De la compréhension de programmes en génie logiciel à la reconnaissance d'algorithmes d'apprenants en EIAH

Ismaïl Bouacha[1], Denis Bouhineau[2], Tahar Bensebaa[3]
[1] Ecole préparatoire aux sciences et techniques, Annaba, Algérie
ismail.bouacha@gmail.com
[2] Université de Grenoble, LIG, France
denis.bouhineau@imag.fr
[3] Université Badji Mokhtar-Annaba, LRI, Algérie
t_bensebaa@yahoo.com

Résumé. Dans le but d'évaluer les productions d'apprenants en algorithmique, nous présentons une méthode de reconnaissance des algorithmes proposés par les apprenants basée sur les techniques de compréhension des programmes du génie logiciel. Notre méthode repose sur une décomposition des algorithmes en tâche et sous-tâche à accomplir ; à associer à chaque sous-tâche des « lignes critiques » et des propriétés spécifiques. Cette décomposition/modélisation permet l'utilisation des techniques de compréhension de programmes de Génie Logiciel et mène à une évaluation/notation des propositions d'algorithmes basée sur le modèle identifié d'une base de modèles étiquetés et sur la distance entre le modèle et la proposition. Une première expérimentation à partir de copies d'examen donne des taux de reconnaissance intéressants et des notes pour les copies reconnues proches des notations manuelles.

Mots-clés. Évaluation, Apprenant, Notation, Algorithmique, Compréhension de programmes, Génie logiciel, Reconnaissance d'algorithmes, Modélisation.

Abstract. In order to assess learners in algorithmics, we suggest a method based on the automatic understanding of the algorithms proposed by the students using comprehension methods from the domain of the software engineering. We use prebuilt models of student's propositions, each one representing algorithms, organized into tasks and subtasks and documented with information and pedagogical characteristics. The mark is based on a distance calculus between the model and the proposition. A first experiment occurs with exam forms. It gives interesting recognition rates and marks for recognized forms close to the marks obtained during the exam.
Keywords. Assessment, Algorithmic, Learner, Program comprehension

1 Introduction

Cette communication aborde un thème central pour l'enseignement : l'évaluation des apprenants. L'avènement des LMS, le renouveau des MOOC ont montré ces dernières années combien il est difficile d'automatiser l'évaluation des apprenants et que les diverses propositions d'évaluations génériques et automatiques « passant à l'échelle » (QCM, ou évaluations par les pairs) ne permettent pas de viser les mêmes objectifs et niveaux d'évaluation et de diagnostic [1, 2].

En informatique et plus particulièrement pour ce qui concerne les apprentissages en algorithmique, il n'y a pas de solution miracle, les difficultés liées à la disciplines sont nombreuses : diversité des éléments à enseigner/évaluer et des niveaux d'abstractions concernés, pluralité des solutions possibles attendues pour n'en citer que quelques unes. Des méthodes d'évaluations ont pu prendre en compte divers aspects ponctuellement : prise en compte du « style d'écriture » avec les méthodes à base de métriques statiques [3], évaluation brutale de la sémantique des propositions à base de jeux de tests [4], évaluation cognitive des erreurs commises [5].

Notre objectif, concrétisé par une première réalisation ainsi qu'une première expérimentation, consiste à nous appuyer sur une expertise humaine pour gagner en généricité et en richesse et complétude d'analyse. Pour atteindre cet objectif, nous avons déplacé le problème de l'évaluation automatique vers celui de la reconnaissance automatique de modèles de copies attendues, ces modèles de copies étant fournies par un expert ou un enseignant. De cette reconnaissance de modèle, nous pensons atteindre une évaluation de meilleure qualité en profitant de ce que ces modèles de copies attendues ont pu être caractérisées « à la main » et nous pensons également libérer l'évaluateur du travail d'attribution des notes en construisant des notes à partir d'un calcul de distance entre les propositions à évaluer et le modèle qui aura été noté préalablement par l'expert.

2 Architecture et processus général

L'architecture globale du système comporte au centre l'outil de compréhension des propositions (en fonction des modèles de propositions, du problème envisagé et de la proposition d'étudiant elle-même). L'espace de proposition des algorithmes a une double fonction : « édition d'algorithmes » et « modélisation de modèles ».

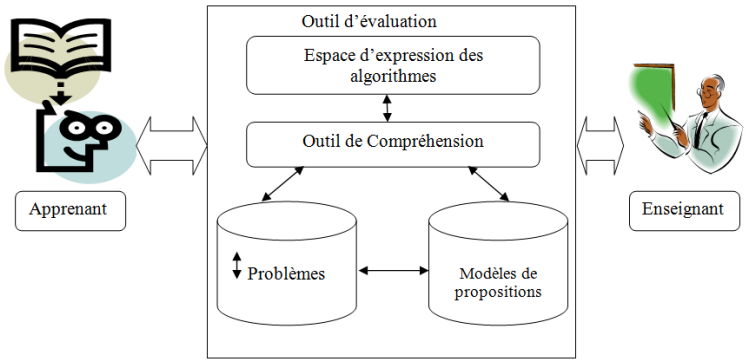

Fig. 1 Architecture globale de l'outil d'évaluation

L'apprenant accède à ce système à travers l'interface d'un éditeur spécifique AlgoEditor appartenant à l'espace d'expression des algorithmes qui structure la production de l'apprenant en lui autorisant seulement l'écriture des structures de contrôles habituelles (déclaration, affectation, conditionnelle, etc.) selon des schémas prédéfinis à compléter. L'emploi de cet éditeur soulage l'apprenant d'une partie des

efforts de rédactions les plus élémentaires. Ainsi, l'apprenant peut se concentrer sur les aspects de plus haut niveau de la rédaction des algorithmes et le système garantit que les propositions sont correctes syntaxiquement.

Cet éditeur est également utilisé par l'enseignant pour modéliser les modèles de propositions. Initialement, l'enseignant peut fournir quelques modèles de propositions a-priori. Lors des tentatives de reconnaissance de proposition, en cas d'échec de la reconnaissance, l'enseignant est interpellé pour évaluer la proposition et lorsque la proposition est intéressante de son point de vue, l'enseignant la transforme en modèle de propositions pour enrichir l'ensemble des modèles de propositions.

3 Modèles de propositions et Algorithmes de reconnaissance

Un modèle de proposition est produit à partir d'une proposition a-priori construite par l'enseignant ou d'une proposition concrète d'apprenant. Elle contient le code d'un algorithme précis et décrit ce qui est caractéristique dans la proposition et les variantes que cette proposition pourrait avoir. En plus, un modèle peut contenir des informations supplémentaires de plus haut niveau (intentions supposées du programmeur, erreurs observées) et enfin une note globale. Le modèle s'appuie sur une décomposition de la proposition en termes de tâches et sous-tâches, auxquelles sont associés des descripteurs. Cette décomposition et ces descripteurs sont inspirés par les méthodes de compréhension de programme de génie logiciel [6]. Un modèle de propositions est donc composé d'une proposition et des informations suivantes :

1. *Note :* Note globale du modèle.
2. *Tâches (et sous-tâches) :* Ensemble des tâches et sous-tâches qui composent la proposition. Chaque tâche possède un nom, une note locale, des indications de « Lignes critiques » facilitant la reconnaissance, les instructions qu'elle contient et des descripteurs (pour avoir plus de détails) :
    a. autoriser l'absence de certaines instructions voire de certaines tâches (ou la pénaliser),
    b. autoriser un ordre quelconque entre certaines instructions voire entre certaines tâches (ou pénaliser le désordre),
    c. alternatives possibles d'une expression.
3. *Commentaires libres*

La figure 2 ci-dessous montre un modèle de propositions composé de quatre tâches : Lecture (ligne 7 à 10), Initialisation (ligne 11 à 12), Test (ligne 13 à 20), et Résultat (ligne 21 à 26, omis sur la figure). En supposant que la note de cet exercice est sur 6 points, nous pourrions avoir la description suivante pour la tâche de lecture :

- La tâche de lecture est notée 1 sur 6.
- Les instructions se trouvent entre la ligne 7 (début) et la ligne 10 (fin).
- Cette tâche contient une indication de ligne critique (lc) pour la ligne 9 (Une ligne critique est une instruction dont l'apparition dans le code de la proposition indique la présence probable d'une tâche d'un modèle donné.)
- Cette tâche est nécessaire, une pénalité de 1 point est attribuée en cas d'absence.

- Cette tâche doit se produire avant les tâches de Test et de Résultat, une pénalité de 0.5 point est attribuée si cette tâche s'effectue après.
- L'ordre entre cette tâche de lecture et la tâche d'initialisation est indifférent.

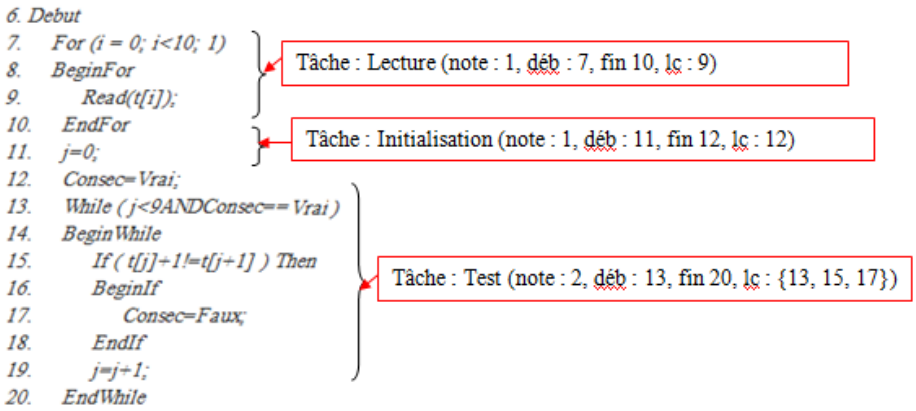

Fig. 2 : Modèle de proposition (décomposition de la proposition en tâche)

La décomposition en tâche et sous-tâche et l'ajout des descripteurs permettent d'étendre la classe des propositions reconnues pour un même modèle. L'objectif est d'atteindre ainsi une plus grande flexibilité pour un modèle donné et obtenir des reconnaissances partielles.

L'algorithme de reconnaissance de modèle est inspiré des travaux en Génie logiciel pour la compréhension de programme et la retro-ingénérie [7]. Elle consiste à s'abstraire du nom des variables, retrouver des motifs prédéfinis d'algorithmes (tâche ou sous-tâche, lignes critiques) et essayer d'assembler ces motifs pour recomposer la proposition. Pour mettre en œuvre le processus de reconnaissance de modèle, nous proposons l'algorithme suivant (simplifié) :

1. Charger les modèles et la proposition,
2. **Pour** chaque modèle :
3. Cloner les variables de la proposition dans le modèle,
4. Chercher les lignes critiques du modèle,
5. **Répéter :**
6. Candidat ← récupérerMeilleureCandidat (modèles),
7. EssayerModèle(candidat)
8. **Tant que** candidat disponible & proposition non reconnue.

Le chargement des variables permet de faire une correspondance entre chacune des variables du modèles et les variables équivalentes dans la proposition, d'où le clone.

Nous cherchons pour chaque modèle les lignes critiques présentes dans la proposition de l'apprenant, après avoir chargé les variables du modèle. Cette recherche permet de classer les modèles avec en premier ceux qui ont le plus de chance d'être compatibles avec la proposition. Pour choisir le meilleur candidat parmi les modèles de propositions, nous estimons le pourcentage de tâches localisées avec le

pourcentage de lignes critiques localisées dans la proposition de l'apprenant.

Après avoir obtenu un modèle candidat, l'étape « EssayerModèle(candidat) » consiste à vérifier la correspondance entre ce candidat et la proposition de l'apprenant, en appliquant l'algorithme suivant (simplifié) :

1. **Pour** chaque Tâche du modèle :
2. **Pour** chaque instruction de la proposition similaire au début de la tâche :
3. **Pour** chaque instruction à la suite de la tâche courante du modèle :
4. Trouver une instruction similaire dans la suite de la proposition
5. DeciderSiPropositionEstReconnue

L'idée consiste à essayer de trouver les tâches du modèle en commençant par leur début. Chaque début possible est identifié dans la proposition. À la suite de ce début possible le reste de la tâche est recherché.

En fin, une dernière procédure détermine si la proposition correspond au modèle et quelle note (ou distance) peut lui être attribuée. Cette procédure opère en fonction des correspondances trouvées et des descripteurs de tâches. Ainsi, seules les tâches ayant un descripteur dans le modèle qui leur permet d'être absentes peuvent manquer. Idem pour les descripteurs concernant l'ordre des tâches. La note donnée à la proposition sera maximale si la correspondance est complète et ne nécessite pas la souplesse autorisée par les descripteurs.

5 Premiers résultats expérimentaux

Pour tester notre approche, nous avons travaillé avec des copies d'examen de seconde année d'informatique de l'école préparatoire aux sciences et techniques Annaba, en Algérie, sur l'exercice suivant : écrire un programme qui vérifie si les éléments d'un tableau (d'entiers) sont consécutifs ou non (exemple : les éléments 4, 5, 6, 7, 8 sont consécutifs tandis que les éléments : 1, 3, 4, 5, 6 ne le sont pas).

Après la correction manuelle de 22 copies, nous avons pu extraire 7 modèles de proposition possibles (correctes et incorrectes). Avec ces 7 modèles, nous avons analysé automatiquement 72 nouvelles copies (issues de 4 groupes de TD différents). Le taux moyen de reconnaissance obtenu a été de 56% avec un taux de reconnaissance similaire pour les différents groupes de copies et similaire pour les copies correctes et les copies incorrectes.

À partir des modèles et copies des groupes 1, 2 et 3 nous avons croisé les résultats de reconnaissance avec une analyse « à la main » effectuée par 3 enseignants. Les accords entres les différentes analyses étaient largement majoritaires (35 accords complets, i.e. 66% des cas où tous les juges avaient le même avis sur le choix d'un modèle ou l'absence de reconnaissance).

Pour les 31 copies reconnues des groupes 1, 2 et 3, une analyse des notes attribuées a été effectuée. Globalement, un tiers des copies (11) a reçu la même note (note donnée sur 6) ; pour la moitié des copies (15), la note reçue présentait une différence de 1 point avec la note obtenue le jour de l'examen ; dans 5 cas (16%), la note différait de 2 ou 3 points (sur 6 points).

6 Conclusion, perspectives

Nous avons présenté une méthode de reconnaissance des algorithmes d'apprenants. Notre méthode tire profit du contexte d'application pour mettre en place et utiliser une base de modèles de propositions d'algorithmes. Les modèles utilisés sont enrichies d'informations permettant l'utilisation des techniques efficaces de compréhension en génie logiciel et la notation des propositions d'algorithmes à partir du modèle identifié et de la distance entre ce modèle et la proposition. Une première expérimentation à partir de copies d'examen a donnée des taux de reconnaissance intéressants (plus de 50%), similaire aux taux de reconnaissance « à la main » et des notes pour les copies reconnues proches des notations manuelles. Pour améliorer la notation, une piste prometteuse consisterait à associer l'algorithme de reconnaissance avec un algorithme d'évaluation dynamique de correction des propositions à base de jeux d'essais [8]. Au delà de la notation, nous pensons aussi pouvoir associer aux modèles des informations sur les connaissances et compétences relatives à chaque tâches et sous-tâches et enrichir l'évaluation.

Références

1. Simkin, M. & Kuechler, W.: Multiple-Choice Tests and Student Understanding: What Is the Connection? Decision Sciences J. of Innovative Ed., Vol 3 (2005) 73 – 98

2. Sitthiworachart, J., & Joy, M.: Effective Peer Assessment for Learning Computer Programming. 8th conf. on Innovation and technology in CSE, UK, (2003) 122 – 126

3. Mengel, S.A., & Ulans, J.V.: A case study of the analysis of the quality of novice students programs. 12th conf. on Soft. Eng. Education and Training, (1999) 40 – 49

4. Chen, P.: An Automated Feedback System for Computer Organization Projects. *IEEE Transactions on Education ,* Vol 47 *,* (2004) 232 – 24

5. Michaelson, G.: Automatic Analysis of Functional Program Style. In proceedings of Australian Software Engineering Conference, (1996) 38 – 46

6. Corbi, T.A.: Program understanding:Challengeforthe1990s. IBM Systems Journal, Vol 28, issue 2, (1989) 294 – 306

7. Waters, R.C., Chikofsky, E.: Reverse engineering : progress along many dimensions. Communications of the ACM, Vol 37, issue 5, (1994) 22 – 25

8. Bouhineau, D.: Utilisation de traits sémantiques pour une méthodologie de construction d'un système d'aide dans un EIAH de l'algorithmique. EIAH'2013, Toulouse (2013) 141 – 152