Porgy : une plate-forme visuelle et interactive pour la réécriture de graphes

Porgy: a Visual and Interactive Graph Rewriting Tool

Bruno Pinaud, Jonathan Dubois et Guy Melançon

English Abstract—Graph rewriting systems (GRSs) operate on graphs by substituting local patterns according to a set of rewriting rules. The apparent simplicity of GRSs hides an incredible complexity and turns the study of these systems into an involved task requiring high-level expertise. We designed PORGY, an interactive visual environment to fully support GRSs related tasks, exploiting a long historical tradition of GRSs with node-link representations of graphs. PORGY enables rule-based modeling and simulation steering through graphical representations and direct manipulation of all GRSs components. This talk presents PORGY and some case studies mainly based on bioinformatics data.

1 Introduction

Les systèmes de réécriture de graphes sont faciles à décrire et à expliquer. Imaginez un jeu dans lequel des règles de transformation sont successivement appliquées sur un graphe jusqu'à atteindre une condition d'arrêt. Une règle décrit un motif local (un sous-graphe) qui doit être identifié dans le graphe et comment transformer ce motif. Le formalisme de la réécriture de graphes est à la fois très riche et complexe rendant l'étude d'un système utilisant ce formalisme difficile. Par exemple, prédire si une suite de règles est applicable dans n'importe quel ordre est bien souvent un problème difficile. Pour la modélisation de systèmes complexes, les formalismes graphiques ont des avantages certains : ils sont intuitifs et rendent plus facile le raisonnement sur le système. Cette présentation décrit la conception d'un système visuel et interactif pour la modélisation de systèmes complexes basé sur la réécriture de graphe, leur simulation et le raisonnement sur ces systèmes. PORGY[1] (Figure 1) est le résultat de trois ans de travail collaboratif avec des experts des systèmes de réécritures. La méthodologie suivie est largement inspirée du travail de Munzner sur le développement et la validation de plate-formes de visualisation [3].

2 Les bases de la réécriture de graphe

Chaque modification autorisée (topologie, attributs des sommets/arêtes) sur un graphe donné G est appelée une *règle de réécriture*. Elles servent à modéliser les connaissances de l'expert sur le système étudié et décrivent comment un (petit) sous-graphe de G doit être modifié ou *réécrit*. Des modifications plus complexes sont réalisées en combinant des séquences de règles avec des opérateurs spécifiques dans une *stratégie*. Une stratégie décrit qui/quand/où/comment appliquer les règles [2]. Plus formellement, une règle de réécriture est de la forme $r_i : L_i \rightarrow R_i$ où L_i et R_i sont des graphes appelés respectivement le *membre gauche* et le *membre droit* de la règle r_i. La figure 1, partie 2 est un exemple. Le dessin permet de déduire immédiatement l'effet de cette règle : si un graphe contient un sous graphe avec cinq sommets dans la configuration donnée par la règle, alors la connexion entre les sommets bleu et vert est supprimée.

3 Fonctionnalités

Le premier souhait de nos utilisateurs a été de pouvoir concevoir et éditer graphiquement des règles. Le challenge principal est alors de savoir si la règle ainsi construite est un modèle correct du système modélisé. Donc, le second souhait a ensuite été de pouvoir observer l'évolution du système modélisé en effectuant des simulations après avoir défini une stratégie de réécriture (voir la figure 2). L'historique des calculs est géré par l'arbre de dérivation (figure 1, partie 4 et figure 3). Ses sommets sont les différents états pris par le graphe en cours de réécriture. Une arête noire indique une application de règle alors qu'une arête verte indique le point de départ et l'état final d'une stratégie. Chaque sommet de l'arbre de dérivation peut servir de point de départ pour l'application d'une nouvelle stratégie et ainsi créer une nouvelle branche. Pour la suite de ce résumé, le terme *graphe* sera réservé au graphe sur lequel les règles sont appliquées.

La complexité du système de réécriture est d'une certaine façon capturée par l'arbre de dérivation, qui est donc un objet central pour l'étude d'un système

• *Université de Bordeaux, UMR 5800 CNRS LaBRI*
E-mail: prenom.nom@labri.fr

1. http://tulip.labri.fr/TulipDrupal/?q=porgy

de réécriture. La visualisation et la manipulation de l'arbre de dérivation se sont donc évidemment révélées être la priorité suivante de nos utilisateurs.

L'étude d'un système de réécriture nécessite de passer en permanence de vues locales (les règles, les graphes) à une vue globale de l'arbre de dérivation. Le principal enjeu est alors de comprendre comment le comportement global du système est dirigé par les règles qui définissent des modifications uniquement locales et une stratégie qui pilote l'application de ces règles. C'est en regardant précisément les graphes dans l'arbre de dérivation tant globalement que localement qu'un expert du domaine peut apprécier l'adéquation du modèle avec le système.

4 Implémentation

PORGY est basé sur la librairie Tulip spécialisée dans la manipulation et la visualisation de graphes [1]. Un des nombreux atouts de Tulip pour ce projet est qu'il permet de pouvoir facilement visualiser et manipuler un graphe de graphes (l'arbre de dérivation, figure 3). L'arbre de dérivation agit comme une structure de haut-niveau dont les sommets contiennent chacun un graphe (sur lesquels sont appliqués les règles). La structure de l'arbre encapsule l'historique des calculs, chaque branche étant déduite de l'application d'une suite de règles. Chaque sommet réfère donc à un graphe qui résulte de l'application d'une règle sur son parent. Chaque arête stocke les informations nécessaires pour pouvoir retrouver facilement la règle qui a été appliquée et où elle l'a été. En fait, nous sommes en train de manipuler un graphe dynamique qui évolue au fur et à mesure que les règles sont appliquées. Néanmoins, la structure de données doit permettre de pouvoir rapidement accéder à n'importe quel état du graphe. Afin de conserver une structure de données cohérente et évolutive, tous les graphes dérivés du graphe original doivent partager leurs éléments communs (sommets, arêtes et propriétés).

La recherche des instances du membre gauche d'une règle est une tâche fondamentale du système. Appliquer une règle $r : L \to R$ sur un graphe G nécessite l'identification d'un sous-graphe $H \subset G$ qui soit isomorphique à L. Donc, une règle r s'applique à un graphe G si il existe au moins un sous-graphe $H \subset G$ tel que L est isomorphe à H. Bien que le problème de la recherche d'un isomorphisme graphe/sous-graphe est connu pour être *NP*-complet dans le cas général (graphe non étiqueté), il devient plus facile dans notre cas (graphe étiqueté, motif à rechercher de petite taille le plus souvent). point to the same metanode (local confluence).

Les graphes sont le plus souvent dessinés avec un algorithme à modèle de forces mais ils peuvent nécessiter l'emploi d'algorithmes plus spécialisés afin de respecter des conventions de dessins spécifiques aux données utilisées. PORGY est capable d'utiliser facilement tous les algorithmes de dessins fournit avec Tulip. Le dessin des règles est aussi un problème complexe. Dans les articles théoriques sur la réécriture de graphes, les règles sont souvent représentées graphiquement avec une vue nœud-lien en utilisant une symétrie entre chaque membre afin de bien mettre en valeur les modifications opérées par la règle. L'algorithme que nous avons mis au point est de dessiner d'abord le membre droit afin d'obtenir un dessin qui soit le meilleur possible puis ensuite de dessiner le membre gauche par symétrie avec le membre droit autant que possible. Bien que nous l'appelions "arbre de dérivation", la présence des arêtes de stratégie (les arêtes vertes) font que nous avons un graphe orienté sans cycle. Nous avons donc utilisé un algorithme de type Sugiyama présent dans Tulip. Ainsi chaque niveau du graphe (la profondeur dans notre cas) correspond au nombre d'application de règles à effectuer pour l'atteindre.

5 Conclusion et travaux futurs

Ce travail s'inscrit dans le contexte d'un projet collaboratif avec des utilisateurs et des informaticiens spécialistes de réécriture. Nous avons donc pu faire de multiples séances de travail et observer l'utilisation des premiers prototypes afin de converger vers le choix de vues et d'interactions appropriées pour mettre en œuvre les tâches identifiées au début de ce travail. PORGY est disponible à http://tulip.labri.fr/TulipDrupal/?q=porgy. Il reste de nombreux problèmes à traiter notamment quand le membre gauche d'une règle est de taille importante ou encore les problèmes de stabilité dans le tracé des graphes dynamiques. Nous avons aussi en projet de généraliser PORGY pour l'ouvrir à de nombreux domaines d'application.

References

[1] D. Auber, D. Archambault, R. Bourqui, A. Lambert, M. Mathiaut, P. Mary, M. Delest, J. Dubois, and G. Mélançon. The Tulip 3 Framework: A Scalable Software Library for Information Visualization Applications Based on Relational Data. Technical Report RR-7860, Inria, Jan. 2012.

[2] M. Fernandez, H. Kirchner, and O. Namet. A strategy language for graph rewriting. In *Proc. of Logic-Based Program Synthesis and Transformation (LOPSTR)*, LNCS, p. To appear. Springer, 2012.

[3] T. Munzner. A nested process model for visualization design and validation. *IEEE Trans. on Visualization and Computer Graphics*, 15(6):921–928, 2009.

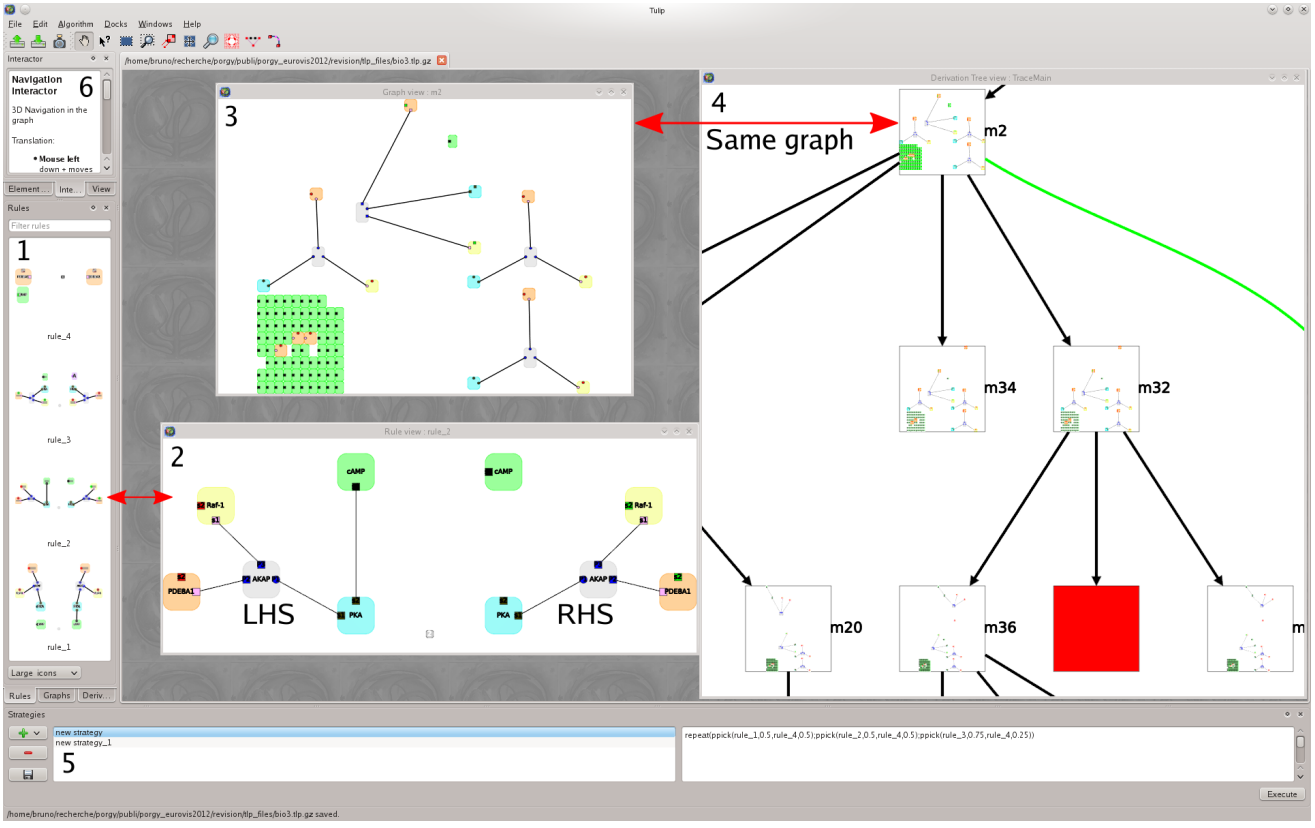
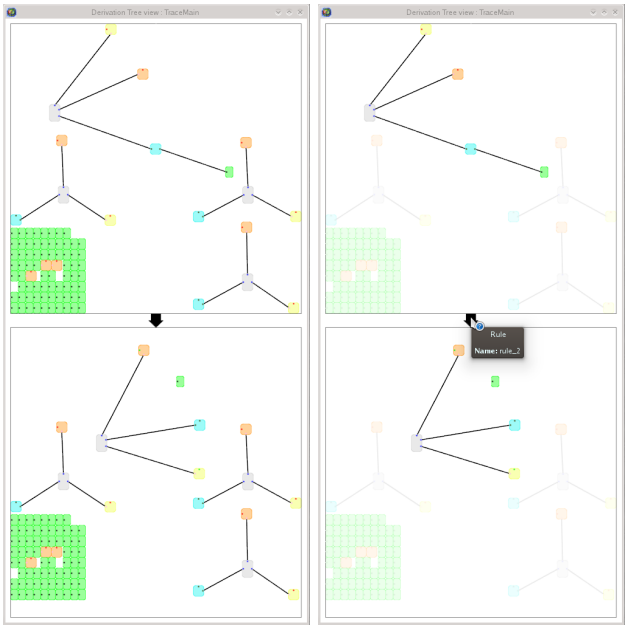


Fig. 1. Vue d'ensemble de PORGY: (1) les règles de réécriture disponibles ; (2) édition d'une règle ; (3) édition d'un état du graphe qui est en cours de réécriture ; (4) une partie de l'arbre de dérivation, un historique complet des calculs effectués ; (5) l'éditeur de stratégie ; (6) options de configuration.



(a) Vue classique (b) Visualisation des élements modifiés par la réécriture

Fig. 2. Une étape de réécriture (règle de la figure 1) avant et après avoir passé le pointeur de la souris sur l'arête qui relie le graphe d'origine (celui du haut) au graphe résultat (bas).

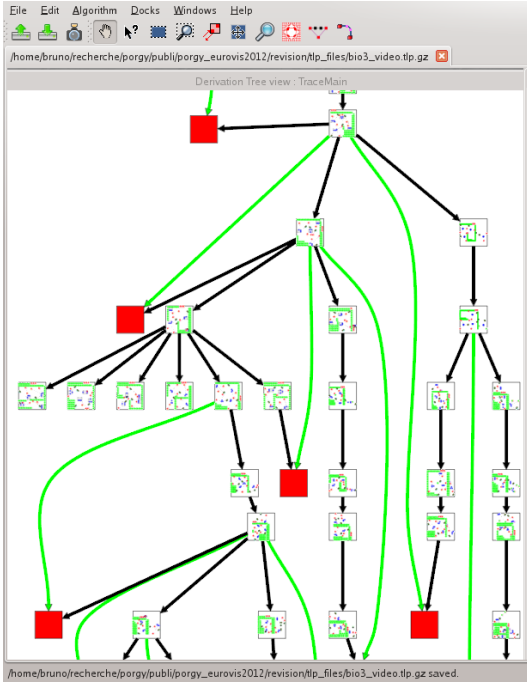


Fig. 3. Extrait d'un arbre de dérivation.