A. Talon : Optimiser un programme de génération de graphes : méthodologie et conseils

Alexandre Talon, G-SCOP, Grenoble, alexandre.talon@grenoble-inp.fr

Cette proposition n'est pas classique : il ne s'agit pas d'un nouveau résultat de recherche, mais de conseils de méthodologie d'optimisation d'un programme. Plus précisément, un programme de génération de graphes avec certaines propriétés. Que faire quand le programme qu'on a codé est beaucoup trop long (et/ou gourmand en mémoire) et ne permet pas de générer les graphes à n sommets pour n suffisamment grand ?

Le contexte de cet exposé est le suivant : dans le cadre de montrer qu'une sous-classe de graphes sans C_4 ni O_4 (ensemble indépendant de 4 sommets) est coloriable en temps polynomial, Cléophée Robin propose de trouver le plus possible de préfixeurs : des graphes sans C_4 ni O_4 ayant certaines propriétés supplémentaires. Il y avait donc besoin de générer tous les graphes de taille n sans C_4 ni O_4 pour n le plus grand possible.

Le point de départ est un programme en Python, utilisant la bibliothèque Python `graph_tool`. Problème : pour les graphes à 13 sommets, il met **3 semaines** ! Après un certain nombre d'optimisations la génération de taille 13 ne met plus que **40 secondes** !

La première étape a été celle avec le meilleur gain en terme de ratio : réécrire le programme en C++ permet d'avoir un temps de 2h30. D'autres optimisations incluent certaines optimisations algorithmiques : détection de C_4, utilisation de la notion de jumeaux dans l'énumération des nouveaux graphes pour en énumérer moins, améliorer l'algorithme de détection d'isomorphisme de raphes. D'autres optimisations sont plus à cheval entre algorithmique et aspects techniques : utilisation du faible nombre de sommets pour condenser les matrices d'adjacence, supprimer les listes d'adjacence, paralléliser le code. Enfin, certaines optimisations sont plutôt dans la catégorie technique : tester des propriétés à l'aide de et et de xor bit à bit, optimiser la taille des variables, et réutiliser au mieux des variables/structures allouées pour éviter trop d'allocations mémoire.

Il sera aussi question d'outils de profiling qui peuvent aider dans les choix des optimisations.

L'exposé n'apporte pas les bases de programmation mais peut être compris en très grande majorité par des personnes ne connaissant pas le C++.