à évaluer un algorithme selon divers critères, et enfin (5) à généraliser. Dans cet article, nous nous intéresserons dans une certaine mesure à la compétence (3), mais plus particulièrement à la compétence (4), que nous comprendrons ici dans le sens spécifique d'*évaluation de l'efficacité d'un algorithme*, préoccupation généralement associée au thème de la *complexité algorithmique*.

La question de l'enseignement de cette notion dans le cadre de cours d'initiation à l'informatique et à la programmation apparaît dans la littérature. Par exemple, [6] plaide pour une introduction précoce, juste après la présentation des boucles. Ceci est illustré par trois exemples de problèmes principalement arithmétiques. Dans [5], les auteurs décrivent un nouveau type d'exercices destiné à sensibiliser des étudiants de première année d'informatique à la notion de complexité et à l'existence possible d'algorithmes d'efficacité variées pour un même problème. L'originalité de notre travail est d'accorder une importance particulière à l'étude de la complexité de *problèmes* algorithmiques, non spécifique à un ou plusieurs algorithmes particuliers. Ceci permet de poser la question de l'existence d'algorithmes efficaces (ou plus efficaces que d'autres) pour un problème donné, et parfois de démontrer qu'il n'en existe pas.

Dans la suite de cet article, après une brève discussion sur le contexte institutionnel (section 2), nous présentons nos objectifs, hypothèses et questions (3). Après un rappel des notions algorithmiques visées (section 4), nous exposons une famille de situations d'apprentissages liée au problème de la recherche d'élément dans une collection triée (section 5), dont nous faisons l'hypothèse qu'elle est susceptible de constituer une situation fondamentale pour la notion de complexité algorithmique en général, et celle de complexité de problème en particulier. Enfin, nous présentons quelques perspectives de recherche, incluant un plan d'expérimentation et d'analyse à divers niveaux scolaires (section 6).

2 La complexité algorithmique dans les programmes

Pour des élèves découvrant la notion de complexité au pire d'un algorithme, une motivation possible est la recherche d'efficacité et d'économie de moyen. Il est entendu qu'un « bon » algorithme est un algorithme efficace (performant,

3. SNT : Sciences numériques et technologie (seconde)
4. NSI : Numérique et Sciences Informatiques (première et terminale)

rapide, économe en ressources). Ceci rejoint d'ailleurs une tendance naturelle de la recherche en algorithmique. Ce thème de la complexité algorithmique est très nettement pris en compte par les programmes officiels de NSI. Par exemple, dans le programme de première [8], on peut lire :

> Quelques algorithmes classiques sont étudiés. L'étude de leurs coûts respectifs prend tout son sens dans le cas de données nombreuses (...).

Cette exigence se reflète dans les attendus relatifs à plusieurs points de la partie algorithmique du programme. Par exemple, dans le cas du parcours séquentiel d'un tableau, « [o]n montre que le coût est linéaire », tandis que pour les algorithmes de tri par sélection ou par insertion « [o]n montre que leur coût est quadratique *dans le pire cas* ». Le programme aborde également l'algorithme de « recherche dichotomique dans un tableau trié », sans cependant mentionner sa complexité. Dans le programme de NSI de terminale [9], le thème de la complexité d'algorithmes est repris en des termes plus précis :

> On continue l'étude de la notion de coût d'exécution, en temps ou en mémoire et on montre l'intérêt du passage d'un coût quadratique en n^2 à $n \log_2 n$ ou de n à $\log_2 n$. [5]

Il y a ici un implicite important : on compare des algorithmes *résolvant le même problème* : ainsi, on peut supposer que le passage « d'un coût quadratique en n^2 à $n \log_2 n$ » fait allusion au problème du tri et au passage d'algorithmes dits naïfs à des algorithmes plus efficaces (tri par fusion notamment, qui est évoqué dans la suite du programme), et que le passage « de n à $\log_2 n$ » fait allusion au problème de la recherche d'élément dans une collection triée et au passage de l'algorithme de recherche séquentiel à la recherche dite « dichotomique » (même si ce dernier n'apparaît explicitement que dans le programme de première).

Il est donc en réalité question ici d'établir des faits à propos de la complexité de *problèmes*, et non seulement d'algorithmes : en passant de première à terminale, il est par exemple attendu que l'élève comprenne que le *problème* du tri, dont on savait en première qu'il était de complexité $O(n^2)$, est en fait en $O(n \log_2 n)$, comme en attestent les algorithmes étudiés en terminale.

Ainsi, il est légitime de poser la question de la limite de ce phénomène de perfectionnement progressif des algorithmes : pour un problème donné, y a-t-il une borne inférieure à l'efficacité des algorithmes qui le résovent ? Autrement dit, est-on en mesure de démontrer que *tout* algorithme résolvant le problème *doit* avoir une complexité supérieure à un certain seuil ? Si oui, est-il possible de mettre au point un algorithme *optimal*, dont la complexité dans le pire cas est précisément égale à cette complexité minimale ? On peut arguer du fait que l'existence d'algorithmes optimaux pour certains problèmes (dont au moins deux, la recherche par dichotomie et le tri par fusion, sont explicitement visés par les programmes de NSI) justifie l'intérêt d'étudier ces algorithmes, même si cette justification n'apparaît pas de manière explicite dans les programmes.

5. Le programme précise ici que « Le logarithme en base 2 est ici manipulé comme simple outil de comptage (taille en bits d'un nombre entier). », ce qui est inévitable puisque la fonction logarithme n'est pas toujours connue à ce niveau de classe.

3 Hypothèses de recherche

Les observations ci-dessus nous amènent à formuler les hypothèses suivantes :

H1 : L'étude de la complexité algorithmique par le biais des *problèmes*, et non seulement des algorithmes, est signifiante pour la compréhension des enjeux de la pensée algorithmique.

H2 : Il est possible d'aborder, même de manière informelle, les notions de borne supérieure et de borne inférieure de complexité d'un problème dans le cadre scolaire, par l'intermédiaire du jeu à deux joueurs.

H3 : Le problème de la recherche d'élément dans une suite triée est un objet d'étude propice à l'émergence des notions de complexité d'algorithme et de problème à différents niveaux scolaires.

L'hypothèse H1, de nature plutôt épistémologique, est soutenue par le fait que c'est l'étude de la complexité d'un problème (et non d'un unique algorithme) qui fournit à la fois le cadre et l'enjeu de l'analyse comparée d'algorithmes, comme le sous-entendent les programmes de NSI (cf. section 2). C'est également le point de vue général sur le problème algorithmique en tant qu'objet qui donne du sens à la notion d'algorithme optimal.

L'hypothèse H2 s'appuie sur des idées issues de la théorie algorithmique des jeux. Dans ce cadre, les joueurs qui s'opposent cherchent à atteindre des objectifs antagonistes, qui peuvent dans certains cas être formulés comme des problèmes d'optimisation : chaque joueur cherche à maximiser son gain et à minimiser celui de l'adversaire. Nous verrons à la section 4.2 que la question de la complexité d'un problème peut être présentée dans le cadre d'un « jeu » entre l'algorithme, qui cherche à minimiser par exemple le nombre d'opérations utilisées, et son environnement qui cherche au contraire à maximiser cette quantité.

En endossant le rôle de l'algorithme, on peut faire l'hypothèse que l'élève renforce sa compréhension de la notion même d'algorithme interagissant avec un contexte d'exécution donné. En endossant le rôle de l'adversaire, l'élève qui joue « contre » l'algorithme cherche à faire apparaître l'un des pires cas pour celui-ci. On peut penser que cela permet de faire émerger ou de renforcer chez l'élève la notion de complexité dans le pire cas, tout en préparant l'idée de borne inférieure sur la complexité du problème.

Dans le prolongement des hypothèses H1 et H2, l'hypothèse H3 repose en particulier sur la nature du problème considéré et des algorithmes qui permettent de le résoudre, ainsi que sur leur place dans les programmes scolaires. En effet, le problème de recherche d'élément dans une liste triée peut être qualifié de fondamental pour le domaine de l'algorithmique : c'est un cas d'école par sa simplicité, mais son immense importance pratique en fait également un ingrédient essentiel dans un grand nombre d'applications. Par ailleurs, ses propriétés sont bien comprises, et abordables avec un bagage mathématique modéré. Enfin, tout comme le problème du tri, il s'agit d'un exemple de cas où l'on sait démontrer l'existence d'algorithmes *optimaux*, dans un sens que nous préciserons.

D'un point de vue plus didactique, nous défendrons à la section 5.3 l'idée que l'exploitation de ce problème dans l'enseignement peut être propice à l'émergence

chez les élèves des notions visées, en complément (au collège) ou en conformité avec les exigences immédiates des programmes (au lycée et après). Par ailleurs l'algorithme de dichotomie, qui joue un rôle central dans l'étude de ce problème, est également identifié comme un contenu visé par les programmes.

L'objectif de ce travail est de commencer à mettre à l'épreuve ces hypothèses, en élaborant tout d'abord une proposition d'activité d'enseignement autour du problème de la recherche d'élément dans une liste triée.

4 Recherche d'élément dans une liste triée

Dans cette section, nous présentons quelques éléments théoriques qui sous-tendent la situation d'apprentissage dont il sera question à la section 5.

On considère une suite finie indexée $L = (a_1, \ldots, a_n)$ – c'est à dire une liste – d'objets comparables deux à deux selon une relation d'ordre $\prec$. On note $L(i) = a_i$ le i-ème élément de L. On suppose en outre que ces listes sont *triées* selon $\prec$, c'est-à-dire que si $i < j$ alors $L(i) \prec L(j)$. On se place dans un modèle de calcul où les seules opérations disponibles sont des questions de la forme C_i : « Comment se comparent e et $L(i)$? » que nous appellerons simplement « opérations de comparaison », où e est une valeur quelconque, et $i \in \{1, ..., n\}$ est un indice valide de la liste L. Les réponses possibles à cette question sont : $e \prec L(i)$, $e = L(i)$ ou $e \succ L(i)$. On considère le problème fondamental suivant :

Problème 1 (recherche dans une liste triée)
Instance : *couple* (L, e)*, où* $L = (a_i)_{1 \leq i \leq n}$ *est une suite non vide et croissante selon la relation d'ordre* $\prec$*.*
Résultat : $i \in \{1, \ldots, n\}$ *tel que* $a_i = e$*, ou* 0 *si* $e \notin L$*.*

4.1 Algorithme et borne supérieure

On remarque qu'un algorithme naïf de parcours exhaustif, comparant e tour à tour à chacun des $L(i)$, résout ce problème :

Algorithme 1 (recherche exhaustive)
RechercheExhaustive***(L, e) :*** *Pour* i *allant de* 1 *à* n*, déterminer si* $a_i = e$*. Si c'est le cas, répondre* i*. Sinon, continuer l'itération. Répondre* 0 *si l'itération se termine sans succès.*

On voit donc que n questions de type C_i peuvent suffire à résoudre le problème à tous les coups. On dit alors que la complexité du *problème* 1 est majorée par $n = |L|$ dans le modèle de calcul considéré. Cette majoration est cependant très grossière, et il existe un algorithme de complexité bien meilleure :

Algorithme 2 (recherche binaire [6])
RechercheBinaire***(L, e) :***
- *On pose* n *la longueur de* L*,* $g = 1$ *et* $d = n$*.*
- *Tant que* $g \leq d$*, faire ce qui suit :*

- *On pose* $m = (g + d)/2$ *(arrondi à l'entier inférieur) ;*
- *Si* $a_m = e$*, l'algorithme répond* m *et s'arrête ;*
- *Sinon, si* $a_m < e$*, on pose* $g = m + 1$ *;*
- *Sinon, on pose* $d = m - 1$*.*

— *Si l'on sort de la boucle avec* $g > d$*, l'algorithme répond* 0 *et s'arrête.*

La complexité dans le pire cas de cet algorithme est bien connue :

Theorem 1. *Sur une instance* (L, e) *telle que* L *contient* n *éléments, l'algorithme 2 effectue dans le pire cas* $\lfloor \log_2(n+1) \rfloor$ *opérations de type* C_i*.*

La notation $\lfloor x \rfloor$ utilisée ici désigne la partie entière de x, pour x un réel positif. On peut démontrer ce théorème par récurrence sur n, ou plus informellement observer qu'à chaque itération le nombre de réponses encore possibles (qui sont les entiers de l'intervalle fermé $[g, d]$) est diminué de 1 et divisé par deux, et que l'algorithme s'arrête au plus tard quand cet intervalle devient vide. Quelle que soit la réponse attendue, l'algorithme ne peut donc pas effectuer plus de $\log_2(n+1)$ itérations, et donc comme chaque itération effectue une et une seule opération C_i, pas plus de $\log_2(n+1)$ opérations de comparaison en tout. Un examen attentif montre qu'il existe pour tout n des cas où ce nombre d'opérations est également *nécessaire*. On en conclut que la complexité dans le pire cas de l'algorithme de recherche binaire est précisément égal à cette quantité.

L'existence de l'algorithme 2 montre que la complexité du problème 1 est majorée par $\log_2(n+1)$. Il est en fait possible de démontrer que ce problème ne *peut pas* être résolu à coup sûr en moins de $\lfloor \log_2(n+1) \rfloor$ opérations de comparaison sur une liste de longueur n.

4.2 Borne inférieure par argument d'adversaire

Comme annoncé, nous allons établir le résultat suivant :

Theorem 2. *Dans le modèle de calcul considéré (opérations de comparaison de type* C_i*), tout algorithme résolvant le problème 1 effectue au moins* $\lfloor \log_2(n+1) \rfloor$ *opérations de comparaison dans le pire cas.*

En conjonction avec le théorème 1, ceci implique le corollaire suivant :

Corollary 1. *Dans le modèle de calcul considéré, l'algorithme 2 de recherche binaire est optimal pour le problème 1.*

Démontrer l'existence d'une borne inférieure sur la complexité d'un problème (et non d'un algorithme particulier) est un exercice un peu délicat. En effet, il faut raisonner de manière générale sur la complexité de *n'importe quel* algorithme le résolvant. On veut en fait démontrer que le meilleur algorithme possible pour

6. Plutôt que « recherche par dichotomie », nous choisissons une appellation proche de l'appellation anglophone pour éviter une possible confusion avec la méthode de dichotomie en mathématiques. Voir [7] pour une discussion à ce sujet.

ce problème doit effectuer, dans certains cas, au moins un certain nombre d'opérations. Ceci n'a de sens que si l'on fixe au préalable un modèle de calcul précis (ce que nous avons pris soin de faire au paragraphe précédent).

Nous allons démontrer ce résultat par un argument appelé argument d'adversaire. Pour cela, on se place dans une situation imaginaire où l'algorithme n'a pas accès directement aux valeurs des éléments qui constituent la liste L, mais est confronté à un environnement antagoniste, appelé *adversaire*, dont le rôle est de fournir les résultats de chaque opération de comparaison de type C_i effectuée.

La taille de la liste L dans les instances (L, e) considérées étant fixée, l'adversaire a pour objectif de pousser l'algorithme à effectuer le plus grand nombre possible d'opérations. En d'autres termes il cherche à « orienter » l'exécution vers l'un des pire cas pour cette taille d'instance. Pour cela, l'adversaire choisit la réponse à apporter à chaque requête de l'algorithme, avec la seule contrainte de ne jamais fournir d'informations contradictoires.

Voici une stratégie possible pour l'adversaire, dans le cadre du problème 1. L'adversaire maintient pour son propre compte une mémoire des bornes (g, d) de l'intervalle des positions de L pour lesquelles l'algorithme ne possède encore aucune information. À chaque nouvelle opération C_i effectuée par l'algorithme, l'adversaire répond selon la procédure suivante :

Algorithme 3 (Stratégie d'adversaire.)
RÉPONSE*(i, g, d) :*

- *Si $i < g$, répondre « $e \succ L(i)$ » ;*
- *Sinon, si $i > d$, répondre « $e \prec L(i)$ » ;*
- *Sinon, si $i \leq (g + d)/2$, poser $g = i$ et répondre « $e \succ L(i)$ » ;*
- *Sinon, si $i > (g + d)/2$, poser $d = i$ et répondre « $e \prec L(i)$ ».*

On peut montrer qu'au fur et à mesure des appels à RÉPONSE(i, g, d), on a toujours $g \leq d$, les valeurs successives de g croissent et celles de d décroissent. D'autre part, l'adversaire n'a jamais répondu « $e \prec L(i)$ » pour un indice i inférieur à g, ni « $e \succ L(i)$ » pour un indice i supérieur à d. L'adversaire ne contredit donc jamais une de ses réponses précédentes.

Supposons maintenant qu'un algorithme, confronté à cet adversaire, propose un résultat après avoir effectué strictement moins de $\lfloor \log_2(n+1) \rfloor$ comparaisons. On peut montrer que dans ce cas, $g \neq d$. Quel que soit le résultat avancé par l'algorithme, l'adversaire peut alors le mettre en échec en exhibant soit une liste triée L' dans laquelle $L(g) = e$, $L(i) \succ e$ pour tout $i > g$ et $L(i) \prec e$ pour tout $i < g$, soit une liste L'' dans laquelle $L(d) = e$, $L(i) \succ e$ pour tout $i > d$ et $L(i) \prec e$ pour tout $i < d$. Ces deux listes sont cohérentes avec toute l'information précédemment fournie par l'adversaire, mais la réponse de l'algorithme ne peut être juste sur chacune d'elles. Ceci permet de conclure que tout algorithme résolvant le problème 1 doit poser au moins $\lfloor \log_2(n + 1) \rfloor$ questions sur certaines instances (L, e) où L est de taille n.

5 Situation d'apprentissage autour des jeux de recherche

Dans cette section, nous présentons un schéma de situation d'apprentissage de type *informatique débranchée*, dont nous faisons l'hypothèse qu'il peut s'instancier à différents niveaux scolaires, et qu'il est susceptible de constituer une situation fondamentale pour la notion de complexité de problème.

Le terme *informatique débranchée* fait référence à un courant éducatif apparu dans les années 1990 mais d'essor relativement récent (voir par exemple [4,1,12], ainsi que les travaux de plusieurs groupes IREM [7]) dont l'objet est l'élaboration de situations d'apprentissage de l'informatique sans recours à des dispositifs électroniques (ordinateurs, robots, etc.) mais plutôt à du matériel tangible « traditionnel ».Ce courant a été à l'origine de la création d'un grand nombre d'activités portant sur des sujets variés, des plus théoriques aux plus appliqués et s'adressant à tous les niveaux d'apprentissage. Parmi ces activités, le jeu à un ou plusieurs joueurs tient un rôle particulier, comme c'est le cas dans d'autres disciplines comme les mathématiques.

5.1 Présentation générale de la famille de situations

Déroulement du jeu et conditions de victoire. Dans sa version la plus élémentaire, le jeu que nous considérons implique deux rôles, le *devinant* et le *répondant*. Le devinant cherche à déterminer un nombre entre 1 et n, en proposant un par un des candidats $i \in \{1, \ldots, n\}$. Le répondant indique, pour chaque proposition du devinant, si le nombre cherché est égal au candidat proposé (réponse dite « positive »), s'il est plus petit ou s'il est plus grand (réponses « négatives »). La partie s'arrête dès que le répondant donne une réponse positive.

Un objectif secondaire du devinant est d'obtenir une réponse positive en proposant le moins de candidats possibles, tandis que le répondant cherche à lui en faire poser le plus possible. Ceci est concrétisé par un nombre k convenu à l'avance : si le répondant donne une réponse positive après moins de k propositions, le devinant gagne la partie. Si plus de k candidats ont été proposés, le répondant gagne la partie. Si exactement k questions ont été posées, la partie est nulle. On considérera en outre que la partie est immédiatement gagnée par le devinant si le répondant fait une réponse contradictoire.

Lien avec le problème de la recherche triée. La situation telle qu'elle est proposée au joueur devinant peut être formulée comme la recherche d'un élément e (de valeur non directement accessible à l'algorithme) dans la liste contenant les entiers de 1 à n dans l'ordre. L'objectif d'apprentissage, suscité en particulier par le choix des conditions de victoire, est d'une part de faire émerger une stratégie de recherche efficace chez le devinant (l'algorithme de recherche binaire présenté au paragraphe 4.1) et une stratégie de réponse efficace chez le répondant (la stratégie d'adversaire présentée au paragraphe 4.2), et d'autre part de susciter

7. IREM : Institut de Recherche sur l'Enseignement des Mathématiques (http://www.univ-irem.fr/). Voir par exemple les IREM de Grenoble ou Clermont-Ferrand.

une réflexion sur les issues possibles des parties en fonction de la valeur du paramètre k, autrement dit sur la complexité exacte du problème.

Pour des élèves ou étudiants plus avancés, l'étude peut aller jusqu'à une mise en situation mettant en jeu la formulation générale du problème de la recherche d'élément dans une liste triée, la programmation des stratégies de chacun des deux rôles, la recherche d'une expression générale de k en fonction de n, voire jusqu'à une exigence de démonstration, mais ce ne sont évidemment pas des objectifs pour les plus petites classes. Selon les cas, les objectifs d'apprentissages secondaires peuvent donc inclure des éléments de programmation, la maîtrise des fonctions $\log_2(n)$ et/ou 2^n, la recherche et la résolution d'un problème mathématique, la preuve (par récurrence ou par invariant), etc.

5.2 Exemple de mise en œuvre

Voici un exemple d'instanciation de cette situation, pour des élèves plutôt jeunes (cycle 3 par exemple, élèves de 9 à 12 ans environ). Le dispositif peut être adapté selon un grand nombre de variables, dont nous détaillerons un certain nombre dans le paragraphe suivant.

Temps 1 : On fixe $n = 15$ et $k = 4$. Les élèves jouent par trois, un devinant, un répondant et un arbitre. L'arbitre demande à l'élève répondant d'inscrire un nombre sur une feuille de papier, et place cette feuille à l'écart face cachée. Au cours de la partie, il note chaque proposition du devinant et contrôle la véracité des réponses du répondant. À l'issue de la partie il compte le nombre de propositions, dévoile la feuille de papier préalablement cachée, et désigne le vainqueur (non par son prénom mais par son rôle, devinant ou répondant). Les rôles changent et les élèves recommencent jusqu'à échéance du temps imparti.

Temps 1' : On confronte les scores des deux rôles dans chacun des trinômes, et on met en commun les stratégies de recherche trouvées. Une question est posée : le joueur devinant est-il certain de pouvoir trouver en 4 coups ou moins ? Quelques parties sont jouées entre l'enseignant (dans le rôle de devinant) et les élèves de la classe (en concertation).

Temps 2 : On conserve $n = 15$ et $k = 4$. Le dispositif est le même qu'au temps 1, mais on ne demande plus au répondant de consigner un nombre à l'avance. L'arbitre vérifie seulement que les réponses sont cohérentes entre elles. Si nécessaire, l'enseignant indique que le répondant n'est plus obligé de choisir un nombre, du moment qu'il ne se contredit pas.

Temps 2' : On confronte les scores des deux rôles dans chacun des trinômes, et on met en commun les stratégies de réponse trouvées. Une question est posée : le répondant est-il certain de pouvoir forcer une partie nulle ou de gagner ? Quelques parties sont jouées entre l'enseignant (dans le rôle de répondant) et les élèves de la classe (en concertation), sous le contrôle d'un arbitre (toutes les parties devraient être nulles si les élèves appliquent une stratégie de recherche binaire correcte, et gagnées par l'enseignant sinon).

Temps 3 : On dresse un bilan des observations, en cherchant à expliquer le choix du paramètre $k = 4$. Que se passe-t-il si l'un des deux joueurs dévie des stratégies mises au point ? Que se passerait-t-il si l'on choisissait un nombre k plus petit ou plus grand ? Quel valeur faudrait-il choisir pour k pour garantir que toutes les parties soient nulles quand $n = 7$? Quand $n = 31$? Quand $n = 20$?

5.3 Analyse de la situation proposée

Le dispositif central est la phase de jeu antagoniste illustrée par le temps 2 décrit ci-dessus : les élèves jouent l'un contre l'autre sous le contrôle d'un arbitre et cherchent à gagner la partie. Il s'agit donc d'une situation *adidactique* au sens de la théorie des situations didactiques [3,2]. On peut arguer du fait que cette situation est fondamentale pour la notion de complexité de problème, au sens où son émergence est requise pour établir l'optimalité de la stratégie des joueurs.

En effet, l'enjeu de la situation est de discuter la possibilité pour chaque acteur (algorithme ou adversaire) de gagner ou de faire partie nulle *à tous les coups*. Acquérir la certitude de ne jamais perdre nécessite donc de la part de l'élève qui joue l'un des deux rôles d'avoir pris conscience du caractère indépassable de la borne k utilisée au cours du jeu : indépassable par l'algorithme, qui ne peut espérer gagner en moins de k questions, et indépassable pour l'adversaire, qui ne peut forcer l'algorithme à poser plus de k questions. Cette situation de status quo est précisément le résultat d'optimalité recherché : on a montré que le problème n'était pas résoluble en moins de k questions, et on a trouvé un algorithme qui peut atteindre cette borne inférieure.

On remarque que la stratégie initiale de recherche exhaustive ne permet pas au devinant d'atteindre son objectif, ce qui justifie l'élaboration d'une stratégie plus avancée. De même, la version du dispositif dans laquelle le répondant se contraint à ne pas changer de nombre ne lui permet pas de garantir un match nul. Par ailleurs, cette situation a un caractère fondamental au sens où elle autorise une grande variaté d'implémentations différentes selon les valeurs données aux variables didactiques qui la constituent. En voici quelques exemples :

Choix de n. L'exemple ci-dessus propose au devinant de déterminer un nombre entre 1 et $n = 15$. Dans ce cas, les propositions successives d'un joueur appliquant strictement la stratégie de recherche binaire sont faciles à anticiper (d'abord 8, puis 4 ou 12, puis 2, 6, 10 ou 14, et enfin une des positions impaires). D'autres choix possibles pour n sont (1) une autre valeur de la forme $2^p - 1$ avec $p \geq 0$; (2) une valeur quelconque. Un choix de type (1) autorisera (ou non) la possibilité de pré-calculer ou de calculer de tête toutes les suites de propositions possibles selon la valeur du nombre choisi. Un choix de type (2) complique la recherche en introduisant la nécessité de procéder à des arrondis. Dans les deux cas, une valeur plus grande de n change aussi la difficulté de la stratégie du répondant, qui doit à chaque tour calculer le nombre médian parmi les candidats restants.

Choix de k. Si l'on suppose que n est compris entre 2^{p-1} et $2^p - 1$ avec $p \geq 1$, l'analyse préalable du problème nous apprend que la borne inférieure de complexité est égale à p comparaisons. Ainsi, en fixant $k = p$, deux joueurs suivant

rigoureusement leur stratégie optimale peuvent forcer une partie nulle à chaque fois. Choisir une valeur différente de k introduit un déséquilibre dans les chances de gain des deux joueurs. Une valeur plus petite de k condamnera le devinant à perdre toutes ses parties contre un répondant optimal, et vice-versa. Une manipulation de cette variable met donc en jeu la capacité des élèves à détecter ce déséquilibre et à prévoir les issues des parties.

Statut des joueurs. plusieurs combinaisons de statuts des joueurs sont envisageables. On a déjà évoqué la possibilité pour l'enseignant d'assumer un des rôles face à un élève ou un groupe d'élèves. Il est également possible de faire intervenir un joueur artificiel (programme informatique) jouant parfaitement.

Nature de l'espace de recherche. La formulation générale du problème de recherche dans une collection triée autorise a priori d'autres espaces de recherche qu'un simple intervalle de nombres. On peut envisager toute collection d'éléments comparables deux à deux, triée vis-à-vis d'un ordre quelconque mais non nécessairement constituée d'éléments consécutifs.

Nature de la question. De même, il est possible de modifier la question exacte posée à l'élève devinant. Par exemple, on peut le charger de chercher un indice i tel que le $L(i) = i$, ou encore le plus petit i tel que $L(i) \geq e$.

Une forme un peu plus éloignée, qui rappelle le problème de la méthode de bissection en analyse, demande étant donnés une fonction croissante f, un entier n et un entier e de chercher un entier $1 \leq i \leq n$ tel que $f(i) = e$. Ce cas peut servir de transition vers une étude de la méthode de bissection en elle-même, mais des écueils peuvent surgir dans le passage du discret au continu. Une discussion sur ce sujet est apparue dans [7].

Matériel de jeu. Plusieurs types de matériel de support sont possibles. Pour favoriser une visualisation de l'espace de recherche, on pourra par exemple munir les joueurs d'une bande numérique plastifiée et de feutres effaçables, leur permettant de délimiter l'intervalle de recherche restant à explorer. Pour de grandes valeurs de n ou pour rechercher par tâtonnement la valeur de la borne de complexité, on peut également fournir une calculatrice ou un ordinateur. L'impact exact de ce choix d'organisation reste à analyser.

6 Conclusion et perspectives

Nous avons présenté des hypothèses sur la possibilité d'aborder en classe, par le biais du jeu à deux joueurs, les notions de borne inférieure et de borne supérieure de complexité de problèmes algorithmiques. En particulier, nous avons présenté une situation d'apprentissage autour du problème de la recherche d'élément dans une liste triée, susceptible de permettre un grand nombre d'adaptations à différents niveaux de classe.

Nous souhaitons à court terme finaliser l'analyse a priori de cette situation d'apprentissage et mettre en œuvre un plan d'expérimentation permettant d'évaluer ses effets et la pertinence des variables didactiques identifiées. Nous envisageons de procéder à une expérimentation préalable à petite échelle dans trois

niveaux de classes différents : une classe de cycle 3 (de préférence élémentaire), une classe de cycle 4 et une classe de NSI. Il sera également nécessaire de poser la question de l'évaluation des apprentissages visés par cette activité, en particulier sur la notion de « coût » d'un algorithme, et le cas échéant de proposer des exemples d'évaluations. On pourra se référer en particulier aux futures épreuves de NSI du baccalauréat français ainsi qu'aux évaluations du supérieur, mais la question reste ouverte pour les plus petites classes.

Parmi les perspectives à plus long terme, nous souhaitons élargir l'étude à d'autres situations proches, en particulier autour des d'algorithmes de type « diviser pour régner ». Enfin, nous souhaitons continuer à explorer le thème des interactions entre mathématiques et informatique dans l'enseignement, d'un point de vue épistémologique et didactique. Nous pensons que le présent travail ouvre par exemple des questions intéressantes sur la fonction logarithme en mathématiques et en informatique, ainsi que sur le raisonnement et la preuve.

Références

1. Bell, T., Alexander, J., Freeman, I., Grimley, M. : Computer science unplugged : School students doing real computing without computers. The New Zealand Journal of Applied Computing and Information Technology **13**(1), 20–29 (2009)
2. Bessot, A. : Une introduction à la théorie des situations didactiques (master "mathématiques, informatique" de grenoble 2003-2004). Les cahiers du laboratoire Leibniz **91** (2003)
3. Brousseau, G. : Théorie des situations didactiques. La pensée Sauvage, Grenoble (1998)
4. Fellows, M., Koblitz, N. : Kid krypto. In : Annual International Cryptology Conference. pp. 371–389. Springer (1992)
5. Gal-Ezer, J., Vilner, T., Zur, E. : Teaching algorithm efficiency at cs1 level : A different approach. Computer Science Education **14**(3), 235–248 (2004)
6. Ginat, D. : Efficiency of algorithms for programming beginners. SIGCSE Bull. **28**(1), 256–260 (1996)
7. Meyer, A., Modeste, S. : Recherche binaire et méthode de dichotomie, comparaison et enjeux didactiques à l'interface mathématiques - informatique. In : Espace Mathématique Francophone (2018)
8. Ministère de l'Éducation nationale et de la Jeunesse : Programme de numérique et sciences informatiques de première générale. BO de l'éducation nationale (2019)
9. Ministère de l'Éducation nationale et de la Jeunesse : Programme de numérique et sciences informatiques de terminale générale. BO de l'éducation nationale (2019)
10. Modeste, S. : Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ? Ph.D. thesis, Université de Grenoble (2012)
11. Selby, C., Woollard, J. : Computational thinking: the developing definition. Project report, University of Southampton (2013)
12. Vincent, J.M., Collectif Tangente : Informatique débranchée. No. 42/43 in Tangente Éducation, Pole (2017)
13. Wing, J.M. : Computational thinking. Commun. ACM **49**(3), 33–35 (2006)