Un court manuel de "Premierjet"

Alexis Tricot

18 septembre 2018

Table des matières

1 Introduction

Ce document est un court manuel d'explication de l'algorithme intitulé *premierjet*, qui est effectivement un premier jet. Ce programme a pour objectif de trouver des bassins d'attraction dans des réseaux biologiques modélisés par des réseaux d'automates. Il exploite une méthode non-exhaustive mais efficace. Il présente le fonctionnement de l'algorithme, les différentes fonctions qui le composent et des propositions pour d'éventuelles améliorations. On trouvera dans les fichiers *.h* des commentaires expliquant les procédures. Le code contenu dans les fichiers *.c* est également commenté, de sorte qu'il soit compréhensible. Je fais parfois référence dans ce manuel à mon rapport de stage, qui permet de comprendre les techniques utilisées dans l'algorithme. Cet algorithme a été développé lors de mon stage au LS2N à Nantes, sous la supervision de Monsieur Olivier Roux. En cas de question, contactez-moi : *alexis.tricot@polytechnique.edu*.

2 Utilisation

Le dossier contenant les fichiers de l'algorithme est muni d'un makefile, pour le compiler il suffit d'exécuter la commande *make*. Il vous faut ensuite exécuter le fichier **premierjet** ainsi créé. La fonction *main* se trouve dans le fichier *premierjet.c*. Pour changer de modèle, il faut décommenter une des lignes qui contiennent un "*getData*", premières lignes de la fonction main. Bien entendu, il faut ensuite recompiler pour exécuter avec le nouveau modèle. Les modèles se trouvent dans le dossier *models*. L'algorithme ne fonctionne que sur des réseaux d'automates binaires (deux états locaux).
Les résultats sont présentés sous la forme suivante : d'abord les différents bassins obtenus (l'ordre des automates est arbitraire), puis le nombre de bassins de chaque taille.

3 Description des différentes composantes de l'algorithme

3.1 Dossier *models*

Ce dossier contient différents réseaux d'automates, au format "*.an*". Tous ces réseaux ne sont pas forcément des réseaux d'automates binaires, aussi ils ne peuvent pas tous être utilisés par l'algorithme. Ils peuvent être utilisés une fois binarisés (automates à plus de deux états locaux transformés en plusieurs automates binaires, cf rapport).

3.2 Dossier *smchr*

L'algorithme inclut un autre algorithme de type SMT intitulé SMCHR [1]. Cependant, ce dernier n'est pas utilisé dans le calcul effectué par notre algorithme. Il est encore présent "au cas-où" seulement. Il peut être retiré pour accélérer la compilation, auquel cas il faut également adapter le makefile.

3.3 Dossier *solvers*

Comme pour le dossier *smchr*, ce dossier est obsolète. Il contient des solveurs SMCHR naïfs écrits par la fonction *writeSolver*.

3.4 *premierjet*

Contient la fonction *main*. Permet de choisir le modèle d'entrée de l'algorithme. Commenter les lignes 41 à 43 permet d'obtenir également les surbassins dans les résultats.

3.5 *getData*

Contient la fonction permettant de récupérer le modèle. Cette fonction permet de récupérer n'importe quel réseau d'automate, les automates n'ont pas besoin d'être binaires.

3.6 *struct.h*

Contient la structure interne à l'algorithme d'un réseau d'automate.

3.7 *simplify*

Contient les fonctions permettant de simplifier le réseau d'automates. Les simplifications sont effectuées en fonction de l'état initial. Les simplifications suppriment des automates et des transitions mais le réseau résultant reste équivalent au réseau de départ pour le problème étudié ici.

3.8 *printAN*

Contient les fonctions permettant d'afficher dans la console des informations sur le réseau d'automates.

3.9 *init*

Permet d'initialiser les automates en les reliant à leurs transitions.

1. https ://www.comp.nus.edu.sg/ gregory/smchr/

3.10 *autolists*

Comprendre "listes d'automates". Ces listes permettent de représenter les bassins, ainsi que la liste des bassins trouvés. La structure de ces listes est contenue dans le fichier *autolists.h*. Le code comprend un certain nombre de fonctions agissant sur ces listes.

3.11 *transition*

La fonction *countTrans* calcule pour chaque transition de chaque automate sa fréquence de transition, ou fonction de dénombrement des transitions (cf rapport). Contient également une fonction pour supprimer une transition (utilisé lors de l'étape *simplify*).

3.12 *frequency*

La fonction *freqlocal* effectue le même travail que la fonction *countTrans*, mais pour un état local précis et sous des conditions (automates fixés dans un état local). Cette fonction est appelée à de nombreuses reprises lors de la recherche de bassins. Elle sert à sélectionner les automates ajoutés. Elle contient une approximation.

3.13 *automata*

Contient une fonction permettant de supprimer un automate (utilisé lors de l'étape *simplify*).

3.14 attractor

Contient les deux fonctions qui permettent effectivement de rechercher les bassins d'attraction. La première fonction permet de rechercher un bassin (le départ est différent en fonction des champs *frequency* des automates). La seconde fonction relance la recherche dans un bassin en utilisant la première fonction. Lorsqu'on ne trouve plus de sous-bassin, on conserve le dernier bassin trouvé et on modifie le champ *frequency* de l'automate à la plus petite fréquence afin de relancer la recherche depuis un état initial différent.

3.15 *writeSolver*

Ecrit un solveur SMCHR correspondant au réseau d'automates parsé dans le dossier *solvers*. Non-utilisé par l'algortihme et laissé "au cas-où".

3.16 *merge*

Non-terminé, devrait contenir les fonctions permettant de fusionner les bassins trouvés. Ces fonctions seront alors appelées depuis la fonction *main* afin

d'obtenir les bassins minimaux. La fonction *intersect* réalise l'intersection de deux bassins. Elle peut être utile à la réalisation de la fonction *merge*.

4 Perspectives d'amélioration

4.1 Fonction *merge*

L'amélioration la plus importante à apporter est sans doute la fonction de fusion des bassins. Cette fonction devra, à partir des bassins trouvés par l'algorithme, obtenir les bassins minimaux résulant des intersections entre les bassins. Cela permettra d'une part d'obtenir des bassins plus grands en nombre d'automates fixés dans un état local (donc plus petits en terme d'états globaux), et d'autre part d'avoir des résultats plus limités donc plus lisibles pour les grands réseaux. La difficulté réside dans le fait qu'il faut peut-être faire beaucoup d'intersections différentes.

4.2 Autres améliorations

Certains éléments de l'algorithme peuvent être modifiés ou améliorés. On peut peut-être trouver une façon de sélectionner les automates qui remplace la fonction de dénombrement de transitions. Un principe permettant de décider combien de fois on relance la recherche de bassins ou de sous-bassins serait également un apport intéressant.