Évaluations de solutions d'exercices d'algorithmique " à la main " versus " automatiques par jeux d'essai "

Denis Bouhineau, François Puitg

▶ **To cite this version:**

Denis Bouhineau, François Puitg. Évaluations de solutions d'exercices d'algorithmique " à la main " versus " automatiques par jeux d'essai ". EIAH 2011 - Conférence Environnements Informatiques pour l'Apprentissage Humain, May 2011, Mons, Belgique. pp.273-285. ⟨hal-00667472⟩

HAL Id: hal-00667472

https://hal.science/hal-00667472v1

Submitted on 7 Feb 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

Évaluations de solutions d'exercices d'algorithmique

« à la main » versus « automatiques par jeux d'essai »

Denis Bouhineau*, François Puitg*

** Laboratoire d'Informatique de Grenoble (LIG)*
Université de Grenoble (Grenoble-I, Univ. J. Fourier)
Denis.Bouhineau@imag.fr, Francois.Puitg@imag.fr

RÉSUMÉ. *Nous avons entrepris une analyse qualitative portant sur l'activité de correction de copies d'algorithmique « à la main » pour servir de repère et permettre une comparaison avec une correction automatique de productions d'étudiants en algorithmique en utilisant des jeux d'essai envisagée pour la mise en place d'un EIAH de l'algorithmique comportant une base d'exercices. La mise en place de la base d'exercices et des jeux d'essai est également rapportée. La communication comporte également quelques éléments relatifs à la motivation des étudiants et la remédiation des erreurs dans une démarche semi-automatique associant les enseignants.*

MOTS-CLÉS : *algorithmique, programmation, Prolog, programmation déclarative, récursivité, évaluation, auto-évaluation, indicateur, diagnostic, rétroaction, correction de copies, correction automatique, jeux d'essai, test, mise en place, collaboratif, acteur, apprenant, enseignant.*

Introduction

Évaluer les productions d'élève, « corriger des copies », pour vérifier, voire certifier des apprentissages, orienter la suite des cours et des apprentissages, … est une pratique professionnelle usuelle mais complexe et lourde du point de vue des enseignants correspondant souvent à un vécu difficile pour les élèves [Veslin & Veslin - 92]. Elle peut prendre des noms variés et avoir des objectifs contradictoires : depuis l'évaluation d'embauche ou les concours avec des visées de sélection et de classement élitiste ; en passant par l'évaluation pronostique en début de cycle ou l'évaluation sommative ou certificative en fin de cycle, pour le brevet des collèges ou le baccalauréat ou en fin de semestre à l'université ; jusqu'à l'évaluation formative en cours de formation pour donner un retour à l'apprenant sur le niveau qu'il a atteint ou déstabiliser ses conceptions erronées et permettre à l'enseignant de prendre le pouls de sa classe pour organiser la suite de ses enseignements. Souvent, pour se restreindre à l'école et au travail quotidien dans les classes, l'évaluation est une pratique qui vise à comparer la production d'un élève avec une solution, ou une famille de solutions de référence, correspondant aux attentes du professeur. Malgré les efforts d'objectivité engagés depuis 1950-60, c'est toujours lié à un contrat didactique. *C'est une activité subjective*. Plus récemment, les chercheurs et pédagogues en évaluation ont cherché une plus grande précision descriptive pour les démarches évaluatives. En cernant des notions indépendantes d'un enseignant ou d'une matière enseignée, mais liées au savoir plus généralement, à cheval sur plusieurs matières éventuellement et donc enseignées par plusieurs enseignants, dans un référentiel commun à l'ensemble des enseignements (cf. le socle commun), sont donc apparues les *compétences* donnant lieu à une nouvelle forme d'évaluation, l'évaluation par compétences [Gérard - 09]. D'aucuns verront là l'influence de travaux en informatique ou des sciences cognitives pour

diagnostiquer objectivement et séparément des capacités et compétences d'élèves en cernant autant d'éléments que nécessaire. L'histoire des formes d'évaluation n'est pas finie, gageons qu'il y aura d'autres formes d'évaluation à venir.

Considérons maintenant les EIAH. La question de l'évaluation des productions d'élèves, si elle n'est pas aussi rébarbative que la correction de copies (dans l'hypothèse où elle peut être automatisée et que ce peut être l'ordinateur qui s'attaque aux piles de copies), elle est, pour les concepteurs d'EIAH, tout aussi complexe et, pour les enseignants et les apprenants, porteuse d'autant d'attentes que dans la classe. Un espoir, comme l'informatique en a fait naître de nombreux, serait que cette évaluation soit effectivement pleinement confiée à l'ordinateur, et libère l'enseignant de ces longues séances de correction de copies. À noter que cela serait au détriment du retour que l'enseignant obtient en corrigeant ses copies pour organiser la suite de son cours et la connaissance qu'il acquiert ainsi directement sur le profil de chaque élève. L'objectivité de la correction ne serait pas pour autant garantie. En effet, l'enseignant, à travers ses choix pour paramétrer ou orienter la correction automatique, et l'informaticien dans ses mises en œuvre introduisent leur subjectivité. Mais l'espoir d'une correction complètement automatisée est encore loin. Le plus souvent, c'est plutôt l'inverse qui s'opère. L'évaluation automatique étant une affaire délicate, le plus simple pour un EIAH naissant, ou une plateforme d'enseignement moderne (i.e. informatique, accessible par internet, …) se voulant générique, consiste à proposer des activités qui s'évaluent facilement de manière générique (QCM, Phrases à trous, etc. [HotPot]) ou par le biais d'appels aux enseignants pour effectuer ces évaluations [Moodle]. Certaines matières cependant s'en sortent mieux. Ainsi, pour les sciences dures, il peut sembler plus facile de trouver des modalités de validation automatique des réponses d'élèves sans en être réduit à des activités « fermées ». C'est le cas pour une grande partie de la géométrie [Bouhineau - 1997] ou de l'algèbre [Bouhineau & al. - 03] comme le projet Aplusix a su l'exploiter. Notons, au passage, pour tirer quelques leçons de l'expérience du projet Aplusix, que l'évaluation automatique de la progression ou de la validité d'une réponse d'élève par l'ordinateur permet plusieurs déplacements vis-à-vis de l'évaluation dans la classe :

- automatique, elle peut être calculée plusieurs fois et pas uniquement à la fin de l'exercice, donnée à chaque instant ou presque au cours de l'exercice, elle peut devenir une rétroaction permanente non intrusive, guidant la résolution de l'exercice.
- L'évaluation ayant perdu son caractère négatif (conclusif et définitif), elle peut changer de statut et s'inscrire parmi les étayages, au delà des évaluations formatives.
- La disponibilité de cette évaluation est un nouveau paramètre pédagogique. L'enseignant peut jouer sur ce paramètre, lui donner un coût, pour renforcer des apprentissages au niveau des contrôles et des stratégies. Sous forme de rétroaction permanente gratuite, cela peut dispenser l'élève de développer ses propres contrôles ; coûteuse ou limitée, elle impose à l'élève un effort pour atteindre le même résultat et favorisera des apprentissages stratégiques.
- Dans la mesure où l'étudiant ne se sert pas d'une rétroaction permanente pour « tricher », et que la possibilité d'une avancée par essais (aléatoires)/erreurs est faible, ce que l'évaluation perd, elle ne le perd pas en efficacité et ne fausse pas les règles de l'exercice, mais le statut de l'erreur change, il y a un « après » l'erreur où l'apprenant change celle-ci en réponse juste, l'élève a ainsi le droit aux essais et aux erreurs.
- Enfin, l'évaluation automatisée favorise l'autonomie des élèves.

Peut-être est-ce lié, l'utilisation d'une évaluation automatique des réponses d'élèves dans le cadre du projet Aplusix a permis d'observer non seulement un glissement du statut de l'erreur mais aussi un glissement du statut de celui qui l'annonce (sous l'effet *Cassandre*) :

- la confiance en l'ordinateur étant grande, quand celui-ci dit que la réponse est fausse, les discussions possibles entre enseignant et élève ou entre élèves sur la justesse d'une solution se terminent rapidement. Il faut cependant tempérer ce résultat, la confiance dans l'ordinateur peut disparaitre avec l'apparition de bugs visibles.
- L'ordinateur semble plus objectif, il ne semble pas répondre à un contrat didactique qui refuserait une réponse valide pour des critères pédagogiques. Cependant croire en l'objectivité de l'ordinateur c'est oublier que le programme utilisé est un artefact humain et que les programmeurs, même en ayant une volonté de réaliser un environnement neutre du point de vue didactique ou pédagogique, se concentrant uniquement sur les mathématiques, ont aussi, malgré tout et malgré eux, leurs a priori et les ont fait passer dans l'objet produit [Traglova & al. - 09].
- Dégagé de la responsabilité de l'annonce de l'erreur, l'enseignant (qui a gagné par ailleurs le temps de la vérification de la production de l'élève) est plus facilement, face à l'ordinateur, du même coté que l'élève, dans une position moins frontale pour apporter son savoir, son expertise, ou plus simplement dispenser son aide, proposer un conseil.

L'informatique en général et l'algorithmique en particulier sont sans nulle doute à classer parmi les sciences dures. L'obtention de méthodes automatiques de validation de productions d'élèves peut sembler envisageable. Cependant, il va falloir cerner quel type de production peut être validée et voir si l'on échappe aux écueils particuliers de l'informatique concernant les problèmes de calculabilité et d'indécidabilités découverts par Gödel et ses successeurs. En effet, en informatique, il existe de nombreux théorèmes limitant les espoirs de preuves automatiques. Ainsi est-il impossible de déterminer automatiquement dans le cas le plus général (c'est-à-dire par un programme-démonstrateur général fonctionnant pour toutes les données-programmes possibles) si un programme donné va s'arrêter ou non, si deux programmes sont équivalents ou non, etc. (ce qui légitime que les étudiants doivent faire des preuves à la main, de la correction et de la complétude de leurs programmes). Deux types d'évaluation sont menés habituellement [Skupiene - 10] : d'une part, une évaluation sémantique qui concerne le « fond », i.e. une évaluation pour assurer la correction des programmes, c'est-à-dire leur bonne adéquation pour la résolution d'un problème donné ; d'autre part, une évaluation plus syntaxique de la forme de ces programmes, i.e. le respect de normes de programmation concernant le nom des variables, l'emploi de commentaires, ainsi que d'autres éléments liés à l'algorithmique et aux structures de données et de contrôle employées et définissant des métriques du code.

La suite de cet article va se concentrer sur l'analyse du « fond », c'est-à-dire de la correction. Deux approches vont être comparées : celle « à la main » et celle « automatisée par jeux d'essai ». Pour cette dernière, des pistes de comparaison avec l'évaluation « à la main » seront évoquées, des questions de mise en œuvres seront soulevées : un outil d'évaluation automatique par jeux d'essai peut-il exister, et à quel coût ? Comment comparer cet outil à la pratique d'évaluation usuelle que les enseignants connaissent ? Le cadre de cette analyse qualitative est donné par la mise en place d'un EIAH de l'algorithmique (EDBA : Exercises DataBase about Algorithms) proposant un mode d'évaluation de productions algorithmiques par jeux d'essai.

1. Analyse à la main

Pour servir de support à une description du travail habituel de correction de copies d'algorithmique, quatre questions d'examen ou de contrôle continu de licence d'informatique (L3-IMA-MIAGE à l'Université Joseph Fourrier – Grenoble 1) ont été analysées avec un recul méthodologique pour une population d'environ 45 élèves par question, sujets étalées sur 3 ans, avec 2. Les questions sélectionnées portaient sur l'écriture de programmes élémentaires, premières questions d'un exercice comportant, en plus de la demande du programme, l'écriture d'une spécification associée à des jeux d'essai. La programmation logique (ou déclarative) avec contraintes était l'objet de l'enseignement dispensé, Prolog était le langage utilisé. L'enseignement abordait, entre autres, la question de la récursivité, pour un volume horaire de 12 semaines comportant 1h30 de cours et 1h30 de TD. Les étudiants, pour la plupart, découvraient ainsi un troisième paradigme de programmation et tous renforçaient leur connaissance dans le domaine de la récursivité. L'une des difficultés de cet enseignement venait du changement de paradigme de programmation, supposant l'abandon des mécanismes acquis en algorithmique avec les paradigmes précédents pour ceux ayant cours dans le paradigme présent, et les changements de vocabulaire associés (ex. : en Programmation logique, la notion de programme ou de fonction est remplacée par celle de prédicat). La maîtrise de la récursivité représentait l'autre difficulté, même si cette notion avait été abordée précédemment. C'est une notion difficile.

1.1. Corpus des sujets d'examen

Sujet 1 (examen 1er session, Déc. 2008, temps indicatif : 10 min) : cet exercice porte sur des listes de 0 et de 1, ex. : [0,0,1,0,1]. Spécifiez, réalisez et donnez des exemples d'utilisation d'un prédicat « rangListe » qui donne le rang du premier 1 dans une liste de 0 et de 1 (le premier élément de la liste est au rang 0).
Variante 2 : sur le dernier 1 (à la place du premier 1).
Sujet 3 (examen 1er session, Déc. 2009, temps indicatif : 10 min) : Spécifiez et réalisez un prédicat qui détermine le nombre N d'occurrences d'une valeur E donnée dans une liste L donnée également.
Variante 4 : une réponse booléenne pour indiquer la présence ou l'absence de 1 dans une liste.

1.2. Analyse a-priori

Les attentes des enseignants en algorithmique recouvrent en partie les productions des étudiants : lorsqu'il s'agit d'écrire des programmes (ou des algorithmes), tout va bien. Quand une question est plus molle et concerne la description d'un programme, l'écriture d'une spécification, l'évocation de schéma d'algorithme, les étudiants semblent oublier la question. Ex : pour le sujet 4, 23 copies sur 45 comportent uniquement le programme demandé, avec deux arguments, sans même une explication sur ces deux arguments. Centrons-nous donc sur les programmes. Les sujets proposés visaient l'évaluation des deux éléments constituant l'enseignement : la maîtrise du paradigme de programmation déclarative et la maîtrise de la récursivité. Les sujets proposés ne posaient pas de réelle difficulté sur le plan algorithmique.

- Pour la maîtrise du paradigme de programmation, certains aspects syntaxiques permettaient d'apprécier la connaissance des principes de haut niveau mis en œuvre dans la programmation déclarative (disjonction des cas à travers la rédaction de différentes règles, conjonction des prédicats à travers la juxtaposition de ces différents prédicats dans la rédaction d'une règle, …). D'autres aspects syntaxiques plus immédiats, propres au langage Prolog associé à ce paradigme de programmation, pouvaient être évalués directement sur les productions (distinction entre constantes et variable, manipulation des listes, arithmétique, notion de relation (prédicat) vs fonction).
- Pour la maîtrise de la récursivité, il s'agissait de vérifier, au niveau conceptuel, la mobilisation de cette forme de programmation dans les différents sujets, la bonne compréhension du découpage en cas de base et cas de propagation (ou d'induction). Au niveau plus technique, la justesse des choix des différents cas pouvait être évaluée.

Pour l'évaluation de la maîtrise du paradigme de programmation et de la récursivité, une hiérarchie des erreurs était possible : erreurs conceptuelles, erreurs techniques (au niveau algorithmique), erreurs syntaxiques. La correction pouvait tenir compte de cette hiérarchie pour s'abstraire des erreurs de bas niveau afin d'évaluer les concepts de plus haut niveau.

1.3. Au cours de la correction

Que ce passe-t-il pendant la correction des copies ? Quels sont les processus mentaux mis en œuvre par le correcteur lors de cette correction ? C'est un sujet rarement abordé si ce n'est pour évoquer la complexité de la tâche et ses aspects fastidieux imparfaitement compensés par les retours que l'enseignant en obtient. Cette section sera donc assez courte. Notons, pour l'algorithmique, la mobilisation probable de plusieurs types de processus :

- des processus de contrôle syntaxique des productions,
- des processus de vérification formelle de la correction, de la complétude et de la terminaison des productions, quand la logique de la production existe et atteignable,
- des processus de validation par l'exemple, ou la recherche de contre-exemple, cherchant à analyser « par l'exécution » ces productions,
- des processus d'appréciation de l'élégance, de l'originalité des productions.

1.4. Retour des corrections

Pour l'essentiel, la correction de copies a apporté les éléments attendus : un lot d'erreurs syntaxiques (essentiellement pour la manipulation des listes et le prédicat d'évaluation effective d'expressions « is ») ; des faiblesses du coté des utilisations de la récursivité (oubli d'appels récursifs) ; une maîtrise globale satisfaisante du paradigme de programmation déclarative et parfois, pour quelques questions, une solution originale.

Pour les découvertes, non imaginées avant la correction, il y a une utilisation erronée récurrente de l'arithmétique. La partie arithmétique d'un programme est souvent une partie délicate, elle concentre les calculs du programmes, il est donc naturel qu'elle comporte des erreurs. Dans le cas de Prolog, le prédicat « is » présente plusieurs pièges : au niveau syntaxique, il ne se comporte pas comme les autres prédicats ; proche de l'affectation, ce prédicat ne doit pourtant pas lui être confondu (N is N+1 n'a pas de sens). Compte-tenu de ces difficultés, une attention particulière lui est consacrée lors de l'enseignement.

Cette étude de la pratique de correction de copies apporte aussi confirmation de la diversité algorithmique des solutions (et des erreurs) pour un problème donné. Plus exactement, elle précise que si les solutions à un problème donné ne sont pas uniques (il y a eu 2-5 solutions pour les sujets proposés), elles se concentrent tout de même essentiellement sur la ou les solutions attendues, à des variations syntaxiques mineures près, le tout représentant un tiers à la moitié des copies. Du coté des erreurs, la correction montre la grande dispersion des erreurs sur tout la gamme typologique (erreurs syntaxiques, erreurs algorithmiques, mauvaise compréhension du sujet, solution inachevée…) Elle montre que les erreurs offrent une diversité beaucoup plus grande que les solutions (il y a en général plus de 20 variantes erronées pour les sujets proposés, représentant l'autre partie des copies). Il ne semble pas que 40 copies suffisent à faire le tour de l'ensemble des propositions incorrectes possibles. Cette grande diversité s'explique par le grand nombre d'erreurs « atomiques » possibles (il semble qu'il puisse y en avoir une vingtaine par sujet ; au bout de 40 copies, le décompte de ces erreurs atomiques semble converger) et la combinatoire exponentielle des assemblages des erreurs entre elles. Remarque : déterminer si deux propositions (justes ou erronées) sont identiques n'est pas immédiat, il faut en outre définir ce que signifie l'identité, on peut par exemple prendre une identité modulo une classe d'équivalence donnée : par exemple modulo le nom des variables et quelques constructions syntaxiques équivalentes.

Considérant que les sujets proposés étaient très proches les uns des autres, il peut venir à l'esprit l'idée de construire une grille d'évaluation systématique descriptive et critériée de ce genre d'exercice. Les apports de ce type de grille d'évaluation sont, par exemple, une plus grande objectivité de la correction, une plus grande masse d'informations disponible pour l'étudiant. Cependant, ce travail n'est pas si immédiat comme [Scallon - 04] le signale. Il faut, de plus, prendre en compte le rapport entre le coût de cette mise en place et ce qu'elle va rapporter. Les sujets considérés ne représentaient que 10% (en moyenne) de l'évaluation ; ce serait peut-être une attention disproportionnée pour l'exercice. De plus, l'utilisation de ce type de grille d'évaluation est souvent associée à une correction nécessitant autant de passes que de critères étudiés, ce qui multiplie le temps de correction.

À noter : la relecture des copies, seconde correction-analyse, a pointé des erreurs (parfois volontaires) lors de la correction initiale. Dans tous les cas observés (2-3 cas pour chaque paquet de 45 copies), il s'agissait d'erreur de correction en faveur des étudiants : des erreurs d'étudiants n'avaient pas été vues. En général, il s'agissait d'oublis en faveur de bonnes copies.

2. Un exemple d'analyse automatique par jeux d'essai

L'analyse de programme par jeux d'essai est une pratique courante et ancienne en algorithmique et dans l'industrie du logiciel. Critiquée depuis longtemps pour vérifier la correction d'un programme, « Program testing can be used to show the presence of bugs, but never to show their absence! » [Dijkstra – 69], elle permet cependant d'avoir à faible coût une idée rapide d'une production algorithmique ; aussi est-elle utilisée dans la plupart des sites de concours d'algorithmique (ex : www.spoj.pl) ou les EIAH d'algorithmique [Douce & Al. – 05], en particulier pour les premiers apprentissages [Pears & Al. – 07]. Dans le cas des concours d'algorithmique ou des EIAH d'algorithmique, ces tests ont à valider une très

grande variété de programmes différents (même s'ils ont la même spécification !), pour la plupart comportant des erreurs, dont des erreurs immédiates. Des allers-retours entre tests et programmes peuvent avoir lieu au cours du temps, les premiers programmes pouvant servir à construire des tests pour les programmes suivants. La variante utilisée ici fait partie des tests dits de « boite noire », c'est-à-dire ne supposant pas la connaissance du programme testé (la spécification seule est supposée connue). Elle peut comporter des tests positifs, fonctionnels ou de conformité (test cherchant à vérifier les résultats attendus sur des données correctes, usuelles), des tests aux limites (pour des données correctes inhabituelles) et de tests négatifs, de vulnérabilité ou de tolérance aux fautes (avec des données incorrectes). Ils sont constitués de couples (donnée, résultat) correspondant aux spécifications et sont effectués en exécutant une production d'étudiant sur la donnée d'un test puis la comparaison des résultats effectifs avec les résultats de références. La version initiale de ces tests peut être rédigée à partir d'une analyse a priori du problème, selon les solutions attendues, les objectifs d'apprentissages visés et les erreurs courantes connues. Les versions suivantes des tests peuvent tirer profit des productions évaluées précédemment.

La passation d'un jeu d'essai suppose que la production à tester correspond à un programme suffisamment correct sur le plan syntaxique pour que son exécution soit possible et que la signature du programme corresponde à la spécification donnée pour les jeux d'essai. Le résultat de la passation d'un jeu d'essai sur une production est avant tout binaire : le test est passé correctement si les résultats obtenus correspondent aux résultats de référence (un test d'égalité souple peut être mis en place pour permettre des fluctuations entre les résultats obtenus et les résultats de référence : ordre sur les résultats indifférent, noms des variables internes indifférent, …). Lors de la passation d'un ensemble de jeux d'essai, le résultat est sous la forme d'un vecteur de résultats binaires. Sur une production d'étudiants ayant passé tous les tests correctement, il ne peut être affirmé positivement que la production est correcte, mais sur une production ayant échoué à un test (même à un seul), il est possible d'affirmer que la production comporte une erreur ou n'est pas optimale (dans l'hypothèse où les tests sont eux-mêmes justes).

Exemple de jeux d'essai pour le sujet 3 (a-priori), avec la spécification suivante, nombreDOccurence(E,L,N) est vrai ssi le nombre d'occurrence de E dans la liste L est N :

Jeux d'essai :	Donnée (E,L,N) :	Résultat :
3.1	(1,[],0)	true
3.2	(0,[],1)	fail
3.3	(2,[0,1,1,0,1,1,0],R)	R = 0;
3.4	(0,[0,1,1,0,1,1,0],R)	R = 3;
3.5	(1,[0,1,1,0,1,1,0],R)	R = 4;
3.6	(1,[0,1,1,1,0,0,1],R)	R = 4 ;
3.7	(0,[0,1,1,1,0,0,1],R)	R = 3;

2.1. Analyse par jeux d'essai vs/en appui d'une analyse à la main

Analyse à la main de copies d'examen et analyse par jeux d'essai sont deux outils très différents dans leur mise en œuvre, pour les objets sur lesquels ils s'appliquent et dont les enseignants et les étudiants peuvent attendre des retombées différentes. D'un coté, un processus expert humain *lent* sur une production papier *définitive riche* ; de l'autre coté une

exécution *rapide* d'un test d'égalité pouvant être effectué sur des productions *intermédiaires* pour obtenir des rétroactions immédiates au cours d'un travail de rédaction. Prenons pour hypothèse que les productions écrites sujettes à une correction manuelle de la section 1 soient disponibles sous format électronique et puissent subir des tests par jeux d'essai.

La correction « à la main » visait la vérification de la bonne compréhension du paradigme de programmation déclarative d'un coté et de la récursivité de l'autre coté. Pour la validation de l'apprentissage de la programmation Prolog, l'analyse par jeux d'essai fournit un verdict rapide pour les aspects syntaxiques : tant que la production de l'étudiant n'est pas suffisamment correcte d'un point de vue syntaxique et ne permet pas son exécution, l'exécution des jeux d'essai est impossible. Heureusement, d'après l'étude « à la main », l'apprentissage de la syntaxe Prolog est assez bonne (la syntaxe prolog est assez simple). Le nombre de copies mises à l'écart est donc faible. Pour le correcteur humain, ces copies peuvent cependant être étudiées en faisant abstraction des erreurs de syntaxes. D'après l'expérience en TP de programmation, nous pouvons ajouter que les erreurs de syntaxe sont rarement indépassables pour les étudiants, dès l'instant que ces erreurs ont été signalées. Nous pouvons donc supposer que le diagnostic automatique est adéquat. Observons la validation de l'apprentissage de la récursivité. « À la main », il s'agit d'observer les cas de base et les cas de propagation. L'ensemble des jeux d'essai retrouve cette séparation entre cas de base et cas de propagation en proposant des tests correspondant aux cas de base (tests positifs et tests négatifs), et des tests nécessitant des cas de propagation (tests positifs et tests négatifs). Les appels récursifs se terminant toujours par des appels aux cas de base, il est clair que la correction des cas de base sera nécessaire à l'évaluation des cas de propagation. Prenons un exemple. Pour le sujet 3, les tests concernant des cas de base sont potentiellement les tests 3.1 et 3.2. Ces deux tests correspondent au cas de base attendu sur une liste vide. Le seul test 3.1 n'aurait pas suffit, car il ne permet pas de mettre en défaut le cas de base erroné trouvé dans plusieurs copies « nombreDOccurence(E, [], N). ». En effet, la réponse à l'exécution sur le test 3.1 est correcte sur ces deux cas de base. L'étude « à la main » des copies a montré l'utilisation d'un cas de base sur la liste singleton [E]. Ce cas de base est moins judicieux, car il n'exonère pas l'étudiant de l'écriture du cas de base sur liste vide. Par ailleurs, si l'étudiant met les deux, il obtient des résultats redondants. L'ajout de jeux d'essai pour ce cas de base pourrait être judicieux. Le choix des jeux de tests est délicat, il peut aider à cerner quelques points, mais le nombre de tests aura tendance à augmenter à mesure que l'on veut cerner avec plus de précision différents cas de figures. Cependant, la combinatoire n'est pas nécessairement en défaveur du testeur, avec N tests binaires, 2^N cas de figures différents peuvent être isolés. L'étude des cas réels a montré la très grande dispersion des erreurs. Le coût pour distinguer l'ensemble de ces cas réels peut donc s'avérer tout de même important. Par ailleurs, la connaissance des tests peut engendrer des failles et permettre à des étudiants de frauder (quelque soit le nombre de ces tests).

D'un autre coté, regardons ce qu'apporte la correction automatique. En continuant avec l'exemple du sujet 3 (avec le jeu de tests initial), l'analyse automatique par jeux d'essai coupe le paquet de copies en 5 tas (avec une analyse après coup des tas obtenus) :

- Avec 0 test passé correctement (sur 7 tests, 5 copies) : Les copies ayant un problème important de syntaxe empêchant l'exécution des tests
- Avec 1 test passé correctement (sur 7 tests, 18 copies) : les copies n'ayant pas résolu le problème (cas de base incorrect)

- Avec 2 et 3 tests passés correctement (sur 7, tests, 7 copies) : les copies ayant un cas de base satisfaisant, mais une propagation incorrecte
- Avec 5 tests passés correctement (sur 7 tests, 3 copies) : les copies ayant une solution juste mais pas optimale (cas de redondance dans les cas de base).
- Avec 7 tests passés correctement (sur 7 tests, 10 copies) : les copies parfaitement justes.

L'analyse par jeux d'essai n'a pas mis en évidence de production vérifiant 4 ou 6 tests correctement (et seulement 4 ou seulement 6). Ce qui suggère une structure implicite engendrée par le choix des jeux d'essai. De même, il est intéressant de constater que pour les productions d'étudiants ayant validé seulement 2 tests, ces 2 tests sont toujours exactement les mêmes : 3.1 et 3.2. Idem pour le tas de production ayant validé 3 tests : les deux tests précédents et un test supplémentaire. Ceci conforte l'idée que le choix des jeux d'essai induit une structure sur les séparations possibles entre différentes productions. Cette structure regroupe certains tests comme *équivalents* pour un point de vue particulier et définit des relations d'inclusion entre ces groupes. La sémantique derrière ces regroupements et inclusions n'est pourtant pas évidente, mais elle semble justifier l'analyse a postériori effectuée en ce début de paragraphe (qui part d'un a priori qui se trouve ainsi validé : l'existence d'une telle structure induisant une sémantique interprétable en terme épistémiques, didactiques ou cognitifs). Enfin, il est à noter que le paquet comportant 7 tests passés correctement sur 7 correspond effectivement aux productions correctes. Malgré les avertissements de E.W. Dijkstra, les tests par jeux d'essai ne sont pas si mauvais.

Il est clair que l'analyse des productions d'étudiants à l'aide de jeux d'essai permettrait une correction plus aisée et plus sûre, peut-être même moins fastidieuse, entre autres en attirant l'attention du correcteur directement vers des paquets de copies relativement homogènes. Ces deux outils de corrections ne sont pas nécessairement si éloignés l'un de l'autre. La rédaction de jeux d'essai se rapproche de la mise en place de grilles d'évaluation descriptives systématiques et critériées avec la même contrainte que pour la mise en place de ces dernières : il se peut que cela soit réservé aux experts et d'une mise au point coûteuse. Mais, à la différence de l'emploi des grilles d'évaluation où une correction effective est nécessaire et reste coûteuse, avec des jeux d'essai, la phase de passation des tests peut être automatisée. Cependant, rappelons que la correction c'est aussi un temps où l'enseignant redécouvre sa matière via le filtre de ses étudiants. Outre les découvertes de détails sur ce qui passe bien ou pas, c'est le moment où peuvent se régler des choix de haut niveau vis-à-vis des éléments enseignés.

2.2. Analyse par jeux d'essai dans un exerciseur d'algorithmique (EDBA)

Changeons de contexte et considérons maintenant l'EIAH dont la mise en place a servi de contexte à cette réflexion sur la correction de copies : l'application web EDBA, Exercises DataBase about Algorithms (www.noe-kaleidoscope.org/public/people/DenisB/EDBA). L'objectif d'EDBA est de fournir une application web sous la forme d'un éditeur de code intégrant les interpréteurs nécessaires à l'exécution du code, un moteur de test accédant à une base de données d'exercices et de jeux d'essai permettant de s'exercer sur des problèmes algorithmiques de tous niveaux, quel que soit le langage de programmation ou la langue maternelle, avec des rétroactions sur les aspects syntaxiques et la correction des productions (à base d'évaluation par jeux d'essai). À ce jour, plus de 200 exercices sont

disponibles dans EDBA avec près de 1500 jeux d'essai, soit environ 7 tests par exercice. Plus de détails sur la mise en œuvre dans [Bouhineau - 11].

Depuis 2009, quelques étudiants ont pu travailler avec les premières versions d'EDBA : l'ensemble de la promotion de L3-MIAGE-IMA de 2009-2010 et de 2010-2011 (en cours), soit environ 45 étudiants chaque année, ont eu un libre accès à EDBA. Seuls une vingtaine a saisi cette opportunité (+10 simples curieux n'ayant effectué qu'une visite rapide inférieure à 1/2h) avec des durées de travail entre 1h30 et 15h, pour un total de plus de 100h. L'observation des logs de ces étudiants montre des enchainements de phases de rédaction de programmes se concluant par des phases de test-rédaction. Après la phase de rédaction, le premier test est rarement concluant (moins d'un cas sur 10 avec les premiers tests justes à 100%, ces cas rares correspondent à des phases de tests se terminant souvent dès le premier test), il s'ensuit des recherches d'amélioration visibles par d'autres test, en général améliorant le taux de réussite par à-coups (ex : 1 test ok sur 5, puis 4/5, puis encore 4/5, puis 5/5 et arrêt des tests), jusqu'à un plateau proche ou égal à 100% de tests réussis. L'étude de ces parcours confirme le fait que tous les cas de figure de test réussis ne sont pas observés. Il y a concentration sur un petit nombre de sous-ensembles de tests réussis. La progression dans les tests, le fait que le travail de rédaction des étudiants ne s'arrête pas avec le premier test, mais se poursuive jusqu'à un taux de réussite de 100% ou proche de 100%, semblent montrer que les étudiants trouvent là une motivation pour prolonger leur travail. Un système à points (points d'*expertise* distribués en fonction du nombre de tests réussis lors d'une passation, dépendant du niveau de difficulté de l'exercice, et capitalisés par l'étudiant pour pouvoir accéder à des exercices de plus grande difficulté) renforce probablement cette motivation. Lors d'entretiens informels, ces tests et ce système à points sont ressortis comme des éléments positifs de l'application. L'organisation du travail semble également modifiée : les étudiants ne découpent pas leur travail en deux phases, une phase de rédaction de tous les programmes, puis une phase de test et de correction globale, comme on peut l'observer couramment dans d'autres contextes d'enseignement de l'informatique (TP sur machine par exemple). Les étudiants ne passent à la rédaction de l'exercice ou du programme suivant qu'après une phase concluante de test de l'exercice courant.

Coté enseignant, ou rédacteur d'exercice/jeux d'essai, l'un des objectifs d'EDBA est de permettre d'introduire un exercice et les jeux d'essai associés en moins de 10 minutes (pour un exercice facile) et de pouvoir améliorer l'ensemble si nécessaire en moins de temps. Pour arriver à ce temps réduit, un minimum d'informations est demandé par EDBA au rédacteur pour introduire un exercice : un énoncé informel de l'exercice, une spécification formelle et des jeux d'essai. Aucune solution de l'exercice n'est demandée. Pour mettre au point l'exercice, le rédacteur peut programmer une solution dans EDBA et celle-ci peut alors servir à la rédaction des jeux d'essai. Pour améliorer l'exercice et les jeux d'essai, le rédacteur d'un exercice a accès aux codes des étudiants ayant essayé de résoudre l'exercice ainsi qu'aux exécutions des jeux d'essai. À travers l'analyse de ces premières productions, le rédacteur peut observer les premières tendances et solutions ou erreurs apparaissant et adapter l'énoncé ou les jeux d'essai en fonction. La section précédente d'analyse par jeux d'essai vs/en appui d'une analyse à la main a montré qu'il y avait là une nécessité d'avoir des outils pour faciliter ce travail. Une vision globale du cycle de vie de l'exercice semble donc encourager cet aller-retour entre rédacteurs et étudiants. On retrouve là une problématique connue au démarrage des EIAH [McLaren & Al. – 04].

Coté concepteur de l'EIAH, deux versants ont été considérés : d'un coté, comment mettre en place une base de jeux d'essai ; de l'autre, comment faire passer ces tests. Les sections précédentes ont montré que la mise en place d'une base de jeux d'essai tire un grand profit d'une intervention experte des enseignants apportant les exercices, non seulement à l'introduction initiale de l'exercice, mais aussi après, au cours de la vie de l'exercice, à la suite de premières tentatives de résolution de l'exercice. L'introduction de tests génériques a été envisagée pour simplifier l'introduction initiale d'un exercice, mais abandonnée car cela signifie remplacer la rédaction des tests par une rédaction d'une solution. Il n'est pas sûr que le rédacteur y gagne au change. Au final, l'effort du concepteur s'est orienté vers la mise en place d'une interface facilitant l'introduction de jeux d'essai contraints par un objectif global : pouvoir rédiger un exercice et ses jeux d'essai en moins de 10 minutes. Cet objectif étant atteint, la voie des tests génériques a été laissée de coté, ce qui nous a épargné la vérification de l'hypothèse de l'existence de données génériques. Pour la passation des tests, comme il a été déjà écrit ici et ailleurs, divers aménagements sont nécessaires pour assurer une bonne passation des tests (souplesse des comparaisons entre résultats de référence et résultats effectifs) et protéger l'EIAH (limitation des ressources et du temps alloué pour éviter les boucles infinies introduites par erreurs par les étudiants). Enfin, une exigence supplémentaire était imposée au concepteur dans EDBA : que la notion d'exercice et que les jeux d'essai soient indépendants d'un langage de programmation, de telle sorte que EDBA soit réellement un EIAH de l'algorithmique et pas seulement un EIAH de Prolog. Pour vérifier la possibilité de cette indépendance vis-à-vis des langages de programmation, un second langage est disponible dans EDBA (Caml, d'autres sont susceptibles d'être ajoutés dans la mesure où ils possèdent un interpréteur en Javascript, ce qui est le cas de Basic, Forth, Javascript, Lisp, Logo, Php, Ruby, Scheme, Smalltalk, X86, … pour ne citer que les langages les plus connus pour lesquels je connaisse un interpréteur Javascript) et deux types des traitements sont effectués pour utiliser les jeux d'essai disponibles (en Prolog) : d'une part des transformations syntaxiques simples (par exemple pour faire coïncider la syntaxe des listes Prolog avec celle des listes Caml) et d'autre part l'utilisation de filtres pour éliminer la passation de tests qui n'ont pas de sens en CAML (CAML étant un langage fonctionnel, sont passés uniquement les jeux d'essai comportant comme « résultat » une et une seule variable, ex : jeux d'essai 3.3, …, 3.8). A ce stade du développement d'EDBA, il s'agit cependant là plus d'une preuve de concept que d'une mise en place éprouvée.

Conclusion, perspectives

L'EIAH pour l'algorithmique EDBA étant dans une phase de mise en place, il fallait pouvoir déterminer si une évaluation de production d'étudiant par jeux d'essai était possible. L'étude qualitative qui est reproduite ici semble montrer que correction « à la main » et correction « automatique par jeux d'essai » peuvent atteindre les objectifs de validation des productions d'étudiant que poursuivent les enseignants. Pour les aspects syntaxiques, cette analyse repose sur les interpréteurs des langages de programmation intégrés à EDBA. Pour les aspects plus sémantiques, cette validation n'est pas immédiate, les premiers jeux d'essai peuvent reposer sur les exemples et contre-exemples qu'une correction « à la main » peuvent mobiliser a priori. Pour une analyse plus fine, le choix des jeux d'essai doit être l'objet d'un travail supplémentaire prenant en compte les erreurs effectivement observées

après l'obtention des première productions d'apprenant afin de mieux faire coïncider erreurs réelles et jeux d'essai. D'autres analyses, à base d'observations syntaxiques (par ex. : présence et décompte des cas de base et de propagation) peuvent être mises en œuvre (d'autres modules d'EDBA le font), mais dépendent des langages de programmation utilisés, à la différence de l'étude des réponses aux jeux d'essai qui peut se faire indépendamment des langages utilisés.

Au delà de l'analyse des productions par jeux d'essai qui fournit à l'étudiant des exemples (contre-exemples) où la solution proposée est mise en échec, des diagnostics plus signifiants peuvent être visés, voire des propositions de correction des erreurs, ou des exercices de remédiation en utilisant le même principe d'identification d'une production avec une production erronée de référence par l'observation de coïncidence des comportements sur un ensemble de jeux d'essai (partageant les mêmes « données » que l'exercice de référence, mais dont les « résultats » seraient spécifiques de chaque erreur de référence). Cependant, l'étude préliminaire donnée en 1 montre une grande dispersion des erreurs, une analyse a priori risque de n'aborder qu'une petite partie de l'espace des possibles. Une piste serait peut-être de faire des regroupements sur les tests erronés d'étudiants pour repérer les cas les plus courants. Ainsi, l'enseignant n'aura pas à écrire les programmes erronés de référence (ils seraient donnés par les étudiants), par contre, l'enseignant aurait à déterminer les ensembles de cas intéressants pour les prendre en compte. Mais, même dans cette démarche, le coût humain risque d'être très élevé pour atteindre une bonne représentativité des erreurs (sans parler de complétude). Aussi, l'appel à une démarche participative, associant le rédacteur initial et d'autres experts ou étudiants plus avancés dans leur apprentissage est envisagée.

Références (pages web accédées en novembre 2010)

[Bouhineau – 97] Denis Bouhineau. *Construction automatique de figures géométriques et programmation logique avec contraintes*, Thèse Univ. Joseph-Fourier - Grenoble I. 1997.

[Bouhineau & al. - 03] Denis Bouhineau, Alain Bronner, Hamid Chaachoua et Thomas Huguet. Analyse didactique de protocoles obtenus dans un EIAH en algèbre. Actes d'EIAH 2003.

[Bouhineau - 11] Denis Bouhineau, Ajax pour EIAH ? Actes d'EIAH 2011.

[Dijkstra – 69] Edsger Wybe Dijkstra. Notes on structured programming. Technological University Eindhoven, Netherlands Department of Mathematics. 1969.

[Douce & Al. – 05] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. J^{al} on Ed^{al} Resources in Computing, (5)3. 2005.

[Gérard 09] François-Marie Gérard. Evaluer des compétences. Ed. de Boeck Univ., Bruxelles. 2009.

[McLaren & Al. – 04] McLaren, B.M., Koedinger, K.R., Schneider, M., Harrer, A., & Bollen, L. Bootstrapping Novice Data: Semi-Automated Tutor Authoring Using Student Log Files. Proc. of ITS-2004, wp on Analyzing student-tutor interaction logs to improve ed^{al} outcomes. 2004.

[Pears & Al. – 07] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. A survey of literature on the teaching of introductory programming. SIGCSE (39)4. 2007.

[Scallon - 04] Gérard Scallon. L'évaluation des apprentissages dans une approche par compétences. Edition de Boeck Université, Bruxelles. 2004.

[Skupiene - 10] Jurate Skupiene, Improving the evaluation model for the lithuanian informatics olympiads. Informatics un education. Vol 9.1. Vilnius. 2010.

[Veslin & Veslin - 92] Odile et Jean Veslin. Corriger des copies, évaluer pour former. Pédagogie pour demain, Nouvelles approches. Hachette éducation, Paris. 1992.