Appel à Défis pour le Génie de la Programmation et du Logiciel

Sur quelques opportunités de l'Intelligence Artificielle pour le Génie Logiciel (et réciproquement)

H.-L. Bouziane, D. Delahaye, C. Dony, M. Huchard,
C. Nebut, A.-D. Seriai et C. Tibermacine

Équipe MaREL, LIRMM, Université de Montpellier, CNRS, Montpellier

1 Introduction

Les différentes branches de l'intelligence artificielle offrent de nombreuses opportunités de développement dans de très nombreux secteurs de l'activité humaine. La science du logiciel en tire depuis longtemps de nombreux bénéfices. Dans la communauté francophone plus spécifiquement, il est intéressant de se rappeler que la conférence LMO (Langages et Modèles à Objets) a regroupé dès son origine des recherches en représentation des connaissances et en programmation et a été le creuset de travaux communs aux chercheurs de ces deux domaines [5] [1]. L'IA peut plus largement apporter des solutions pour la réduction des coûts et des temps de développement et de maintenance, ainsi que pour l'amélioration de la qualité, dont la vérification. Dans cet article de positionnement, nous attirons l'attention sur la problématique générale (section 2), puis plus particulièrement sur le rôle que pourrait jouer l'IA dans les domaines de la sûreté de fonctionnement (section 3) et de l'exploitation des bases de code (section 4), et inversement quelques éléments sur le support que notre domaine peut apporter au domaine de l'intelligence artificielle (sections 3 et 5).

2 IA pour le GL : problématique générale

La science du logiciel et l'intelligence artificielle partagent plusieurs secteurs communs. Les systèmes multi-agents et les systèmes à composants partagent de nombreuses problématiques comme l'organisation en petites entités autonomes et communiquantes, la résolution distribuée de problèmes, l'auto-adaptation ou encore la fiabilité et la tolérance aux fautes, avec de la fertilisation croisée entre les domaines [1]. La définition de modèles et d'architectures de logiciels compréhensibles s'appuie sur des techniques de représentation des connaissances, telles que les graphes de connaissances ou les ontologies [6]. L'interopérabilité entre services Web ou l'automatisation des transformations de modèles ou de méta-modèles s'appuient sur des techniques d'alignement de schémas ou d'ontologies [14]. La génération automatique de compositions de services utilise des techniques de planification [3]. Les techniques de traitement de la langue naturelle sont couramment appliquées par exemple à l'ingénierie des exigences [4], à l'analyse des identificateurs [9], ou encore aux termes apparaissant dans la documentation ou les rapports de bogues [22]. La programmation par contraintes est utilisée dans la localisation de fautes [2] ou pour la génération de modèles conformes à un méta-modèle [10]. On

1. `https://hal.archives-ouvertes.fr/hal-01401179/file/carre1995a.pdf`.

peut noter de plus en plus de travaux utilisant de l'apprentissage statistique (comme des réseaux de neurones). Pour en donner quelques exemples, Tufano et al. montrent qu'il est possible, dans un contexte restreint d'apprendre des modifications de code pour le *refactoring* ou la correction de bogues à partir des codes sources et des *pull requests* [23] (ICSE 2019) ; Zhanh et al. [25] identifient des instructions suspectes dans du code (SANER 2019) ; enfin, Palmerino et al. [18] utilisent la régression linéaire multiple pour les systèmes auto-adaptatifs (ASE 2019).

L'apprentissage statistique prend actuellement une grande place grâce aux nouvelles capacités des ordinateurs. Cependant, les deux courants de l'intelligence artificielle ont chacun leur rôle à jouer. Les approches symboliques et logiques sont utiles pour leur aspect déductif et leur prise en compte des aspects conceptuels des domaines étudiés (ici les données particulières du génie logiciel). Les approches statistiques sont utiles pour leur efficacité et leur aspect prédictif. Les limites des approches symboliques tiennent dans le besoin d'expliciter les connaissances et les règles, et de mener des raisonnements parfois coûteux. Les approches statistiques ont leurs limites également comme le fait de prendre peu en compte la connaissance d'un domaine et la structure des informations. On leur reproche aussi leur côté « boîte noire » et la difficulté, voire l'impossibilité à expliquer les résultats. L'entraînement à partir d'exemples crée des biais dans l'apprentissage, qui sont potentiellement aussi graves pour les pratiques et constructions logicielles que pour les aspects sociaux [17]. Des travaux s'intéressent à certains des problèmes posés, comme le test des algorithmes à base d'apprentissage profond dans [13], les vulnérabilités [24], la prise en compte des structures comme les arbres de syntaxe abstraite et la tentative d'apporter une explication [12]. Frank van Hermelen, lors d'un exposé invité à la conférence EGC 2018 [2] développe par ailleurs une vision consistant à proposer des schémas de combinaison des deux courants (symbolique et statistique) plutôt que de les opposer.

L'un des défis à relever ici est de réaliser ces combinaisons dans le contexte de la science du logiciel. L'approche d'extraction d'architecture présentée dans [21] est un exemple d'une telle combinaison entre une approche de classification statistique (regroupement hiérarchique basé sur une métrique de similarité) et une approche symbolique (l'analyse formelle de concepts). Un autre défi consiste à intégrer aux approches statistiques la connaissance du domaine du logiciel et du métier du logiciel ciblé, des structures et des concepts qui nous sont propres, et à dépasser les problèmes d'explicabilité, de biais et de vulnérabilité.

3 Sûreté de fonctionnement et IA

Une IA plus sûre. Ces dernières années, l'apprentissage automatique statistique a concentré l'attention du monde de la recherche académique mais aussi de l'industrie dans les secteurs critiques comme l'automobile, l'industrie, la finance ou encore la santé. Si l'on sait aujourd'hui assurer la correction de programmes que l'être humain écrit en se basant notamment sur des algorithmes dont le comportement est clairement spécifié, qu'en est-il de la correction de programmes qui ne sont plus écrits mais appris ? Dans l'automobile par exemple, l'apprentissage automatique combiné à la puissance de calcul informatique qui ne cesse de croître permet d'identifier en temps réel les éventuels obstacles qui se présentent sur une route, une compétence indispensable pour développer des véhicules autonomes. Mais ce type de détection n'est pas infaillible et des accidents dramatiques sont déjà à déplorer. Cependant, vérifier les algorithmes d'apprentissage automatique est complexe. Il semble difficile de spécifier formellement un algorithme d'apprentissage automatique, dont on ne

2. « Combining Learning and Reasoning : New Challenges for Knowledge Graphs », exposé disponible à l'adresse suivante : `http://www.canalc2.tv/video/15255`.

sait pas toujours expliquer le comportement et dont les résultats ne sont pas toujours reproductibles (deux apprentissages sur le même jeu de données peuvent donner deux systèmes différents). De ce fait, on peut imaginer abandonner l'approche certifiée pour une approche certifiante, qui consiste à vérifier le résultat au coup par coup. Mais cette approche est bien moins ambitieuse et surtout ad hoc. Il est, en outre, indispensable de demander à la méthode d'apprentissage automatique de fournir un certificat, qui permettra de vérifier le résultat et qui est nécessaire lorsqu'il est impossible de vérifier le résultat seul. Ce certificat peut être vu comme une explication et c'est justement ce que les algorithmes d'apprentissage sont incapables de bien produire aujourd'hui.

L'IA pour la sûreté de fonctionnement. Inversement, on peut se demander si l'apprentissage automatique peut apporter de l'aide au domaine de la sûreté de fonctionnement. Aujourd'hui, il semble que l'apprentissage automatique ne produit pas forcément de très bons résultats lorsqu'il est appliqué dans des contextes très formels. Par exemple, dans le domaine de la preuve interactive (permettant de démontrer formellement la correction de programmes, par exemple), les expérimentations récentes dans des systèmes comme Isabelle [15] ou Coq [7] montrent des difficultés de l'apprentissage automatique à guider l'utilisateur dans la preuve, c'est-à-dire lorsqu'il faut recommander à l'utilisateur une tactique de preuve. Les difficultés sont encore plus grandes lorsque la tactique de preuve est paramétrée car actuellement, il est impossible de reconstruire ces paramètres par apprentissage automatique. On est donc bien loin d'une situation satisfaisante. De même, dans le domaine de la preuve automatique, l'apprentissage automatique ne permet pas de déduire la preuve automatiquement. En revanche, si on est moins ambitieux, on peut utiliser l'apprentissage automatique pour aider la recherche de preuve. Par exemple, l'apprentissage automatique pourrait permettre de sélectionner les axiomes d'une théorie nécessaires pour démontrer une certaine formule. Cette sélection est importante car elle permet de réduire l'espace de recherche de preuve et donc d'améliorer significativement la recherche de preuve dans certains cas. Des expérimentations récentes [8] montrent que cette technique donne de bons résultats dans des théories réputées difficiles en preuve automatique, comme la géométrie par exemple. Ainsi, aujourd'hui, il semble raisonnable d'utiliser l'apprentissage automatique comme aide aux méthodes formelles, mais certaines difficultés subsistent si l'on souhaite le faire apparaître comme un acteur principal du processus de développement. Ces difficultés sont liées à un certain nombre d'éléments que l'apprentissage automatique ne prend pas en compte. Par exemple, si les données d'entrée sont des formules de logique du premier ordre, on aimerait bien que la structure arborescente, ainsi que les liaisons des variables, soient analysées et exploitées, plutôt que de voir ces formules comme de simples chaînes de caractères.

4 Ré-ingénierie du logiciel et IA

Depuis quelques années, on voit l'émergence de grandes bases de code ouvert, comme GitHub.com, GitLab.com, Maven Central ou Software Heritage. Ces bases de code ne cessent de grandir en taille. Elles fournissent de précieuses informations sur l'évolution de systèmes à code source ouvert, leurs différentes versions, voire même, dans certaines bases, les descriptions des modifications apportées à ces systèmes entre les versions (les messages des *commits*, les journaux de modifications ou *changelogs*, ...). Toutes ces informations peuvent être utilisées dans l'apprentissage automatique pour entraîner des classificateurs permettant par exemple de : (1) reproduire l'évolution de programmes utilisant une certaine bibliothèque vers une autre bibliothèque (synthétiser des adaptateurs [16] ou générer des opération de *refactorings*) ; (2) migrer un projet d'une version à une

autre d'une bibliothèque, voire même d'un langage ou d'un paradigme à un autre, par exemple du paradigme objet vers le paradigme Workflow [20] ; (3) restructurer une grande application à objets vers un système de modules. Le défi à relever ici est celui de l'exploitation de ces grandes collections de données de projets à code source ouvert, qui incluent beaucoup de connaissances enfouies, pour extraire et apprendre les bonnes pratiques suivies par les développeurs afin d'assister l'activité de ré-ingénierie du code. Toute la difficulté dans ce défi réside dans la définition du modèle adéquat et de l'optimisation de ce dernier, avec les paramètres précis, pour réaliser l'apprentissage automatique mentionné ci-dessus. L'autre point difficile à traiter concerne le pré-traitement à réaliser sur ces collections de données pour, entre autres, éliminer tout le bruit qui peut exister dans ces données, comme les mauvaises pratiques de développement et d'évolution. Il est intéressant de noter que ce défi est directement connecté à d'anciens défis du génie logiciel. Par exemple, au début des années 1970, Chuck Rich, Dick Waters et Howard Shrobe, inspirés par un certain nombre d'autres personnes du laboratoire d'IA du MIT (par exemple, Terry Winograd, Carl Hewitt et Gerry Sussman), ont proposé l'idée d'un apprenti du programmeur [19], un assistant intelligent qui aiderait un programmeur à écrire, déboguer et faire évoluer des logiciels. La présence d'un large sous-ensemble de tous les logiciels du monde disponible en ligne dans des dépôts combinée aux avancées récentes de l'apprentissage automatique, permettant de faciliter l'acquisition de connaissances par fouille de données, devrait permettre de faire un grand pas pour faire de cette vision une réalité.

5 Conclusion

Dans cette courte synthèse et selon notre point de vue, nous avons cherché à montrer comment l'IA peut soutenir nos travaux en science du logiciel dans ses efforts pour proposer des méthodes, des pratiques et des outils pour maîtriser le développement et la maintenance. Plusieurs défis apparaissent comme (1) l'utilisation de l'apprentissage automatique pour la sûreté de fonctionnement des systèmes, (2) l'exploitation de toutes les données aujourd'hui disponibles sur les projets logiciels, (3) la combinaison des différentes approches d'IA, avec proposition de modèles de représentation de connaissances adéquats pour les données logicielles, les processus, les pratiques pour les utiliser de manière appropriée en association avec les algorithmes d'apprentissage, (4) la constitution d'une base pour l'interprétation/l'explication des résultats, (5) l'ajout aux approches les plus récentes (lignes de produits, IDM/MDE, DevOps, etc.) de capacités d'anticipation et de recommandation. Inversement, le génie logiciel peut apporter ses méthodes à l'IA, notamment sur la confiance [3] (voir section 3), le test (voir la conférence *Artificial Intelligence Testing* [4]) et la construction des *frameworks* [11].

Références

1. J.-P. Arcangeli, V. Noel, and F. Migeon. Software Architectures and Multi-Agent Systems. In M. Oussalah, editor, *Software Architectures*, volume 2, chapter 5, pages 171–208. Wiley, mai 2014.
2. N. Aribi, M. Maamar, N. Lazaar, Y. Lebbah, and S. Loudni. Multiple Fault Localization Using Constraint Programming and Pattern Mining. In *ICTAI 2017*, pages 860–867, 2017.
3. A. Bekkouche, S. M. Benslimane, M. Huchard, C. Tibermacine, H. Fethallah, and M. Mohammed. QoS-Aware Optimal and Automated Semantic Web Service Composition With User's Constraints. *Service Oriented Computing and Applications*, 11(2) :183–201, 2017.

3. `https://gl.frama.io/manifeste/`.
4. `http://ieeeaitests.com/`.

4. E. E. Bella, M. Gervais, R. Bendraou, L. Wouters, and A. Koudri. Semi-Supervised Approach for Recovering Traceability Links in Complex Systems. In *ICECCS 2018,*, pages 193–196, 2018.
5. B. Carré, R. Ducournau, J. Euzenat, A. Napoli, and F. Rechenmann. Classification et objets : programmation ou représentation ? In *5e journ. nat. PRC-GDR IA*, pages 213–237, Feb. 1995.
6. C. Coral, R. Francisco, and P. Mario. *Ontologies for Software Engineering and Software Technology.* Springer-Verlag, Berlin, Heidelberg, 2006.
7. D. Delahaye and A. Izard. Utilisation de techniques d'apprentissage automatique pour l'aide à la preuve dans le système Coq. Technical report, Université de Montpellier, 2019.
8. D. Delahaye and B. Lemoine. Déduction automatique en géométrie en utilisant l'apprentissage profond. Technical report, Université de Montpellier, 2019.
9. J. Falleri, M. Huchard, M. Lafourcade, C. Nebut, V. Prince, and M. Dao. Automatic Extraction of a WordNet-Like Identifier Network from Software. In *ICPC'10*, pages 4–13, 2010.
10. A. Ferdjoukh, A. Baert, E. Bourreau, A. Chateau, R. Coletta, and C. Nebut. Instantiation of Metamodels Constrained with OCL - A CSP Approach. In *MODELSWARD 2015*, pages 213–222, 2015.
11. Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and L. Xiaohong. An Empirical Study towards Characterizing Deep Learning Development and Deployment across Different Frameworks and Platforms. In *ASE 2019*, 2019.
12. L. Jiang, H. Liu, and H. Jiang. Machine Learning Based Automated Method Name Recommendation : How Far Are We. In *ASE 2019*, 2019.
13. J. Kim, R. Feldt, and S. Yoo. Guiding Deep Learning System Testing using Surprise Adequacy. In *ICSE 2019*, pages 1039–1049, 2019.
14. L. Lafi, J. Feki, and S. Hammoudi. Metamodel Matching Techniques : Review, Comparison and Evaluation. *IJISMD*, 5(2) :70–94, 2014.
15. Y. Nagashima and Y. He. PaMpeR : Proof Method Recommendation System for Isabelle/HOL. In *ASE 2018*, pages 362–372, 2018.
16. M. Nita and D. Notkin. Using Twinning to Adapt Programs to Alternative APIs. In *ICSE'10 - Volume 1*, ICSE '10, pages 205–214. ACM, 2010.
17. C. O'Neil. *Algorithmes : la bombe à retardement* . Les Arènes, Paris, 2018.
18. J. Palmerino, Q. Yu, T. Desell, and D. Krutz. Improving the Decision-Making Process of Self-Adaptive Systems by Accounting for Tactic Volatility. In *ICSE 2019*, 2019.
19. C. Rich and R. C. Waters. The Programmer's Apprentice : A Research Overview. *Computer*, 21(11) :10–25, Nov. 1988.
20. A. Selmadji, A. Seriai, H. Bouziane, and C. Dony. From Object-Oriented to Workflow : Refactoring of OO Applications into Workflows for an Efficient Resources Management in the Cloud. In *ENASE 2018, Revised Selected Papers*, pages 186–214, 2018.
21. A. Shatnawi, A.-D. Seriai, and H. Sahraoui. Recovering Software Product Line Architecture of a Family of Object-Oriented Product Variants. *Journal of Systems and Software*, 131 :325–346, Sept. 2017.
22. Y. Tian and D. Lo. A Comparative Study on the Effectiveness of Part-of-Speech Tagging Techniques on Bug Reports. In *SANER 2015*, pages 570–574, 2015.
23. M. Tufano, J. Pantiuchina, C. Watson, G. Bavota, and D. Poshyvanyk. On Learning Meaningful Code Changes via Neural Machine Translation. In *ICSE 2019*, pages 25–36, 2019.
24. J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang. Adversarial Sample Detection for Deep Neural Network through Model Mutation Testing. In *ICSE 2019*, pages 1245–1256, 2019.
25. Z. Zhang, Y. Lei, X. Mao, and P. Li. CNN-FL : An Effective Approach for Localizing Faults using Convolutional Neural Networks. In *SANER 2019*, pages 445–455, 2019.