Visualisation automatique de graphes pour l'apprentissage des algorithmes

Pierre Baron, Laurent Roche, Thibault Raffaillac

Visualisation automatique de graphes pour l'apprentissage des algorithmes

Pierre Baron[1], Laurent Roche[1] et Thibault Raffaillac[1,2]

[1] École Centrale de Lyon
pierre.baron@ecl19.ec-lyon.fr, laurent.roche@ecl19.ec-lyon.fr
[2] LIRIS, CNRS UMR5205, F-69621, France
thibault.raffaillac@ec-lyon.fr

Résumé. Lors de l'apprentissage des algorithmes en informatique, les visualisations sont généralement appréciées pour aider à raisonner sur des structures de données abstraites. Cependant les outils dédiés nécessitent souvent un investissement conséquent, qui freine leur adoption et réduit l'autonomie des étudiants. Cet article présente AutoGraph, un module pour Python permettant de visualiser toute implémentation d'un graphe, quel que soit son format, sans borner la pratique au sein d'un logiciel spécifique. Nous présentons ici un prototype fonctionnel, les usages auxquels il s'adresse et notre démarche de dissémination auprès de lycées et de formations post-bac.

Mots-clé: apprentissage de l'informatique · programmation · algorithmes de graphes · visualisation interactive · autonomie

Abstract. When learning algorithms in computer science, visualizations are generally appreciated to help reason about abstract data structures. However, dedicated tools often require a significant investment, which hinders their adoption and reduces the autonomy of students. This paper presents AutoGraph, a module for Python that allows visualizing any implementation of a graph, whatever its format, without restricting the practice within a specific software. We present a functional prototype, its intended uses and our approach to disseminate it to high schools and undergraduate courses.

Keywords: computer learning · programming · graph algorithms · interactive visualization · autonomy

1 Introduction

Depuis de nombreuses décennies, les algorithmes sont devenus des outils essentiels dans de nombreux domaines externes à l'informatique : génie civil, bioinformatique, économie, sociologie, etc. En particulier, les algorithmes de graphes sont très populaires et répandus aujourd'hui. Ils servent à traiter par exemple des cartes routières, des chaînes de protéines, des réseaux de change entre devises

ou encore des réseaux sociaux. Une conséquence de cette importance croissante est un enseignement de plus en plus précoce : autrefois réservés aux cursus universitaires, les graphes sont enseignés en terminale aujourd'hui [5].

Pourtant, la structure de données *graphe* et les algorithmes liés restent des notions difficiles à enseigner. En effet, contrairement à d'autres structures fondamentales (ex. tableaux, dictionnaires, piles), les langages de programmation comme Python n'ont pas de types natifs pour les graphes, ce qui rend leur implémentation particulièrement verbeuse. De plus leur représentation en code est souvent peu lisible (figure 1), ce qui impose un effort d'abstraction lors de l'implémentation d'algorithmes. Face à ces limites, nous cherchons à étudier **en quoi la visualisation de graphes lors de leur programmation peut améliorer l'apprentissage des algorithmes de graphes**.

```
[
  [0, 1, 0, 0, 1],
  [1, 0, 1, 0, 0],
  [0, 0, 0, 0, 1],
  [0, 1, 1, 0, 0],
  [0, 0, 0, 1, 0]
]
```

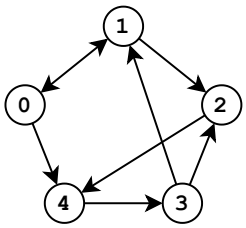

Fig. 1. Représentation d'un même graphe en code (matrice d'adjacence), et image.

De nombreux travaux de recherche ont proposé des plateformes complètes incluant éditeur de code, syntaxe de spécification de graphes et outil de visualisation intégré [6, 2, 1]. D'autres se sont basés sur des représentations textuelles interopérables (ex. DOT ou Newick) en facilitant les transformations vers et depuis ces formats [3]. Cependant en dépit de résultats positifs liés à l'engagement actif des étudiants dans l'apprentissage des algorithmes, ces outils ont été relativement peu utilisés en pratique [4].

Nous proposons ici une nouvelle alternative : la visualisation automatique de graphes, par détection d'implémentation lors de l'exécution du programme (voir https://github.com/sical/autograph). Ce travail en cours permet aux enseignants et étudiants d'utiliser toutes représentations plus ou moins complexes, sans être restreints à une syntaxe donnée. Nous présentons ici notre méthode de recherche, ainsi qu'une démonstration fonctionnelle en cours de validation.

2 Méthode de recherche

Notre méthode est itérative et centrée utilisateurs : nous interrogeons les enseignants et étudiants tout au long du projet, afin d'affiner notre compréhension des besoins et faire évoluer le prototype de recherche en conséquence. Ainsi une preuve de concept a été produite initialement, puis présentée à des enseignants

de 3 établissements d'enseignement supérieur locaux. Nous avons principalement relevé des environnements de développement en Python hétérogènes, ce qui a conditionné le choix d'une interface au format HTML, qui puisse être intégrée à un éditeur de code en ligne, un notebook Jupyter, ou un éditeur hors ligne avec le module `pywebview`. Le second prototype est celui présenté dans cet article. Il doit être testé directement par les enseignants, pour recueillir des besoins spécifiques à l'utilisation de l'outil. Une troisième version sera produite ultérieurement pour être utilisée en conditions écologiques par les étudiants.

Nous prévoyons de diffuser AutoGraph auprès de lycées et formations post-bac selon le protocole suivant :

- Pour chaque classe dans laquelle AutoGraph est introduit, nous organisons un workshop de 30min à 1h avec des enseignants, pour les former et recueillir des idées d'améliorations dans une démarche de co-conception.
- Durant toute la période de mise à disposition de l'outil, sous réserve d'accord des enseignants nous enregistrons les utilisations qui en sont faites, en vue d'analyser à quel point les étudiants se l'approprient.
- À l'issue de cette période, nous prévoyons des entretiens avec les étudiants et enseignants, pour recueillir en particulier (i) les problèmes observés avec l'utilisation d'AutoGraph, (ii) les fonctionnalités jugées utiles en pratique, et (iii) les différences perçues dans l'enseignement des graphes par rapport aux années précédentes.

3 AutoGraph, un module de visualisation de graphes

Ainsi qu'évoqué en introduction, la faible utilisation des environnements éducatifs de programmation de graphes nous a poussés à concevoir un outil facile à adopter. Celui-ci doit s'intégrer aux outils des utilisateurs avec le moins de contraintes possibles, et être utilisable avec le moins de connaissances préalables. Notre hypothèse est que la simplicité et la versatilité d'un tel outil favoriseront son utilisation sans supervision par les étudiants, et pourraient ainsi améliorer son impact sur l'apprentissage.

Nous avons donc conçu un module pour Python, qui puisse être utilisé aussi bien depuis un environnement Web (ex. Jupyter, tableau de bord) qu'hors ligne (ex. Spyder, PyCharm). Ce module contient une unique fonction `autograph`, qui prend en argument un objet quelconque (ex. matrice 2D, liste de paires d'entiers) et génère une visualisation au format HTML. Pour cela, l'objet est comparé à 8 implémentations reconnues de graphes. S'il correspond à l'une d'entre elles, une image est générée selon la représentation détectée. Deux observations importantes sont à l'origine d'AutoGraph :

- le "typage dynamique" de Python permet de passer à une même fonction `autograph` tous types d'implémentations sans la dupliquer en autant de variantes, ce qui est plus contraint avec les langages C/C++ ou Java.
- le nombre de possibilités *raisonnables* d'implémenter d'un graphe, c'est-à-dire qu'on rencontre dans les milieux professionnels, est limité (nous en avons énuméré pour l'instant 8).

La validation automatique de graphes permet aux étudiants de vérifier que leur compréhension d'une implémentation est correcte, et ensuite d'observer l'évolution de la structure à différents moments d'un algorithme. Il est également possible de créer un graphe par manipulation directe dans l'interface et de l'exporter en code, comme sur la plateforme https://visualgo.net/.

En revanche, AutoGraph n'est pas capable de détecter les implémentations erronées de graphes, car il y a trop d'erreurs possibles à prendre en compte. Il est alors important que l'équipe enseignante oriente les étudiants au préalable vers des implémentations correctes, ou qu'ils commencent par dessiner dans l'outil avant d'exporter le code correspondant. Ensuite l'outil n'est pas conçu a priori pour des usages avancés de visualisation de données ou de problèmes algorithmiques complexes. Pour ces cas de figure, AutoGraph peut convertir les graphes d'entrée pour le module externe `networkx` plus avancé.

4 Conclusion et perspectives futures

Cet article a présenté notre étude en cours sur l'utilisation de visualisations interactives de graphes, pour améliorer l'apprentissage des algorithmes. Nous avons introduit AutoGraph, un outil versatile conçu pour une utilisation non supervisée par les étudiants. À l'avenir, nous envisageons de faire évoluer cet outil pour appuyer son adoption hors du cadre expérimental, et améliorer la portée écologique des résultats. Ainsi nous considérons des scénarios tels que la génération d'images pour des sujets de TD, le support de graphes de grande taille, ou encore la visualisation d'algorithmes durant les cours.

References

1. Berry, J., Dean, N., Goldberg, M.K., Shannon, G.E., Skiena, S.: LINK: A system for graph computation. Software: Practice and Experience **30**(11), 1285–1302 (2000). https://doi.org/10.1002/1097-024X(200009)30:11¡1285::AID-SPE340¿3.0.CO;2-W
2. Chen, D.Y., Chuang, T.R., Tsai, S.C.: JGAP: A Java-based graph algorithms platform. Software: Practice and Experience **31**(7), 615–635 (2001). https://doi.org/10.1002/spe.379
3. Eaton, D.A.R.: Toytree: A minimalist tree visualization and manipulation library for Python. Methods in Ecology and Evolution **11**(1), 187–191 (2020). https://doi.org/10.1111/2041-210X.13313
4. Hundhausen, C.D., Douglas, S.A., Stasko, J.T.: A Meta-Study of Algorithm Visualization Effectiveness. Journal of Visual Languages & Computing **13**(3), 259–290 (Jun 2002). https://doi.org/10.1006/jvlc.2002.0237
5. Ministère de l'Éducation nationale: Programme de l'enseignement de spécialité de numérique et sciences informatiques de la classe terminale de la voie générale. https://www.education.gouv.fr/bo/19/Special8/MENE1921247A.htm (Jul 2019)
6. Schliep, A., Hochstättler, W.: Developing Gato and CATBox with Python: Teaching Graph Algorithms through Visualization and Experimentation. In: Borwein, J., Morales, M.H., Rodrigues, J.F., Polthier, K. (eds.) Multimedia Tools for Communicating Mathematics. pp. 291–309. Mathematics and Visualization, Springer, Berlin, Heidelberg (2002). https://doi.org/10.1007/978-3-642-56240-2_18