Recherche de chemins contraints dans les réseaux biochimiques

Grégoire Dooms — Yves Deville — Pierre Dupont

Département d'ingénierie informatique
Université catholique de Louvain
B-1348 Louvain-la-Neuve - Belgique
{dooms,yde,pdupont}@info.ucl.ac.be

RÉSUMÉ. *L'analyse des réseaux biochimiques est généralement implantée à l'aide de langages procéduraux ou relationnels. La combinaison ou la modification de ces programmes pour mener de nouvelles analyses peut être laborieuse et ce travail de modification ne peut généralement pas être réutilisé. Pour pallier ces problèmes nous introduisons, pour l'analyse des réseaux biochimiques, CP(BioNet) un nouveau domaine de calcul en programmation par contraintes. Les analyses sont formulées à l'aide de « variables domaine graphe » et de contraintes sur ces variables. Ces contraintes sont alors résolues par un résolveur de contraintes conçu pour l'analyse de réseaux biochimiques. Cette approche permet de combiner facilement des analyses simples pour en conduire des plus complexes. Nous nous focalisons ici sur la* recherche de chemins contraints, *trouver un chemin d'un noeud A à un noeud B avec des contraintes additionnelles comme le fait que ce chemin passe aussi par un ensemble de noeuds prédéfinis. Des nouvelles contraintes pour la recherche de chemins sont introduites et l'implantation de leurs propagateurs décrite. Un prototype illustrant les bénéfices de cette approche est présenté et les résultats d'expériences de recherche de chemins contraints sont discutés.*

ABSTRACT. *The analysis of biochemical networks is mainly done using relational or procedural languages. Combining or designing new analyses requires lot of programming effort that cannot be reused for other analyses. To overcome these limitations, we introduce CP(BioNet) a new constraint programming domain for the analysis of biochemical networks. Analyses are formulated using constraints over graph domain variables. The constraints are then solved by a constraint solver designed for biochemical networks. This provides a flexible and powerful approach as simple analyses can then easily be combined to form complex ones. We focus here on* Constrained path finding, *finding a path from node A to node B in a graph with additional constraints, such as requiring this path to include a predefined set of mandatory intermediate nodes. Constraints for path finding are introduced and their implementation (propagators) is described. A prototype is presented and constrained path finding experiments are performed and analyzed to illustrate the benefits of this new approach.*

MOTS-CLÉS : *Réseaux biochimiques, analyse de réseaux, programmation par contraintes, analyse de graphes*

KEYWORDS: *Biochemical networks, Network analysis, Constraint programming, Graph analysis.*

1. Introduction

Les réseaux biochimiques sont généralement subdivisés en trois catégories : les réseaux métaboliques, de régulation et de transduction de signal. Les réseaux métaboliques décrivent les protéines, les gènes, les réactions, *etc* et concernent les flux de matière dans les cellules. Dans les réseaux de régulation, l'attention est portée plus particulièrement sur l'interaction entre les différents contrôles (catalyse d'une réaction, régulation de l'expression d'un gène, inhibition d'une catalyse, *etc*) par exemple les boucles de rétroaction négative. Quant aux réseaux de transduction de signal, ils se focalisent sur les flux d'information dans la cellule et entre la cellule et son environnement, par exemple la façon dont l'information traverse la membrane cellulaire pour venir affecter l'expression d'un gène.

Ces réseaux biochimiques sont souvent représentés dans des bases de données spécialisées [DEV 03] dédiées à un type particulier de réseaux biochimiques. En l'occurrence, KEGG(LIGAND) (gènes, enzymes et réactions) [GOT 98] se focalise sur les réseaux métaboliques alors que la base de données BIND [BAD 01] s'intéresse aux interactions de protéines. D'autre part, la base de données EcoCyc [KAR 02] et le projet aMAZE [AM, LEM 04] utilisent des modèles de données intégrant les trois grandes catégories de réseaux biochimiques : les réseaux métaboliques, de régulation et de transduction de signal.

L'analyse des réseaux biochimiques est une tâche importante pour améliorer notre connaissance du fonctionnement de la cellule. Les analyses consistent à répondre à des requêtes (paramétrées) comme :

– trouvez le processus transformant A en B en mois de X étapes.

– trouvez les gènes dont l'expression est affectée par l'entité A.

– trouvez les composés produits à partir d'une entité A en moins de X étapes.

– trouvez les voies contenant une liste L d'entités, réactions, contrôles, *etc*.

Plusieurs projets (aMaze [AM], KEGG [MIN 02], BioCyc [KAR 02], Um-BBD [ELL 02], Emp [EMP], PathDB [SCH], CSNDB [TAK 99]) permettent de poser des requêtes prédéfinies similaires à celles présentées ci-dessus. De telles requêtes permettent d'exprimer plusieurs analyses grâce à la liberté de choix des paramètres (indiqués par des lettres majuscules dans nos exemples). Cependant ces requêtes sont généralement limitées à des analyses assez simples dont la réponse est fournie soit par le gestionnaire de base de données soit par des routines ad-hoc simples.

Des analyses plus compliquées sont intéressantes d'un point de vue biologique et leur implantation peut demander un effort de conception et d'implantation conséquent tout en couvrant un spectre d'analyses beaucoup plus restreint. La combinaison ou l'adaptation de programmes effectuant ces analyses demande un gros effort de programmation généralement difficilement réutilisable pour d'autres analyses.

Objectif

Nous proposons ici une approche en programmation par contraintes au problème de l'analyse de réseaux biochimiques. Une analyse est exprimée à l'aide d'un ensemble de contraintes ; celles-ci sont résolues par un résolveur de contraintes conçu pour l'analyse des réseaux biochimiques. Cette approche flexible et puissante permet de combiner facilement des analyses simples pour exprimer des analyses plus complexes. Dans cet article, nous nous focalisons sur la recherche de chemins, une large famille d'analyses. La *recherche de chemins contraints* consiste à trouver un chemin (sans répétition de noeuds) d'un noeud A à un noeud B dans un graphe avec des contraintes supplémentaires comme de contenir un ensemble de noeuds internes supplémentaires. Cette recherche fait partie du projet de recherche interdisciplinaire BioMaze où nous collaborons avec des biologistes.

Résultats

La première contribution de ce travail est l'introduction de CP(BioNet), un nouveau domaine de calcul en programmation par contraintes. Ce domaine de calcul est principalement destiné à l'analyse des réseaux biochimiques. Nous proposons les premières contraintes de ce domaine de calcul : un ensemble de contraintes pour la recherche de chemins contraints. Une première implantation, basée sur la plateforme libre de programmation par contrainte Oz-Mozart [Moz 02] est aussi décrite. Enfin, des résultats expérimentaux montrant la faisabilité de cette approche sont discutés.

Aperçu

La section 2 présente des informations préliminaires et un état de l'art sur la modélisation des réseaux biochimiques et la programmation par contraintes. La section 3 décrit notre approche pour CP(BioNet) : la définition d'un nouveau type de variables et de nouvelles contraintes sur ces variables. Ce nouveau domaine de calcul permet une formulation aisée d'analyses nouvelles et complexes. Les variables domaine graphe et leur implantation sont décrites dans la section 4. Les premières contraintes sur les variables domaine graphe sont décrites dans la section 5. La section 6 décrit nos expériences et discute leurs résultats. La section 7 conclut cet article et présente le travail courant et futur.

2. Prérequis

Cette section donne une introduction de base sur la modélisation des réseaux biochimiques utilisée dans ce travail. Un état de l'art y est aussi présenté.

2.1. *Modélisation de réseaux biochimiques*

Un modèle orienté-objet des réseaux biochimiques est développé dans le projet de base de données aMAZE [AM, van 00, van 01, DEV 03]. Ce modèle contient 3

classes d'objets principales dont les autres classes héritent : les *bio-entités*, les *transformations* et les *contrôles* ; celles-ci sont illustrées sur la figure 1.

Les *bio-entités* sont les molécules que l'ont retrouve dans les cellules : les protéines, composés, gènes, ARN messagers, enzymes, molécules minérales comme l'eau ou l'oxygène, *etc*. Les *transformations* sont des mécanismes faisant entrer en relation des bio-entités : la réaction d'un ensemble de molécules pour en donner un autre, la traduction de l'ARN messager en protéine, l'expression d'un gène en ARN messager, *etc*. Les *contrôles* sont des mécanismes où les bio-entités affectent des transformations ou d'autres contrôles : un enzyme catalysant une réaction, une protéine antagoniste d'une catalyse, une protéine régulant l'expression d'un gène, *etc*.

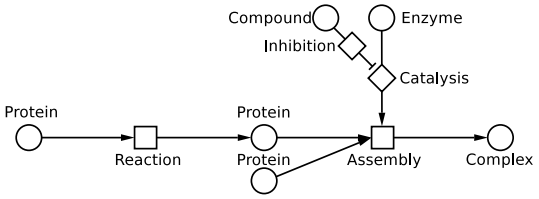

Figure 1. *Un petit réseau biochimique dans le modèle orienté-objet contenant des bio-entités, des transformations et des contrôles.*

Alors que le modèle orienté-objet est très efficace comme modèle de base de données, un modèle de graphe est plus approprié pour la recherche de chemins contraints. En effet, les réseaux biochimiques peuvent aussi être vus comme des graphes où les noeuds sont les objets des trois classes de base précitées et où un arc dénote une relation logique entre les noeuds. Des attributs sont associés aux noeuds. L'information contenue dans ces attributs est extraite de la base de données et varie en fonction de la famille d'analyses que l'on désire conduire. Ce modèle de graphe est plus riche que les graphes de réactions ou les graphes de composés où les noeuds sont respectivement restreints aux réactions et aux composés. Il permet des analyses spécialisées et complexes des réseaux biochimiques. Pour simplifier l'exposé, les graphes considérés dans la suite de cet article seront des graphes simples non-orientés. Les raisonnements et algorithmes exposés peuvent être étendus aux graphes simples orientés.

2.2. *Etat de l'art*

Un aperçu des modèles de données utilisés pour l'analyse des réseaux biochimiques peut être trouvé dans [DEV 03]. La plus part des approches existant pour l'analyse des réseaux biochimiques est basée sur des algorithmes de graphes ; les algorithmes implantés sont soit des algorithmes classiques de graphe ou des extensions ad-hoc de ces algorithmes. Un modèle de graphe simple a été utilisé dans [FEL 00, WAG 01] pour l'analyse de propriétés topologiques (connexité, longueur, propriétés statistiques, ...). Les graphes simples ont aussi été utilisés dans [OGA 00] pour la détection de groupes d'enzymes aux fonctions proches. Dans [JEO 00], les réseaux

métaboliques de 43 organismes sont modélisés sous forme de graphes bipartis afin de réaliser une analyse comparée systématique afin de montrer que ces réseaux ont les même propriétés topologiques d'échelle. Les graphes bipartis ont aussi été utilisés dans [HEL 01, van 02] pour l'analyse de réseaux métaboliques stockés dans KEGG. Les auteurs analysent des propriétés structurelles globales et proposent différentes opérations d'analyse de graphe comme la recherche de chemins à l'aide d'algorithmes classiques.

En bio-informatique, la programmation par contraintes a été utilisée pour des problèmes particuliers ; se référer à [BAC 01] pour un survol de la bio-informatique et des contraintes. Dans le contexte des réseaux biochimiques, la satisfaction de contraintes a été introduite pour résoudre des problèmes particuliers d'analyse de graphe comme la reconnaissance de patrons dans les graphes [RUD 00, LAR 00] et la modélisation de systèmes [BOC 02].

La définition d'un domaine de calcul spécifique pour les réseaux biochimiques, ainsi que l'introduction des variables domaine graphe est une nouveauté en programmation par contraintes. Toutefois, ces extensions tirent profit de techniques pré-existantes de programmation par contraintes comme les algorithmes de graphe dans l'implantation des contraintes globales [BEL 00], les variables ensemble finis [DOV 98, GER 97], et la planification de routes sous contraintes [QUE].

3. CP(BioNet)

L'analyse des réseaux biochimiques est généralement effectuée à l'aide de langages de requête de bases de données relationnelles (SQL) ou de langages procéduraux. SQL est utilisé pour les requêtes les plus simples et des algorithmes ad-hoc sont conçus et implantés lorsque la requête est trop complexe pour le gestionnaire de base de données (trop de jointures dans la requête). L'approche de SQL est essentiellement déclarative puisque le programmeur établit des critères à respecter par les valeurs stockées dans les tables de la base de données. D'un autre côté, dans l'approche des langages procéduraux, le programmeur écrit la séquence des opérations à effectuer pour calculer la solution à chaque requête. Bien que permettant de traiter des requêtes plus complexes, la seconde approche présente un gros inconvénient : une toute petite modification de la requête peut nécessiter de gros changements dans le programme ; un nouvel algorithme sera même parfois nécessaire.

L'utilisation de la programmation par contraintes, un paradigme déclaratif, résout partiellement ce problème. La plupart du temps, une légère modification de la requête demandera simplement le retrait ou l'addition de quelques contraintes dans le modèle. Dans le pire des cas, une nouvelle contrainte et son propagateur devront être conçus. La conception d'une nouvelle contrainte peut être difficile mais l'ajout d'une telle contrainte à l'ensemble des contraintes disponibles dans CP(BioNet) devrait permettre la formulation et la résolution d'un grand nombre de requêtes supplémentaires. Beaucoup de requêtes, comme celles listées à la section 1, peuvent être formulées comme

la recherche d'un sous-graphe (non-orienté) dans un graphe de référence (non-orienté lui aussi) représentant la base de données. Le problème de recherche d'un chemin contraint est l'un d'eux puisqu'un chemin est un type particulier de graphe. Le but de CP(BioNet) est de permettre des analyses complexes de réseaux biochimiques représentés sous forme de graphes. Dans CP(BioNet), une analyse est définie comme un ensemble de contraintes sur des *variables domaine graphe* (gd-variables). Le domaine d'une telle variable est un ensemble de graphes. Plusieurs analyses de réseaux biochimiques peuvent être formulées à l'aide de contraintes sur des gd-variables. Des requêtes telles que « trouvez le processus transformant A en B en moins de X étapes », « trouvez toutes les voies exprimées par un ensemble de gènes » ou « montrez comment l'expression du gène G est affectée par le composé A » sont des exemples typiques. Elles se traduisent, respectivement, en « trouvez un chemin entre le noeud A et le noeud B de longueur inférieure à X, ce chemin ne passant que par des entités et transformations », « trouvez le plus grand sous-graphe ne contenant pas d'autres gènes que ceux donnés et respectant des contraintes reliant les entités aux réactions et aux catalyses selon les règles sémantiques biochimiques habituelles (ex : absence d'une réaction si absence d'un de ses substrats ou de son catalyseur) », ou « trouvez les chemins entre un noeud de régulation attaché à l'expression du gène G et le noeud A ».

Le problème de recherche de chemins contraints illustre bien le bénéfice apporté par la nature déclarative de la programmation par contraintes. Une instance de ce problème est de trouver un chemin (sans répétition de noeuds) entre le noeud n_s (associé à la bio-entité A) et le noeud n_e (associé à la bio-entité B) avec la contrainte additionnelle que ce chemin contienne les noeuds d'un ensemble de noeuds donné $\{n_{i1}, n_{i2}, \ldots\}$. Ce problème peut être formulé à l'aide de deux contraintes simples (décrites dans la section 5) de CP(BioNet). Une approche procédurale serait significativement plus compliquée. Cette approche permet aussi d'exploiter l'information associée aux noeuds (attributs) pour contraindre le chemin. Par exemple on peut chercher un chemin contenant une réaction catalysée par un enzyme d'une certaine famille ou un chemin contenant au plus trois kinases.

Un algorithme permettant de trouver un chemin entre le noeud n_s et le noeud n_e est la recherche en largeur d'abord qui retourne le chemin le plus court entre ces deux noeuds. Avec l'addition d'un noeud intermédiaire obligatoire n_i le problème revient à trouver deux chemins disjoints entre n_s et n_i et entre n_i et n_e. Un algorithme trivial serait d'énumérer tous les chemins entre n_s et n_e et d'en sélectionner un qui passe par n_i. Toutefois, le nombre de chemins possibles est exponentiel en fonction du nombre de noeuds dans le graphe. Lorsque l'on considère plusieurs noeuds intermédiaires, l'ordre des noeuds intermédiaires dans le chemin doit être pris en compte ; le nombre de permutations de ces noeuds intermédiaire multiplie alors le nombre de chemins disjoints à considérer. Ce problème est en fait NP-difficile en fonction du nombre de noeuds intermédiaires puisque le problème du chemin hamiltonien (trouver un chemin qui passe par tous les noeuds d'un graphe exactement une fois) est une instance du problème des noeuds intermédiaires. La programmation par contraintes a été utilisée de

nombreuses fois pour traiter des instances de problèmes NP-difficile (TSP [CAS 97], TSPTW [FOC 99], ordonnancement [NUI 98], *etc*).

L'implantation actuelle de CP(BioNet) est un prototype illustrant notre approche. Elle consiste en une structure de données pour les gd-variables et une implantation des propagateurs des contraintes associées à ces variables.

4. Variables domaine graphe

La plupart des plateformes de programmation par contraintes supportent trois types de variables domaines : les variables booléennes, entières et ensembles finis. Lors de la déclaration d'une de ces variables l'utilisateur définit leur domaine initial. Le domaine initial d'une variable booléenne est $\{true, false\}$. Le domaine initial d'une variable entière est un intervalle (*e.g.* $X \in [0, 10]$ ou $Y \in [-100, 200]$). Lors de la recherche, des valeurs peuvent être enlevées de ce domaine initial. Le domaine des variables entières devient alors un ensemble fini d'entiers. Quant aux variables ensemble fini, leur domaine initial est l'ensemble de tous les sous ensembles d'un ensemble d'entiers (*e.g.* $\{1, 2, 3\}$). Il peut être représenté par l'élément maximum (par rapport à l'inclusion d'ensembles) de ces sous-ensembles (l'ensemble $\{1, 2, 3\}$ dans ce cas). Les variables ensemble fini sont souvent représentées par un vecteur de variables booléennes, chaque variable indiquant si une valeur est présente dans l'ensemble (*true*), n'est pas présente dans l'ensemble (*false*), ou pourrait être présente dans l'ensemble ({*true,false*}). Ce vecteur représente donc un nombre fini de sous ensembles de l'ensemble spécifié initialement.

Le domaine d'une *variable domaine graphe* (gd-variable) est un ensemble de graphes. Une gd-variable a un graphe de référence qui lui est associé. Celui-ci représente le maximum (par rapport à l'inclusion de graphes) des valeurs possibles de la gd-variable. Dans ce travail, on suppose que toutes les variables utilisées pour l'analyse d'un réseau biochimique ont le même graphe de référence donc le même domaine initial. Des analyses concernant la comparaison de réseaux biochimiques différents ne rentrent pas dans le cadre de ce problème.

Une gd-variable G peut être implantée à l'aide de variables booléennes. Une variable booléenne par noeud du graphe de référence spécifie si ce noeud est présent dans le domaine de la gd-variable. Ce vecteur de variables booléennes est dénoté $nodes(G)$. La présence d'arcs dans le domaine d'une gd-variable est actuellement encodée à l'aide d'une matrice d'adjacence de variables booléennes (voir figure 2). Si N désigne le nombre de noeuds dans le graphe de référence, chaque gd-variable est représentée avec $N^2 + N$ variables booléennes (en fait à peu près la moitié car la matrice est symétrique). Cette matrice est dénotée $adjMat(G)$. On associe à chaque variable booléenne une contrainte liant ses variables booléennes de manière à garantir que la présence d'un arc implique la présence de ses deux noeuds extrémités. Cette

contrainte peut être implantée à l'aide d'un ensemble de contraintes booléennes de la forme :

$$adjMat(G)_{ij} \Rightarrow nodes(G)_i \wedge nodes(G)_j$$

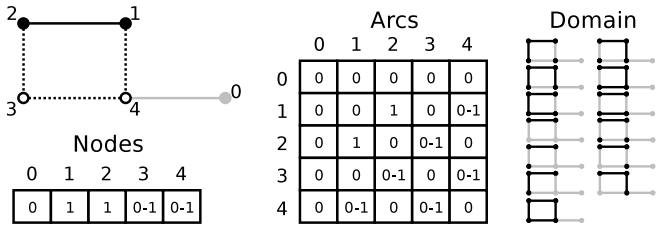

Figure 2. *Implantation d'une variable domaine graphe. Une image du domaine de la variable, prise au cours de la recherche, est représentée sous la forme d'un graphe et de tables de variables booléennes. Un noeud ou un arc est plein (noeuds 1 et 2 et l'arc les joignant) lorsqu'il est présent dans tous les graphes du domaine de la gd-variable. Un noeud ou un arc gris (noeud 0 et arc (0,4)) n'est inclus dans aucun graphe membre du domaine. Un arc pointillé ou un noeud creux (tous les autres arcs et noeuds) peut être présent ou absent dans les graphes membres du domaine. Tous les graphes membres du domaine courant de la gd-variable sont représentés sur la droite de la figure.*

5. Contraintes sur les variables domaine graphe

Les contraintes $NodeInGraph(n, G)$ et $ArcInGraph(a, G)$ sont les contraintes unaires les plus simples sur une variable domaine graphe G. Elles imposent respectivement la présence dans G du noeud n et de l'arc a du graphe de référence de G (voir figure 3). Ces contraintes ont un statut particulier dans CP(BioNet) car elles sont réifiées. Ceci signifie qu'elles peuvent être utilisées sous une forme négative pour spécifier qu'un noeud ou un arc ne doit pas faire partie de la variable domaine. Ces contraintes peuvent aussi être utilisées avec n'importe quel opérateur de la logique propositionnelle pour spécifier des contraintes plus complexes.

$Path(P, n_s, n_e, maxlength)$ est la contrainte à utiliser pour formuler un problème de recherche de chemins. Une gd-variable P satisfaisant cette contrainte est un chemin (sans répétition de noeud) du noeud n_s au noeud n_e dont la longueur (nombre d'arcs) est au plus $maxlength$ (voir figure 4). Le paramètre $maxlength$ peut être une valeur entière ou une variable domaine entière avec un domaine implicite $[1, N-1]$.

Quelques autres contraintes ont été conçues, implantées et utilisées dans des problèmes de recherche de chemins contraints. Elles sont décrites ci-dessous.

– La contrainte unaire $EveryArc(G)$ sur la gd-variable G indique que si deux noeuds sont inclus dans G et qu'un arc joignant ces noeuds existe dans le graphe de référence de G, alors cet arc doit être inclus dans G.

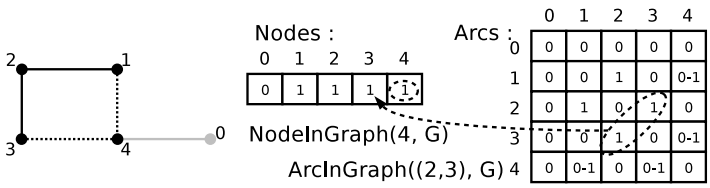

Figure 3. *Effet des contraintes $NodeInGraph$ et $ArcInGraph$: quelques variables domaine booléen deviennent vraies (1). L'effet de ces contraintes est entouré en lignes discontinues. La flèche illustre l'effet de la propagation de la contrainte de consistance inclue dans toutes les gd-variables (si un arc est présent chacune de ses extrémités doit l'être aussi).*

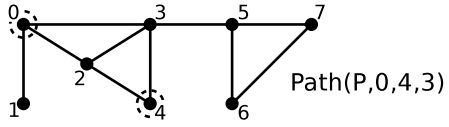

Figure 4. *La contrainte $Path$. La variable domaine graphe doit être un chemin (simple) du noeud 0 au noeud 4 et contenir au plus 3 arcs (au plus deux noeuds supplémentaires). Les noeuds 0 et 4 sont entourés dans le graphe de référence.*

– La contrainte binaire $SubGraph(P, G)$ sur les gd-variables P et G force P à être un sous-graphe de G (les noeuds et arcs de P doivent aussi être dans G). Comme indiqué précédemment ces deux gd-variables ont le même graphe de référence.

– Une contrainte $ExistsPath(G, n_s, n_e, maxlength)$ sur la gd-variable G, dérivé affaibli de $Path$, indique qu'un chemin de n_s à n_e doit exister dans le graphe G (mais G peut contenir d'autres noeuds et arcs). Cette contrainte est sémantiquement équivalente à l'introduction d'une nouvelle gd-variable P avec les contraintes $SubGraph(P, G)$ et $Path(P, n_s, n_e, maxlength)$. Toutefois, cette introduction de nouvelle variable serait beaucoup moins efficace.

– La contrainte unaire $Connected(G)$ impose que la gd-variable G soit un graphe connexe. Ceci est sémantiquement équivalent à poser une contrainte $ExistsPath$ entre toutes les paires de noeuds de G.

Un problème de recherche de chemins contraints est modélisé par un ensemble de contraintes : une contrainte $Path$ et d'autres contraintes sur le chemin pour contraindre davantage le problème. Le propagateur de $Path$ agit indépendamment des propagateurs des autres contraintes mais ils coopèrent en réduisant successivement le domaine de la gd-variable. Un simple problème de recherche de chemins sans contrainte supplémentaire est modélisé avec une seule contrainte $Path$. Un problème de recherche de chemins contraints tel que ceux utilisés dans les expériences (section 6) est modélisé à l'aide d'une contrainte $Path$ pour contraindre le sous-graphe extrait à être un chemin et une contrainte $NodeInGraph$ pour chaque noeud interne supplémentaire. Il est aussi possible de combiner des contraintes $Path$ par des contraintes

reliant les noeuds et arcs des différents chemins pour former des requêtes complexes. Un exemple est l'extraction de la voie métabolique TIM/PER à l'aide de 3 gd-variables P_1, P_2 et G :

$$Path(P_1, tim, TIM-PER), \forall r \in RegulNodes : \neg NodeInGraph(P_1, r)$$
$$Path(P_2, per, TIM-PER), \forall r \in RegulNodes : \neg NodeInGraph(P_2, r)$$
$$SubGraph(P_1, G),\ SubGraph(P_2, G),\ AllSubsProdsControls(G)$$

La contrainte $AllSubsProdsControls(G)$ indique que l'inclusion d'une réaction doit se faire avec tous les noeuds entités et contrôles qui lui sont incidents (ainsi que les noeuds contrôles et entités incidents aux contrôles). La recherche du plus petit G qui satisfasse à ces contraintes donne la voie métabolique TIM/PER.

5.1. *Implantation du propagateur de* $Path$

Cette section décrit l'implantation de la contrainte $Path(G, n_s, n_e, maxlength)$. Nous décrivons d'abord l'algorithme puis analysons sa complexité calculatoire.

5.1.1. *Algorithme*

Le propagateur de la contrainte $Path(P, n_s, n_e, maxlength)$ est implanté en trois parties. La première utilise des contraintes booléennes fournies par la plateforme Oz-Mozart [Moz 02]. La deuxième utilise des algorithmes standards de graphe. La troisième et dernière partie de cette implantation utilise des algorithmes de graphes plus poussés pour réduire plus avant le domaine de la gd-variable.

1) P est contraint à ne contenir que des noeuds de degré un ou deux. Les deux noeuds n_s et n_e ont un degré un et tous les autres noeuds inclus dans P sont contraints à avoir un degré deux. L'effet de ces contraintes est que le graphe P doit contenir un chemin (simple) entre les noeuds n_s et n_e. Toutefois elles permettent à P de contenir des cycles composés de noeuds non présents sur le chemin (dans la figure 4, un graphe P constitué du chemin de 0 à 4 et du cycle 5,6,7 satisfait ces contraintes de degré des noeuds). Cette première partie du propagateur est implantée à l'aide de contraintes de somme sur les lignes de la matrice d'adjacence de la gd-variable P forçant ces lignes à ne contenir que x (1 ou 2) variables booléennes dont la valeur est *true* (dans la somme, *true* vaut 1 et *false* vaut 0) :

$$\forall n \in \{n_s, n_e\} : \sum_j adjMat(P)_{n,j} = 1$$

$$\forall n \in nodes(P) \setminus \{n_s, n_e\} : \sum_j adjMat(P)_{n,j} = 2$$

Les cycles pouvant être présents dans d'autres composantes connexes seront évités par la deuxième partie du propagateur. Il est aussi possible de contraindre le nombre

de noeuds dans le chemin en utilisant l'information $maxlength$ puisqu'un chemin de longueur au plus $maxlength$ ne pourra contenir plus de $maxlength + 1$ noeuds :

$$\sum_i nodes(P)_i \leq maxlength + 1$$

2) P est contraint à ne contenir qu'une et une seule composante connexe. Ceci implique que P ne contienne que le chemin de n_s à n_e puisque les cycles mentionnés au point précédent se trouvent dans d'autres composantes connexes. Nous utilisons une structure de donnée nommée $ConnGraph$; il s'agit du suprémum (par rapport à l'inclusion de graphes) de tous les graphes du domaine de P. Un noeud ou un arc de P n'est pas présent dans $ConnGraph$ si et seulement si sa variable booléenne dans P vaut *false*. Si sa variable vaut *true* ou est indéterminée (*i.e.* {*true*,*false*}) alors ce noeud ou cet arc est présent dans $ConnGraph$.

A chaque fois qu'une variable booléenne associée à un arc dans la matrice d'adjacence prend une valeur *false*, on vérifie que tous les noeuds déjà inclus dans P appartiennent bien à la même composante connexe. Deux cas peuvent se présenter :

- la contrainte échoue s'ils ne sont pas dans la même composante connexe.

- sinon, tous les noeuds et arcs appartenant à toute autre composante connexe peuvent être exclus du domaine de P.

Une recherche en largeur d'abord avec limite de profondeur ($maxlength$) effectuée dans le graphe $ConnGraph$ réalise la vérification de composante connexe. Cette recherche collecte tous les noeuds présents dans la même composante connexe que n_s dans un rayon de $maxlength$ (si $maxlength$ est une variable domaine, la valeur maximale de son domaine est utilisée). Cette recherche est aussi utilisée pour vérifier la présence de cycles dans ce graphe. S'il ne contient pas de cycles, cette composante connexe partant de n_s est un arbre et l'on peut complètement déterminer la valeur de la gd-variable comme étant le seul chemin entre n_s et n_e dans cet arbre. Ceci est implanté à l'aide d'une recherche en profondeur d'abord entre n_s et n_e dans $ConnGraph$.

3) Les parties 1 et 2 garantissent de trouver une solution si elle existe. Une contrainte additionnelle améliore la propagation en détectant dès que possible que certains arcs doivent être ou ne pas être inclus dans le graphe P.

Un *isthme* dans une composante connexe est un arc dont la suppression casse la composante connexe en deux. On dit d'une composante connexe qu'elle est *2-arcs connectée* si elle ne contient pas d'isthme. Un algorithme de recherche des composantes 2-arcs connectées est utilisé pour trouver tous les isthmes de $ConnGraph$ [GRO 99, GON 95, COH]. Cet algorithme permet de construire $BridgeTree$, une structure de données supplémentaire représentant l'arbre dont les arcs sont les isthmes de $ConnGraph$ et dont les noeuds sont les composantes 2-arcs connectées de $ConnGraph$. Les noeuds représentant les composantes 2-arcs connectées contenant n_s et n_e sont étiquetés respectivement n_1 et n_2 (voir figure 5).

Dans ce $BridgeTree$, tous les arcs sur le chemin de n_1 à n_2 doivent être inclus dans P et tous les autres arcs (et les composantes 2-arcs connectées à leur bout) ne peuvent pas être inclus dans P. Cette information est propagée en ajoutant ou enle-

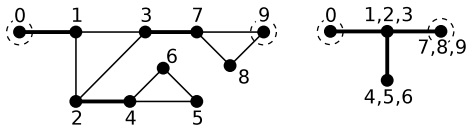

Figure 5. *A droite, l'arbre $BridgeTree$ comprend les isthmes et les composantes 2-arcs connectées du graphe de gauche. L'isthme (2,4) et la composante 2-arcs connectée 4,5,6 ne peuvent être présents dans le chemin de 0 à 9 alors que tous les autres isthmes doivent l'être.*

vant ces arcs et noeuds du domaine de P. Un raisonnement similaire peut être conduit à propos des noeuds d'articulation (noeuds dont la suppression casse la composante connexe en deux) et un algorithme unique s'occupe des isthmes et noeuds d'articulations sans complexité supplémentaire.

5.1.2. *Analyse de complexité*

A chaque suppression d'un arc du domaine de P,

– deux propagateurs de somme sont appelés (un pour chaque extrémité) ;

– une recherche en largeur d'abord (avec limite de profondeur) est conduite dans la structure $ConnGraph$ actualisée ;

– une recherche en profondeur d'abord peut être réalisée dans $ConnGraph$ pour trouver l'unique chemin possible (si $ConnGraph$ est un arbre) ;

– une recherche en profondeur d'abord est réalisée pour trouver les isthmes et les noeuds d'articulation de $ConnGraph$, puis une recherche en profondeur d'abord est réalisée dans $BridgeTree$.

Soit N le nombre de noeuds et A le nombre d'arcs dans le graphe de référence de P. A est clairement $O(N^2)$. Quand un arc est déterminé comme étant présent ou pas dans le domaine de P, au plus N propagateurs de somme sont exécutés. La complexité amortie de toutes les exécutions d'un propagateur de somme est $O(N)$ puisque nous sommes en présence d'un cas particulier de contrainte de cardinalité [Van 91]. Quand il ne reste qu'une possibilité de satisfaire la contrainte de cardinalité (*e.g.* pour n_s quand une seule variable vaut *true* et toutes les autres sont indéterminées ou *false*, les indéterminées peuvent être mises à *false* et quand une seule variable est indéterminée et que toutes les autres sont *false*, l'indéterminée peut être mise à *true*), le propagateur de la somme réduit, en une fois, le domaine de toutes les variables de la somme à un singleton. Quand il ne réduit pas ces domaines, il s'effectue en $O(1)$. Comme il y a N propagateurs, la complexité totale est $O(N^2)$ pour une assignation complète.

Toutes les recherches dans les graphes se font en temps $O(A)$ par suppression d'arc. Comme il y a au plus A suppressions d'arc, on obtient une complexité de $O(A^2)$ par assignation complète. Ceci nous conduit à une complexité de $O(max(N^2, A^2))$ par assignation complète.

La troisième partie du propagateur, calculant les isthmes et points d'articulation, a la même complexité calculatoire que le reste du propagateur. L'expérience montre que l'introduction de cette contrainte additionnelle ne pénalise pas le temps d'exécution lorsqu'elle n'apporte pas d'élagage, mais peut réduire le temps d'exécution lorsque des isthmes ou points d'articulation sont détectés.

6. Expériences

Des expériences pratiques ont été conduites pour s'assurer de l'efficacité de notre approche sur des données biologiques réelles. Des problèmes de tailles variées ont été conçus pour étudier la complexité de plusieurs variantes du problème de recherche de chemins contraints.

6.1. *Implantation du prototype CP(BioNet)*

Comme indiqué précédemment, CP(BioNet) est implanté au dessus de la plateforme de programmation par contraintes Oz-Mozart [Moz 02]. Le langage Oz est un langage de programmation multi-paradigmes (fonctionnel, logique, concurrent, orienté-objet, réparti) supportant la programmation par contraintes. La plateforme Oz-Mozart est l'implantation libre du langage Oz. Dans cette plateforme les propagateurs des contraintes supportées par la plateforme sont implantés en C++ et s'exécutent comme des threads Oz. La recherche dans l'espace des assignations se fait en clonant des espaces de calculs (avec les threads, variables logiques et variables d'état) et en imposant, sur une variable, une contrainte supplémentaire dans chaque branche de la recherche. Des « distributeurs » permettent de programmer les choix effectués lors de ces branchements. Les distributeurs et les moteurs de recherche sont écrits en Oz, rendant les extensions ou de nouvelles définitions aisées.

Le prototype est implanté en Oz et fait actuellement 500 lignes. Une classe est définie pour les gd-variables et une autre pour $ConnGraph$. Les contraintes sur les gd-variables sont des méthodes de la classe gd-variable. Les méthodes sont implantées de préférence dans un style fonctionnel mais certaines parties du propagateur de $Path$ nécessitent de modifier un état, d'où l'usage de cellules et dictionnaires. La matrice d'adjacence de chaque gd-variable est encodée sous forme d'un tuple de tuples contenant des variables booléennes.

6.2. *Données*

Des graphes de taille croissante (50, 100, 200, et 500 noeuds) ont été extraits d'un réseau métabolique comportant 4492 entités chimiques et 5281 réactions. Ces données proviennent du projet KEGG et concernent deux organismes : *Escherichia Coli* et *Saccharomyces cerevisiae*. L'extraction de graphes de plus petite taille a été conduite en respectant approximativement la distribution des degrés des noeuds du graphe de

référence. Pour être plus précis, un graphe extrait est une composante connexe dont le degré moyen des noeuds est environ 4 et dont le degré maximum est au moins de 18 pourcents de son nombre de noeuds.

6.3. *Tests et résultats*

Cinq tests ont été conduits sur les graphes extraits. Ce sont des problèmes de recherche de chemins exprimés dans CP(BioNet) à l'aide de la contrainte $Path$. Pour son paramètre $maxlength$, le nombre de noeuds dans le graphe a été utilisé (pas de contrainte sur la longueur du chemin trouvé).

1) Recherche de chemins entre deux noeuds pris au hasard dans le graphe (il y a toujours une solution puisque le graphe est connexe).

2) Recherche de chemins entre deux noeuds pris au hasard dans le graphe avec la contrainte supplémentaire de contenir deux autre noeuds préalablement choisis au hasard dans le graphe.

3) Recherche de chemins entre deux noeuds non joignables pris au hasard dans un graphe double (deux composantes connexes séparées ont été créées en clonant le graphe extrait ; jamais de solution).

4) Recherche de chemins entre deux noeuds pris au hasard dans le graphe avec la contrainte supplémentaire de contenir entre un et cinq autre noeuds préalablement choisis au hasard dans le graphe.

5) Sélection d'un chemin p de k noeuds au hasard dans le graphe. Recherche de chemins entre le premier et le dernier noeud de p, avec la contrainte supplémentaire de contenir entre un et $k-2$ autres noeuds choisis au hasard dans p (toujours une solution).

Le temps de calcul de chaque requête a été mesuré. Pour les trois premiers tests, mille requêtes ont été effectuées sur chacun des graphe extraits. Les quatrième et cinquième tests ont été conduits sur le graphe de 200 noeuds. Pour le quatrième test, mille requêtes ont été effectuées pour chaque nombre de noeuds intermédiaires. Pour le cinquième, on a procédé comme pour le quatrième pour k valant 7, 10 et 15.

La figure 6 montre la moyenne et l'écart type du temps pris pour la recherche du chemin pour chacun de ces tests. Les résultats des tests 2 et 4 sont séparés en deux courbes : une courbe pour les requêtes ayant conduit à une solution et une courbe pour les requêtes où le système a déterminé qu'il n'y avait pas de solution.

6.4. *Analyse*

Les tests 1 et 3 concernent la simple recherche de chemins dans un graphe (sans contrainte supplémentaire). Ce problème n'est pas pertinent pour l'analyse de réseaux biochimiques et des algorithmes dédiés sont évidemment plus efficaces. Ces tests ont été réalisés pour analyser le propagateur de $Path$ seul. En ce qui concerne le test 3, la

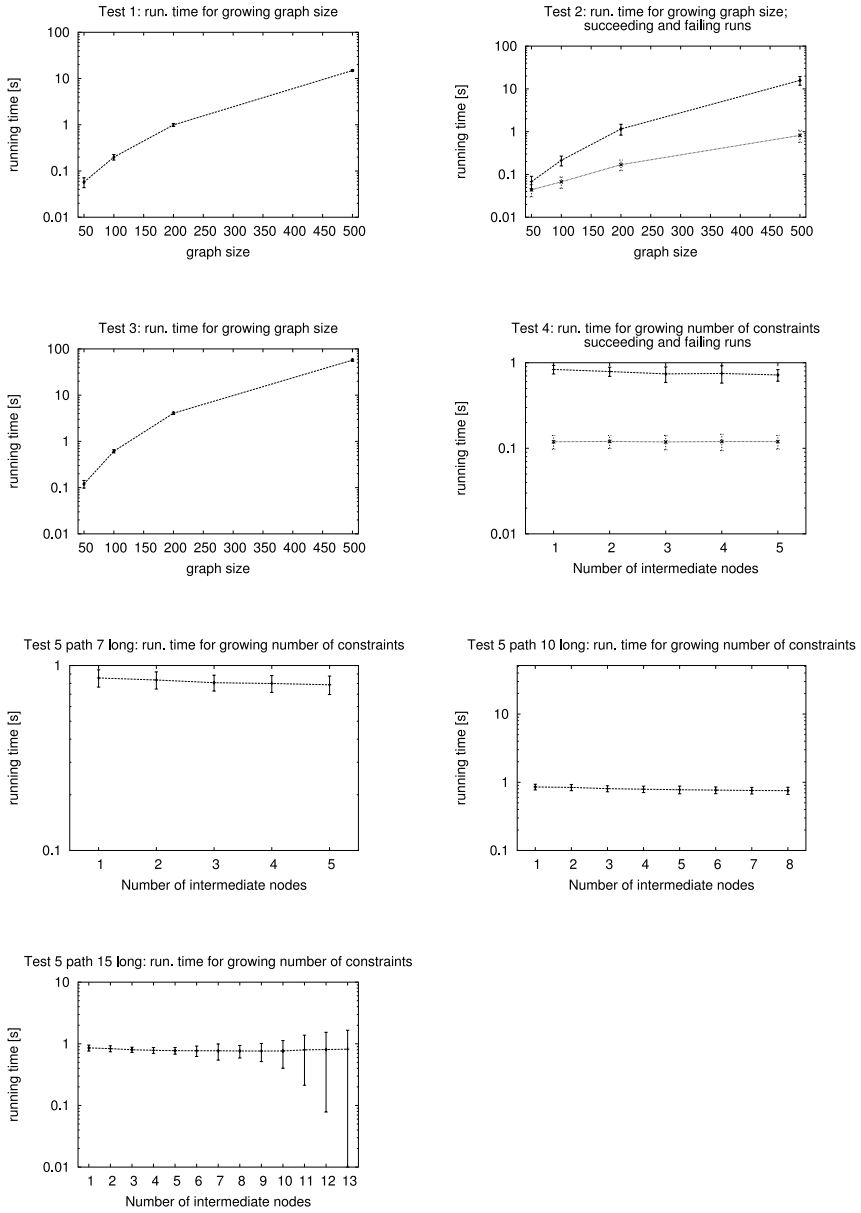

Figure 6. *Temps d'exécution des cinq tests. Echelle logarithmique sur l'axe des ordonnées.*

taille indiquée en abscisse est la taille d'une composante connexe du graphe (le graphe fait deux fois cette taille). Les graphiques de ces tests montrent une croissance sous-exponentielle et de faibles écarts types. Ces tests montrent que ce propagateur reste efficace pour des tailles croissantes de graphes.

Les tests 2, 4 et 5 concernent le problème de recherche de chemins contraints. Deux paramètres sont pris en compte dans cette analyse : la taille du graphe et le nombre de noeuds intermédiaires. Le test 2 montre l'évolution du temps de calcul d'une requête avec deux noeuds intermédiaires en fonction de la taille du graphe. Le graphique montre deux courbes : celle du dessus représente les requêtes fructueuses (le résolveur de CSP a trouvé une solution) et celle du dessous représente celles qui ont raté (le résolveur de CSP n'a pas trouvé de solution). Les résultats montrent que ces courbes sont similaires à celles où seul le propagateur de $Path$ tournait. La seule différence notable étant une augmentation de l'écart type.

Les tests 4 et 5 montrent l'évolution du temps de calcul sur un graphe de 200 noeuds en fonction du nombre de noeuds intermédiaires. Le test a été conduit pour pouvoir montrer des résultats pour des requêtes fructueuses avec des nombres importants de noeuds intermédiaires. En effet, quand les noeuds intermédiaires sont choisis au hasard, les chances d'avoir une requête fructueuse sont maigres. Les graphiques montrent que le temps moyen de ces tests est presque constant alors que l'écart type a tendance à augmenter.

Une faible fraction des exécutions (entre 0.08% et 1% selon les tests) des tests de recherche de chemins contraints ont montré des temps d'exécution plus grand de plusieurs ordre de grandeur que la moyenne. Ceci illustre en quelque sorte le caractère NP-difficile de ces problèmes. Des graphiques avec et sans ces résultats sont comparés sur la figure 7.

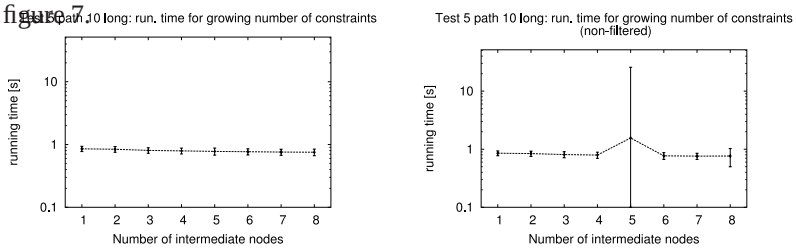

Figure 7. *Comparaison des résultats du test 5 avec une longueur de chemin de 10. Les résultats filtrés sont affichés à gauche et ceux non filtrés à droite. La différence provient de 2 exécutions (parmi les 8000 présentées) pour 5 noeuds intermédiaires : la première a duré 765 s et la seconde 18 s. L'écart type est plus affecté par ces résultats que la moyenne.*

Nos résultats montrent que la contrainte $Path$ est efficace lorsqu'utilisée seule, bien que des algorithmes spécialisés soient plus efficaces. Lorsqu'on utilise celle-ci avec d'autres contraintes d'inclusion de noeuds (pour spécifier un problème NP-difficile), le temps d'exécution moyen ne varie presque pas (mis à part quelques rares résultats divergeant) par rapport à la contrainte $Path$ seule, indépendamment du nombre de noeuds supplémentaires. Des contraintes supplémentaires sur le type et

les attributs des noeuds du réseau biochimique peuvent donc être conçues et utilisées dans le cadre de CP(BioNet) pour la recherche de chemins contraints. CP(BioNet) pourra alors exploiter la richesse du modèle des réseaux biochimiques.

7. Conclusion

Cet article montre comment utiliser la programmation par contraintes (CP) pour l'analyse de réseaux biochimiques. Pour réaliser ces analyses, un nouveau domaine de calcul en CP a été présenté : CP(BioNet). Avec celui-ci, nous introduisons les variables domaine graphe (variable dont le domaine est un ensemble de graphes) et des contraintes sur ces nouvelles variables. Une implantation de CP(BioNet) est décrite ainsi que des expériences et analyses sur la recherche de chemins contraints. La recherche de chemins contraints est un problème NP-difficile et la programmation par contraintes est adaptée à ce type de problèmes. Ces expériences illustrent la faisabilité de l'approche. La nature déclarative de la programmation par contraintes facilite la construction de nouvelles analyses à partir de contraintes existantes. Nous pensons que la programmation par contraintes peut être utilisée pour réaliser de nombreuses analyses, différentes et complexes, de réseaux biochimiques dans un cadre unique et avec un modèle de données intégré.

Le travail en cours concerne des représentations plus efficaces en mémoire pour les variables domaine graphe, des améliorations de l'implantation des contraintes et la conception de nouvelles contraintes. La recherche dans l'espace d'états est aussi à l'étude avec notamment de nouvelles façons de choisir les variables lors de cette recherche.

Les variables domaine graphe pourraient bénéficier de meilleures structures de données ; les listes d'adjacence pourraient remplacer les matrices d'adjacence. Une autre possibilité concerne l'utilisation de structures de données semblables à celles utilisées pour les variables ensembles finis. La contrainte de chemin pourrait être généralisée avec l'utilisation de variables domaine au lieu de valeurs pour les paramètres de noeud de début et de fin. Ceci permettrait de chercher un chemin d'un noeud à un noeud non connu à l'avance, avec éventuellement d'autres contraintes sur ce chemin. On pourrait aussi chercher un chemin de n'importe quel gène à n'importe quel polypeptide passant par un ensemble donné de réactions.

Le propagateur de la contrainte de chemin pourrait être amélioré grâce à des algorithmes de graphe plus poussés ou à des algorithmes incrémentaux. La communauté des graphes dynamiques a produit des algorithmes très efficaces pour le test de connexité, de 2-arc connexité, de biconnexité et de détection des isthmes et noeuds d'articulation dans des graphes ou des noeuds et des arcs peuvent être ajoutés et retirés du graphe. Ces algorithmes ne recalculent pas toute l'information mais la mettent à jour ce qui est plus efficace [HOL 01].

De nouvelles contraintes vont aussi être implantées. Entre autres, citons une contrainte $NoCycle(G)$ qui impose qu'une gd-variable G ne contienne pas de cycle et

une contrainte $Tree(G)$ qui impose que G soit un arbre. Une contrainte ternaire *Diff(A,B,D)* sur les gd-variables A, B et D stipulant que D est la différence entre A et B est aussi à l'étude.

Un ensemble de contraintes spécifiques à l'analyse de réseaux biochimiques inclurait une contrainte « flux de matière » qui établirait que tous les substrats et produits d'une réaction doivent être présents dans le graphe si celle-ci y est. De même, un noeud de catalyse ne peut être présent sans sa réaction et son catalyseur. Une autre contrainte stipule qu'une réaction doit être dans le graphe si sa catalyse et son catalyseur s'y trouvent. Ces contraintes peuvent aussi être adaptées à une représentation des réseaux biochimiques sous forme de graphe orienté pour conduire des analyses où les réactions sont considérées comme orientées des substrats vers les produits.

Ce cadre CP(BioNet) avec ses variables domaine graphe et les contraintes sur ces variables devraient pouvoir être utilisés pour d'autres applications que l'analyse de réseaux biochimiques telles que l'analyse de réseaux d'interconnexion télécom. Ces applications pourraient bénéficier de l'implantation actuelle des contraintes existantes et nécessiter la conception de nouvelles contraintes.

Remerciements
Ces travaux de recherche sont financés par la Région Wallonne, projet BioMaze (WIST 315432). Merci à Pierre Flener et Patrick Prosser pour les discussions fructueuses. Merci aux membres du comité de lecture de JPLNC pour leurs commentaires.

8. Bibliographie

[AM] « The aMAZE data-base project », `http://www.amaze.ulb.ac.be/`.

[BAC 01] BACKOFEN R., GILBERT D., « Bioinformatics and Constraints », *Constraints*, vol. 6, n° 2/3, 2001, p. 141-156.

[BAD 01] BADER G., DONALDSON I., WOLTING C., OUELLETTE B., PAWSON T., HOGUE C., « BIND the biomolecular interaction network database », *Nucleic Acids Research*, vol. 29(1) :242-5, 2001.

[BEL 00] BELDICEANU N., « Global Constraints as graph properties on structured network of elementary constraints of the same type », Technical Report n° T2000/01, 2000, SICS.

[BOC 02] BOCKMAYR A., COURTOIS A., « Using hybrid concurrent constraint programming to model dynamic biological systems », *18th International Conference on Logic Programming*, vol. LNCS 2401, Springer, July 2002, p. 85–99.

[CAS 97] CASEAU Y., LABURTHE F., « Solving small TSPs with Constraints », *Proc. of the 14th International Conference on Logic Programming*, The MIT Press, 1997.

[COH] COHEN J., « Théorie des graphes et algorithmes », Course notes. `http://www.univ-paris12.fr/lacl/cohen/poly_gr.ps`.

[DEV 03] DEVILLE Y., GILBERT D., VAN HELDEN J., WODAK S., « An overview of data models for the analysis of biochemical networks », *Briefings in Bioinformatics*, vol. 4(3), 2003, p. 246–259.

[DOV 98] DOVIER A., PIAZZA C., PONTELLI E., ROSSI G., « On the Representation and Management of Finite Sets in CLP-languages », *Proc. of 1998 Joint Int. Conference and Symposium on Logic Programming.*, The MIT Press, 1998, p. 40-54.

[ELL 02] ELLIS L., HOU B. K., KANG W., WACKETT L., « The University of Minesota Biocatalysis/Biodegradation Database : post-genomic data mining », *Nucleic Acids Research*, vol. 31(1), 2002, p. 262–265.

[EMP] « EMP Project », Informations about EMP can be found at : `http://www.empproject.com/`.

[FEL 00] FELL D., WAGNER A., « Animating the cellular map », chapitre Structural properties of metabolic networks : implications for evolution and modeling of metabolism, p. 79-85, Stellenbosch University Press, 2000.

[FOC 99] FOCACCI F., LODI A., MILANO M., « Solving tsp with time windows with constraints », *ICLP'99 International Conference on Logic Programming*, 1999.

[GER 97] GERVET C., « Interval Propagation to Reason about Sets : Definition and Implementation of a Practical Language », *CONSTRAINTS journal*, vol. 1, 1997, p. 191-244.

[GON 95] GONDRAN M., MINOUX M., *Graphes et algorithmes*, Eyrolles, 1995, 3ème éd.

[GOT 98] GOTO S., NISHIOKA T., KANEHISA M., « LIGAND : Chemical Database for Enzyme Reactions », *Bioinformatics*, vol. 14, 1998, p. 591-599.

[GRO 99] GROSS J., YELLEN J., *Graph Theory and its Applications*, CRC Press, 1999.

[HEL 01] VAN HELDEN J., GILBERT D., WERNISCH L., SCHROEDER M., WODAK S., « Applications of regulatory sequence analysis and metabolic network analysis to the interpretation of gene expression data », GASCUEL O., SAGOT M.-F., Eds., *Computational Biology : First International Conference on Biology, Informatics, and Mathematics, JOBIM 2000*, vol. LNCS 2066, Springer, 2001, p. 155-172.

[HOL 01] HOLM J., DE LICHTENBERG K., THORUP M., « Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity », *Journal ACM*, vol. 48(4), 2001, p. 723-760.

[JEO 00] JEONG H., TOMBOR B., ALBERT R., OLTVAI Z., BARABASI A., « The large-scale organization of metabolic networks », *Nature*, vol. 406, 2000, p. 651–654.

[KAR 02] KARP P., RILEY M., SAIER M., PAULSEN I., COLLADO-VIDES J., PALEY S., PELLIGRINI-TOOLE A., BONAVIDES C., GAMA-CASTRO S., « The EcoCyc Database », *Nucleic Acids Res.*, vol. 30(1), 2002, p. 56–8.

[LAR 00] LARROSA J., VALIENTE G., « Graph pattern matching using constraint satisfaction », *Proc. Joint APPLIGRAPH/GETGRATS Worksh. Graph Transformation Systems*, 2000, p. 189–196.

[LEM 04] LEMER C., ANTEZANA E., COUCHE F., FAYS F., SANTOLARIA X., JANKY R., DEVILLE Y., RICHELLE J., WODAK S. J., « The aMAZE LightBench : a web interface to a relational database of cellular processes », *Nucleic Acids Res*, vol. 32, 2004, p. D443-D448.

[MIN 02] MINORU K., SUSUMU G., SHUICHI K., AKIHIRO N., « The KEGG databases at GenomeNet », *Nucleic Acids Research*, vol. 30(1), 2002, p. 42–46.

[Moz 02] MOZART CONSORTIUM, « The Mozart Programming System version 1.2.5 », December 2002, `http://www.mozart-oz.org/`.

[NUI 98] NUIJTEN W., PAPE C. L., « Constraint-based job shop scheduling with ILOG SCHEDULER », *Journal of Heuristics*, vol. 3, 1998, p. 271-286.

[OGA 00] OGATA H., FUJIBUCHI W., GOTO S., KANEHISA M., « A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters », *Nucleic Acids Res.*, vol. 28, n° 20, 2000, p. 4021-8.

[QUE] QUESADA L., « Solving constrained path problems in non-monotonic environments », UCL/INGI/INFO Research Report 2004-03.

[RUD 00] RUDOLF M., « Utilizing Constraint Satisfaction Techniques for Efficient Graph Pattern Matching », EHRIG H., ENGELS G., KREOWSKI H.-J., ROZENBERG G., Eds., *6th International Workshop on Theory and Application of Graph Transformations*, vol. 1764, Berlin, 2000, Springer-Verlag.

[SCH] SCHILKEY F., « PathDB : a pathway database », http ://www.ncgr.org/pathdb.

[TAK 99] TAKAI-IGARASHI T., KAMINUMA T., « A Pathway Finding System for the Cell Signaling Networks Database », *Silico Biology*, vol. 1, 1999, p. 129-146.

[Van 91] VAN HENTENRYCK P., DEVILLE Y., « The Cardinality Operator : A New Logical Connective for Constraint Logic Programming », FURUKAWA K., Ed., *ICLP'91 Proceedings 8th International Conference on Logic Programming*, MIT Press, 1991, p. 745–759.

[van 00] VAN HELDEN J., NAIM A., MANCUSO R., ELDRIDGE M., WERNISCH L., GILBERT D., WODAK S., « Representing and analyzing molecular and cellular function using the computer », *Journal of Biological Chemistry*, vol. 381(9-10), 2000, p. 921–35.

[van 01] VAN HELDEN J., NAIM A., LEMER C., MANCUSO R., ELDRIDGE M., WODAK S., « From molecular activities and processes to biological function », *Briefings in Bioinformatics*, vol. 2(1), 2001, p. 81-93.

[van 02] VAN HELDEN J., WERNISCH L., GILBERT D., WODAK S., « Graph-based analysis of metabolic networks », *Bioinformatics and genome analysis*, p. 245-274, Springer-Verlag, 2002.

[WAG 01] WAGNER A., FELL D., « The small world inside large metabolic networks », *Proc. of the Royal Soc. of London, Series B, Biol. Science*, vol. 268(1478), 2001, p. 1803-10.