CP(Graph): Introducing a Graph Computation Domain in Constraint Programming

Gregoire Dooms, Yves Deville, Pierre Dupont

Department of Computing Science and Engineering
Université catholique de Louvain
B-1348 Louvain-la-Neuve - Belgium
{dooms,yde,pdupont}@info.ucl.ac.be*

Abstract. In an increasing number of domains such as bioinformatics, combinatorial graph problems arise. We propose a novel way to solve these problems, mainly those that can be translated to constrained subgraph finding. Our approach extends constraint programming by introducing CP(Graph), a new computation domain focused on graphs including a new type of variable: graph domain variables as well as constraints over these variables and their propagators. These constraints are subdivided into kernel constraints and additional constraints formulated as networks of kernel constraints. For some of these constraints a dedicated global constraint and its associated propagator are sketched. CP(Graph) is integrated with finite domain and finite sets computation domains, allowing the combining of constraints of these domains with graph constraints.
A prototype of CP(Graph) built over finite domains and finite sets in Oz is presented. And we show that a problem of biochemical network analysis can be very simply described and solved within CP(Graph).

1 Introduction

Combinatorial graph problems are present in many domains such as communication networks, route planning, circuitry, and recently bioinformatics. The motivation for this work lies in graph problems of biochemical network analysis. Biochemical networks model the components of the cells (molecules, genes, reactions, etc...) and their interactions. They can be modeled as directed labeled graphs. Their analysis consists in assessing the properties of these graphs. Various problems have been solved to better understand the structure of the biochemical networks [1]. Some of these problems can be modeled as constrained path finding or constrained subgraph extraction problems.

The analyses performed on biochemical networks are varied and evolve at a rapid pace. A declarative framework based on constrained programming could enable a quick expression and resolution of these problems. It would allow the bioinformaticians to spend less time on implementing dedicated algorithms, keeping the focus on designing new queries and analyzing the results.

This paper introduces a graph computation domain, called CP(Graph), in constraint programming. A new type of domain variables, graph domain variables, and constraints

*This research is supported by the Walloon Region, project BioMaze (WIST 315432). Thanks also to the EC/FP6 Evergrow project for their computing support.

on these variables are proposed. CP(Graph) can then be used to express and solve combinatorial graph problems modeled as constrained subgraph extraction problems.

Related work CP(Graph) is built over the finite set computation domain [2]. It also shares its lattice structure. The usage of sets in a language able to express and solve hard combinatorial problems dates back to 1978 with ALICE in the seminal work of Laurière [3]. The usage of graphs as structures of symbolic constraint objects was proposed in 1993 by Gervet [4]. In that work, a graph domain is modeled as an endomorphic relation domain. In 2002, Lepape et al. defined path variables [5] which were used to solve constrained path finding problems in a network design context.

Graphs play an important role in constraint programming for the specification, design and implementation of global constraints [6], but graphs are there mainly used for representing and exploiting a network of elementary constraints. A global path constraint has been proposed in [7, 8] and a global tree constraint in [9]. A path constraint is included in CP(Graph) and its implementation is based on these related works. Finally, the theoretical framework is related to the work on edge set quantification in monadic second order logic of graphs [10] as our kernel constraint language on graphs also allows quantification on nodes and arcs. This work is also an extension of our preliminary work [11].

The first section presents the variables and constants used in CP(Graph) then the constraints linking graph domain variables to the other variables. The constraints of CP(Graph) can be separated into two classes: the kernel constraints (Section2.3) and the others. The kernel constraints form the minimal set of constraints necessary to express the other constraints as networks of kernel constraints. We show how to incrementally build graph constraints by combining kernel constraints for a specific class of problems: constrained subgraph extraction (Section 2.4).

The combination of kernel constraints is a rapid way of implementing other graph constraints. However, it is possible to achieve a better filtering by designing a so-called global constraint (Section 4). In order to characterize and compare the filtering of the propagators of the constraints in CP(Graph), we introduce *mixed consistency* in Section 3. It consists of bound consistency on sets and graph domain variables coupled with arc consistency on finite domain variables.

Finally, in Section 6, the practicality of CP(Graph) is assessed by expressing a CP(Graph) problem for a biochemical network analysis problem and by analyzing the evolution of computation time and memory usage with problems of increasing size.

Contributions the main contributions of this work are the following:

- graph domain variables, and constraints on these variables are the major contribution of this work. We show how to use them to express other constraints on graphs and to solve constrained subgraph extraction problems. We generalize the mode of usage of the reachability and path constraints by allowing end-nodes to be domain variables.
- Definition of a graph computation domain in CP.
- Specification of a minimal set of constraints on graphs.

– Suitability assessment for expressing and solving the class of constrained subgraph extraction problems.
– Practical assessment of the suitability of CP(Graph) for constrained subgraph extraction problems.

2 The CP(Graph) Framework

This section presents the basics of the CP(Graph) computation domain. Graph domain variables and domains are described along with their integration with finite sets and finite domains. Then, primitive constraints called kernel constraints are presented. Finally, more complex constraints are built using the kernel constraints. The construction of the their propagator and the analysis of their consistency is presented in section 3.

2.1 Constants and Variables

A graph $g = (sn, sa)$ is a set of nodes sn, and a set of arcs $sa \subseteq sn \times sn$. We are first considering directed graphs. An extension to undirected graphs is handled in a later section.

CP(Graph) introduces graph domain variables (gd-variables for short) in constraint programming. However, CP(Graph) deals with many types of constants and variables related to graphs. They are presented in Table 2.1. This table presents the notations used in this paper for constants and domain variables of each type. It also shows one particular aspect of graphs: the inherent constraint stating a arc can only be present if both end nodes are present too. Nodes and Arcs in CP(Graph) can be labeled with integer weights through the use of weight functions. Such functions are seen as constants in CP(Graph), there is no domain variable for weight functions. CP(Graph) can handle graphs with multiple weights per node or arc by using multiple weight functions.

Type	Representation	Constraint	Constants	Variables
Integer	$0, 1, 2, ...$		$i_0, i_1, ...$	$I_0, I_1, ...$
Node	$0, 1, 2, ...$		$n_0, n_1, ...$	$N_0, N_1, ...$
Arc	$(0, 1), (2, 4), ...$		$a_0, a_1, ...$	$A_0, A_1, ...$
Finite set	$\{0, 1, 2\}, \{3, 5\}$ ...		$s_0, s_1, ...$	$S_0, S_1, ...$
Finite set of nodes	$\{0, 1, 2\}, \{3, 5\}$ ...		$sn_0, sn_1, ...$	$SN_0, SN_1, ...$
Finite set of arcs	$\{(0, 3), (1, 2)\}, ...$		$sa_0, sa_1, ...$	$SA_0, SA_1, ...$
Graph	(SN, SA) SN a set of nodes SA a set of arcs	$SA \subseteq SN \times SN$	$g_0, g_1, ...$	$G_0, G_1, ...$
Weight functions	$\mathcal{N} \cup \mathcal{A} \rightarrow \mathbb{N}$		$w_0, w_1, ...$	–

Table 1: The different variables and constants of CP(Graph) along with their notations. Note only the graph has an inherent constraint. $\mathcal{N}$and$\mathcal{A}$ are the universal sets of nodes and arcs.

Similarly to sets, there exists a partial ordering among graphs, defined by graph inclusion: given $g_1 = (sn_1, sa_1)$ and $g_2 = (sn_2, sa2)$, $g1 \subseteq g_2$ iff $sn_1 \subseteq sn_2$ and $sa_1 \subseteq$

sa_2. We define graph domains as the lattice of graphs included between two bounds: the greatest lower bound and the least upper bound of the lattice.

The domain of each gd-variable is defined according to a least upper bound graph and a greatest lower bound graph. The least upper bound graph defines the set of possible nodes and arcs in the graph variable, while the greatest lower bound defines the set of nodes and arcs which are known to be part of the graph variable (see Figure 1). If G is a gd-variable, we will denote $dom(G) = [g_L, g_U]$ with $g_L = glb(G)$ and $g_U = lub(G)$. If S is a finite set variable, we denote $dom(S) = [s_L, s_U]$, with $s_L = glb(S)$ and $s_U = lub(S)$.

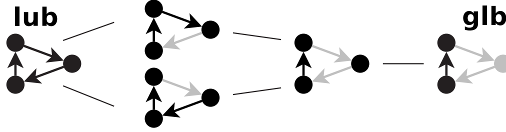

Fig. 1: Illustration of a small graph domain with its least upper bound (lub) and greatest lower bound (glb). Greyed nodes and arcs are displayed for convenience but are not part of the respective graphs.

The presence of arc variables and set of arc variables along with the nodes and set of nodes is motivated first by the works on expressiveness of monadic second order logic on graphs [10]. That work shows that a logic where it is possible to existentially quantify sets of nodes can be strictly less expressive than one where it is possible to existentially quantify sets of nodes and sets of arcs. Another incentive, in the constraint community, was the comparison of the models used in [7, 8] and [12] for a path constraint. Successor finite domain variables were used in [12] while [7, 8] use arc boolean variables. Yet the path propagator in [8] reasons about mandatory nodes. It is clear that providing only arc variables is impractical as a graph cannot be constrained to contain isolated nodes and constraints about nodes must be stated as disjunctions of arcs. Hence, offering node and arc variables enables to express more CSPs and properties about graph variables.

2.2 Classical Finite Set Constraints

CP(Graph) is integrated with the finite domain and finite set computation domains. Classical constraints from these domains can be combined with graph constraints to express a CSP in CP(Graph). We present the minimal standard set of constraints on finite domain and finite sets assumed to be present in the system.

The set constraints used in this paper are set inclusion ($S_1 \subseteq S_2$), set intersection ($S_1 \cap S_2 = S_3$), set difference ($S_1 \setminus S_2 = S_3$), set cardinality ($\#S = I$), set membership ($I \in S$ and $I \notin S$), set inequality ($S_1 \neq S_2$) and the set weight constraint ($Weight(S, w, I)$) which holds if I is the sum of the weights of the elements of S according to the weight description w. We also suppose it is possible to post a constraint for each value in a set variable S: $\forall i \in S : C(i)$. This can be done in two ways. Either by posting $\#s_U$ constraints of the form $i \in S \Rightarrow C(i)$ or by waiting until i is known to

be in S to post the constraint $C(i) : \sigma \models i \in S \rightarrow C(i)$ While the former filters more, the latter uses less memory.

In addition to the boolean constraints like implication, negation, conjunction and disjunction, we use the constraint of sum (linear combination of finite domain variables using constant factors) in CP(Graph).

2.3 Kernel Graph Constraints

The kernel graph constraints constitute the minimal set of constraints needed to express the other graph constraints of CP(Graph). These constraints relating graph variables with arc and node variables provide the suitable expressiveness of monadic second order logic [10].

The kernel graph constraints are $ArcNode$, $Nodes$ and $Arcs$.

$Arcs(G, SA)$ SA is the set of arcs of G.

$Nodes(G, SN)$ SN is the set of nodes of G.

$ArcNode(A, N_1, N_2)$ The arc variable A is an arc from node N_1 to node N_2. This relation does not take a graph variable into account as every arc and node has a unique identifier in the system. If A is determined, this constraint is a simple accessor to the tail and head of the arc A and respectively if both nodes are determined.

All CMS-definable sets of graphs [13] can be defined as constraints in CP(Graph) by using these kernel constraints (CMS stands for countable monadic second order logic). This can be shown by translating the building blocks of CMS logic of graphs into CP(Graph). Monadic means that only 1-ary relation (i.e. sets) can be quantified. CP(Graph) allows quantification over sets of nodes and arcs. Countable stands for a predicate telling the size of a set. It is handled by the set cardinality constraint. The **edg** binary relation on nodes and the incidence (**inc**) ternary relation on an arc and two nodes are expressed by $\mathbf{inc}_g(a, n_1, n_2) \equiv ArcNode(a, n_1, n_2) \wedge n_1 \in Nodes(g) \wedge n_2 \in Nodes(g) \wedge a \in Arcs(g)$ and $\mathbf{edg}_g(n_1, n_2) \equiv \exists a : ArcNode(a, n_1, n_2) \wedge \mathbf{inc}_g(a, n_1, n_2)$ where a is determined by n_1 and n_2. We do not know a common graph property which cannot be expressed using CP(Graph).

2.4 Building graph constraints over kernel constraints

While the kernel constraints enable to express the target problems of CP(Graph), defining higher level constraints eases the formulation of these problems. Such constraints can be built as combinations of kernel constraints. Such networks of constraints may not propagate as much as a dedicated global propagator for the constraint but are useful as a reference implementation or as a quickly implemented prototype. We focus here on constraints suitable for constrained subgraph extraction problems.

To alleviate the notation, we use a functional style for some constraints by removing the last argument of a constraint and considering that the resulting expression denotes the value of that omitted argument (e.g. $Nodes(G)$ denotes SN in $Nodes(G, SN)$). We also write $(n_1, n_2) \in Arcs(G)$ instead of $a \in Arcs(G) \wedge ArcNode(a, n_1, n_2)$.

The $SubGraph(G_1, G_2)$ constraint can be translated to

$$SubGraph(G_1, G_2) \equiv Nodes(G_1) \subseteq Nodes(G_2), Arcs(G_1) \subseteq Arcs(G_2)$$

To cope with linear optimization problems we introduce the $Weight$ constraint for graphs:

$Weight(G, w, I)$ holds if I is the total weight associated to the graph variable G according to the weight function w.

$$Weight(G, w, I) \equiv I = Weight(Nodes(G), w_n) + Weight(Arcs(G), w_a)$$

Where w_a is the restriction of the weight function to the arcs domain, and respectively, w_n for nodes. CP(Graph) allows to express and solve constrained subgraph optimization problems and some examples are given in the next section. A constrained shortest path problem is also presented in the experiments section.

$InNeighbors(G, N, SN)$ holds if SN is the set of all nodes of G from which an inward arc incident to N is present in G. If N is not in G then SN is empty. It can be expressed as the following network of constraints.

$$InNeighbors(G, N, SN) \equiv SN \subseteq Nodes(G) \wedge (\#SN > 0 \Leftrightarrow N \in Nodes(G)) \wedge \\ \forall n \in Nodes(g_U) : n \in SN \Leftrightarrow (n, N) \in Arcs(G)$$

The last constraint must be posted for all possible member of SN and for all possible in-neighbor of N. In this expression it is posted on a superset of these sets: $Nodes(g_U)$.

Similar expressions exist for inward arcs and the "out" versions of these constraints. $OutDegree$ and $InDegree$ are the cardinality of these sets.

$Reachable(G, N, SN)$ states SN is the set of nodes reachable from N in G. Again, G, N and SN are domain variables. This constraint is presented in [14] in the case of N determined. First we need to define the $QuasiPath(G, SN, N, n_2)$ constraints stating the graph induced by SN in G is a path from N to n_2 with possibly additional mutually disjoint cycles also disjoint from the path [10]. This is expressed by forcing every node in SN to have an inward and outward degree of 1 in the induced subgraph (except for the source N and sink n_2).

$$QuasiPath(G, SN, N_1, N_2) \equiv N_1 \in SN \wedge N_2 \in SN \wedge \\ \forall n \in SN : O = OutNeighbors(G, n) \cap SN \wedge \#O \leq 1 \wedge (n \neq N_2) \Rightarrow \#O = 1 \wedge \\ \forall n \in SN : I = InNeighbors(G, n) \cap SN \wedge \#I \leq 1 \wedge (n \neq N_1) \Rightarrow \#I = 1$$

Then $Reachables(G, N, SN)$ is expressed by:

$$\forall n \in SN : \exists SN' \subseteq Nodes(G) : QuasiPath(G, SN', N, n)$$

The directed acyclic graph constraint $DAG(G)$ states a graph cannot contain cycles. $DAG(G)$ can be translated using this property: the set of in-neighbors of each node must be disjoint from the set of nodes it can reach.

$$DAG(G) \equiv \forall n \in Nodes(G) : InNeighbors(G, n) \cap Reachable(G, n) = \emptyset$$

The path constraint can be expressed in a similar way:

$Path(G, N_1, N_2)$ holds if G is a path from node N_1 to node N_2, all of which are domain variables.

$$Path(G) \equiv QuasiPath(G, Nodes(G), N_1, N_2) \wedge \#Nodes(G) = \#Arcs(G)+1$$

The $InducedSubGraph(G_1, G_2)$ constraint is used in next section to express a k-cut problem.

$InducedSubGraph(G_1, G_2)$ holds if G_1 is an induced subgraph of the graph G_2 i.e. the greatest subgraph of G_2 containing the nodes of G_1.

$$InducedSubGraph(G_1, G_2) \equiv SN = Nodes(G_2) \setminus Nodes(G_1) \wedge$$
$$\forall (n1, n2) \in Arcs(G_2) : (n_1 \in SN \vee n_2 \in SN) \text{ XOR } (n1, n2) \in Arcs(G_1)$$

2.5 Combining Graph Constraints to Solve Problems

Numerous NP(Hard) graph problems can be stated in CP(Graph). The graph constraints presented in other works [6–8] can be implemented in the CP(Graph) framework and used to solve these problems. CP(Graph) is particularly suited for problems of subgraph extraction. We list a few example problems to show the expressiveness and conciseness of CP(Graph). In these expressions, $SubGraph(G, g)$ is used to declare a new graph domain variable G with initial upper bound g. The $Cycle(G)$ constraint holds if G is a closed directed path.

– Finding the TSP in graph g with weights w: minimize $Weight(G, w)$ s.t.

$$SubGraph(G, g) \wedge Cycle(G) \wedge Nodes(G) = Nodes(g)$$

– Finding the shortest weight constrained (maximum weight k) path of g with weights w, length function w_l, start node n_1, end node n_2: minimize $Weight(G, w_l)$ s.t.

$$SubGraph(G, g) \wedge Path(G, n_1, n_2) \wedge Weight(G, w) \leq k$$

– Finding the minimum vertex k-cut of g with source nodes $\{n_1, \ldots, n_s\}$, target node n_t and the weight function w :minimize $Weight(Nodes(g) \setminus Nodes(G), w)$ s.t.

$$InducedSubGraph(g, G) \wedge \forall i \in [1, n] : n_t \notin Reachable(G, n_i)$$

– Prize Collecting Steiner Tree Problem: g is the initial graph, the arc weights and node prices are w_a and w_n: minimize $Weight(G, w_a) + Weight(SN, w_n)$ s.t.

$$SubGraph(G, g) \wedge Tree(G) \wedge SN = Nodes(g) \setminus Nodes(G)$$

– Graph partitioning problem: equicut of a graph g of even order:
minimize $\#\,(Arcs(g) \setminus (Arcs(G_1) \cup Arcs(G_2)))$ subject to:

$$SubGraph(G_1, g) \wedge SubGraph(G_2, g) \wedge Nodes(G_1) \cup Nodes(G_2) =$$
$$Nodes(g) \wedge \#Nodes(G_1) = \#Nodes(G_2) = \frac{1}{2}\#Nodes(g)$$

Section 6 will present the expression of a constrained shortest path finding problem: finding the shortest simple path in a graph given a set of nodes which must be present in the path and a set of pairs of mutually exclusive nodes.

3 Consistency in CP(Graph)

This section covers the propagation rules of the constraints in the kernel CP(Graph) language. We first define mixed consistency for constraints combining graph, finite set and finite domain variables. The constraints of the kernel are mixed consistent and mixed consistency will be applied to other constraints in a later section.

3.1 Mixed Consistency

Given a constraint $C(\boldsymbol{X})$ over the variables $\boldsymbol{X} = X_1, \ldots, X_n$ with domains $\mathcal{D} = D_1 \times \ldots \times D_n$, we first define the set of solutions of the constraint C on the domain $\mathcal{D}$ of its variables.

$$Sol(C, \mathcal{D}) = \{\mathbf{x} \in \mathcal{D} \mid C(\mathbf{x})\}$$

We denote $Sol(C, \mathcal{D})[X_i]$, the projection of this set on the ith component of its tuples. We also note FD for finite domain variables, FS for finite set variables and GD for graph domain variables.

Definition 1. *For a graph domain variable or a finite set variable X_i with domain $D_i = [x_{iL}, x_{iU}]$, we say C is* bound consistent *on X_i with respect to $\mathcal{D}$ iff*

$$x_{iL} = glb\,(Sol(C, \mathcal{D})[X_i])\,,\; x_{iU} = lub\,(Sol(C, \mathcal{D})[X_i])$$

Definition 2. *For a finite domain variable X_i with domain $D_i = \{a_0, a_1, \ldots, a_n\}$, we say C is* arc consistent *on X_i with respect to $\mathcal{D}$ iff*

$$D_i = Sol(C, \mathcal{D})[X_i]$$

Definition 3. *C is* mixed consistent *with respect to $\mathcal{D}$ iff for all $1 \leq i \leq n$*
if X_i is a GD or FS variable, C is bound consistent on X_i with respect to $\mathcal{D}$,
if X_i is a FD variable, C is arc consistent on X_i with respect to $\mathcal{D}$.

3.2 Propagation Rules of the Kernel Constraints

This section covers the consistency and propagation rules of the kernel constraints of CP(Graph). All of the rules have the domains of the variables implicitly defined by $dom(G) = [g_L, g_U]$, with $g_L = (gsn_L, gsa_L)$ and $g_U = (gsn_U, gsa_U)$, $dom(SA) = [sa_L, sa_U]$ and $dom(SN) = [sn_L, sn_U]$.

We consider an $O(1)$ complexity for the inclusion or exclusion of a value in/out of a finite domain or finite set bound and similarly for an arc of a graph domain variable. As we consider the internal constraint of graphs ($gsa_X \subseteq gsn_X \times gsn_X$ where X stands for both U and L), the removal of a node of G can trigger up to d arc removals where d is the maximal degree of g_U. We also consider a propagator knows for which variable and value it is run in case of an update event.

The Arcs Constraint The $Arcs(G, SA)$ constraint propagation rule is unique. Its application leads to bound consistent domains. The new bounds of the variables G and SA are denoted with a prime. Obviously, only the set of arcs of the bounds of G are updated.

$$sa'_L = gsa'_L = sa_L \cup gsa_L$$
$$sa'_U = gsa'_U = sa_U \cap gsa_U$$

The complexity of this rule is $O(1)$ per update as it just suffices to forward update events from one variable to the other.

The Nodes Constraint The propagation rule of the $Nodes(G, SN)$ constraint is similar to the $Arcs$ propagation rule:

$$sn'_L = gsn'_L = sn_L \cup gsn_L$$
$$sn'_U = gsn'_U = sn_U \cap gsn_U$$

This rule also achieves bound consistency and its complexity is $O(d)$ per update where d is the maximal degree of g_U. That is $O(|sa_U| + |sn_U|)$ over a branch of the search tree as each node and arc can only be removed once.

The ArcNode Constraint The $ArcNode(A, N_1, N_2)$ constraint links an arc variable to two node variables. The update of the domains is straightforward:

$$dom'(A) = dom(A) \cap (dom(N_1) \times dom(N_2))$$

$$dom'(N_1) = \{n_1 \in dom(N_1) | \exists n_2 \in dom(N_2), (n_1, n_2) \in dom(A)\}$$
$$dom'(N_2) = \{n_2 \in dom(N_2) | \exists n_1 \in dom(N_1), (n_1, n_2) \in dom(A)\}$$

Once a fixed-point is reached, the domains are arc-consistent. The complexity is similar as the previous one. The removal of a node from a node domain leads to at most d removals of arcs. Here the graph under consideration is the union of the initial least upper bounds of the graph variables in the CSP.

4 Global Constraints

In Section 2.4, we showed how graph constraints can be built by combining kernel constraints. It however appears that dedicated propagators can be more efficient than a combination of propagators of kernel constraints. This amounts to write a so called global constraint, where global refers to operational globality when more pruning is achieved or algorithmic globality when the same level of pruning is achieved [15].

We here focus on the $Path$ and $Reachable$ constraints and sketch global propagators for these constraints. However, other existing global constraints enforcing graph properties or relations between graphs can be integrated with CP(Graph).

4.1 The Reachable Constraint

$Reachable(G, N, SN)$ holds if SN is the set of nodes reachable from N in G. This constraint encodes the transitive closure of the adjacency relation of the graph. It is expressible using kernel constraints but it requires to post a lot of constraints (see section 2.4). If more pruning is to be done (detection of cutnodes, bridges, etc...), then even more propagators have to be posted. On the other hand, an imperative algorithm can handle these problems easily. Computation of connected components, strongly connected components, bridges, etc... can be done with variants of depth first search in linear time. Incremental algorithms have also been designed to handle dynamic graphs [16]. Hence a global propagator is much more efficient for such constraints.

In CP(Graph), N is a node variable. Constraint propagators have been defined for a determined source node $N = n$ [14, 11]. It is however simple to adapt these propagators to an unknown source. A simple schema is to execute these propagators for each of the values of the domain of the source node and perform their filtering for the values on which they all agree. For instance, if for each value in the domain of N, the node n of G is found to be mandatory, then it is indeed mandatory for any value of N. If one of these propagators would do a pruning which is inconsistent with the current domains, then it means the according value of N can be removed from its domain. By applying this generic reasoning to the existing propagators it is possible adapt them.

4.2 The Path Constraint

We introduce the constraint $Path(G, N_1, N_2, w, I)$, the global version of $(Path(G, N_1, N_2) \wedge Weight(G, w, I))$. It holds if G is a path from N_1 to N_2 whose total weight is I according to the weight function w. With such a constraint, it is possible to do cost based filtering. Note that all parameters of this constraint can be variables except the weight function w which must be a constant. In this section, we show how to adapt the work of [7] on cost-based filtering to this constraint in CP(Graph).

The most general mode of usage of the $Path$ constraint is the case with four unbound variables. However it can be directly reduced to a problem with two unbound variables G' and $I : Path(G', n_s, n_e, w', I)$ by introducing a virtual source n_s and sink n_e. These virtual nodes are assigned a null weight in w'. We do not introduce an additional graph domain variable G' to do this filtering. We just pretend to temporarily update the data structure of the least upper bound of G, g_U (the updated g_U is noted g'_U) to add these nodes, arcs and weights. The problem of filtering in this structure is equivalent to the filtering in the following problem:

$$Path(G', n_s, n_e, w', I)$$
$$Nodes(G') = Nodes(G) \cup \{n_s, n_e\}$$
$$Arcs(G') = Arcs(G) \cup \{(n_s, n) | n \in dom(N_1)\} \cup \{(n, n_e) | n \in dom(N_2)\}$$
$$w'(x) = \begin{cases} w(x) & \text{if } x \in G, \\ 0 & \text{if } x \in G'/G. \end{cases}$$

The domains of N_1 and N_2 are easily filtered: all filtering made on the arcs incident to n_s and n_e is reflected on the domains of N_1 and N_2.

$$dom(N_1) := OutNeighbors(g'_U, n_s), \;\; dom(N_2) := InNeighbors(g'_U, n_e)$$

By introducing these virtual nodes, we can also move all the weights to the arcs (average of the end-nodes weights) while preserving the total weights of all the paths from n_s to n_e. It allows to apply an algorithm for the cost-based filtering of the domain of G using the lub of the domain of I. This consists in a shorter path constraint presented for arc-weighted graphs in [7]. A lower bound of I is also obtained as a side product of this algorithm. The complexity of this filtering is $O(|gsa|.|gsn|log|gsn|)$ over a branch of the search tree.

Using the lub of I to filter the domain of G is possible by applying the longest path propagator for directed acyclic graphs of [17] on the component graph (the graph where the strongly-connected components of the original graph are condensed to a single node).

5 Undirected Graphs

CP(Graph) also supports undirected graphs through an undirected view of a directed graph variable. Undirected graphs are handled like directed graphs by the framework, only the constraints differ. Some constraints have an undirected semantic while others have a directed graph semantic. Some graph properties like being a single connected component are indeed defined for undirected graphs. As a undirected graph is a special case of directed graph, properties defined for directed graphs can be applied as constraints on undirected graphs. On the other hand, a constraint with an undirected graph semantic can be applied to a directed graph as it just operates on the undirected view of the graph (regardless of the orientation of the arc). This view is handled by an additional constraint handling the unordered couples of nodes for the undirected arcs:

$UndirArcNode(A, N_1, N_2)$ A is an undirected arc between node N_1 and N_2. This relation holds iff $ArcNode(A, N_1, N_2)$ or $ArcNode(A, N_2, N_1)$ holds.

6 Experiments

This section describes the prototype of CP(Graph) and the constrained path finding experiments we did to show its practicality. Then, it discusses the results of the experiments.

6.1 Prototype of CP(Graph) implemented in Oz/Mozart

We implemented a prototype of CP(Graph) over the Oz/Mozart[18] constraint programming framework. In this prototype, graph domain variables are implemented using set variables. One set is used for the nodes of the graph and one for the arcs of the graph. This prototypes allows to state constraint satisfaction problems as well as optimization

problems. The constraint propagators are implemented as combinations of kernel constraints or as dedicated global propagators. We implemented the kernel constraints, a reachability propagator and the path propagator of [8] using the Oz language. The other constraints are implemented by combining these constraints with finite set constraints. As Mozart does not support finite sets of couples of integers, we use an integer encoding of arcs. $ArcNode$ provides an accessor for the end nodes of an arc and for the arc number of a couple of nodes through the use of hash tables.

6.2 Biochemical network analyses

We used CP(Graph) to model and solve a problem for biochemical network analysis. Biochemical network analysis consists in assessing the properties of the biochemical networks. These networks are composed of all the genes, molecules, reactions and controls (e.g. catalysis of a reaction) and their interactions, which may occur in one or several organisms. They can be modeled by a labeled simple digraph [1, 11].

We focus here on metabolic networks, that is biochemical networks describing reactions and their substrates and products. A pathway is a specific subgraph of a metabolic network which has a known function in the metabolism. Such pathways were identified experimentally and described in the molecular biology literature.

One type of analysis of biochemical networks consists in trying to computationally find pathways in the metabolic network. An application of this type of analysis lies in the explanation of DNA chip experiments: in a given context, the cell will activate a subset of its possible reactions. A DNA chip enables to list the activated reactions. Given such a set of reactions actually used by the cell, biologists would like to know which pathways were at work in the cell. Our approach is to first develop a CSP able to recover known pathways and then use it to discover new pathways as a result of a DNA chip experiment.

Constrained Shortest Path Finding As about half of the known pathways are simple paths [19], one type of experiment consists in trying to find these pathways by using constrained path finding in a directed graph (knowing a few nodes of the path). In [20], several computational path finding experiments were described. The best experiment consisted in doing point-to-point shortest path finding in a network where each node has a weight proportional to its degree.

Our experiment consists in redoing the former experiment with an additional constraint of inclusion of some intermediate reactions and mutual exclusion for certain pairs of reactions. These pairs are reverse reactions (the reaction from substrates to products and the one from products to substrates). Most of the time, these reactions are observed in a single direction in each species. Hence we wish to exclude paths containing both in our experiment. These two addtional constraints could not be easily integrated in the previous dedicated algorithm [20]. In CP(Graph) it just consists in posting a few additional constraints. If $n_1, ..., n_m$ are the included reactions and $(r_{i1}, r_{i2}), 0 < i \leq t$ the mutually exclusive nodes, the program looks like: miminize $Weight(G, w)$ s.t.

$$SubGraph(G, g) \wedge Path(G, n_1, n_m) \wedge \forall 0 < i \leq m : n_i \in Nodes(G) \wedge$$
$$\forall i \in [0, t] : (r_{i1} \notin Nodes(G) \vee r_{i2} \notin Nodes(G))$$

In our experimental setting we first extract a subgraph of the original metabolic bipartite digraph by incrementally growing a fringe starting by the included nodes. Then, given a subset of the reactions of a reference pathway, we try to find the shortest constrained path in that subgraph. The first process of extraction of a subgraph of interest is done for efficiency reasons as the original graph is too big to be handled by the CSP (it contains around 16.000 nodes). The results are presented in Table 2, it shows the increase of running time, memory usage and size of the search tree with respect to the size of the graph for the extraction of three illustrative linear pathways shown in [20]. All reactions are mandatory in the first experiment. The results of another experiment where one reaction out of two successive reactions in the given pathway is included in the set of mandatory nodes, is presented in Table 3. The running time increases greatly with

Glycolysis (m=8)					Heme (m=8)					Lysine (m=9)				
Size	t	Time	Nodes	Mem	Size	t	Time	Nodes	Mem	Size	t	Time	Nodes	Mem
50	12	0.2	20	2097	50	22	0.2	32	2097	50	18	0.2	38	2097
100	28	2.5	224	2097	100	36	0.3	22	2097	100	40	4.7	652	2097
150	48	41.7	1848	4194	150	62	1.0	28	2097	150	56	264.3	12524	15204
200	80	55.0	1172	5242	200	88	398.8	7988	18874	200	70	-	-	-
250	84	127.6	4496	8912	250	118	173.3	2126	9961	250	96	-	-	-
300	118	2174.4	16982	60817	300	146	1520.2	21756	72876	300	96	-	-	-

Table 2: Comparison of the running time $[s]$, number of nodes in the search tree and memory usage $[kb]$, for the 3 pathways and for increasing original graph sizes. m is the number of node inclusion constraints and t the number of mutual exclusion constraints.

Glycolysis (m=5)					Heme (m=5)					Lysine (m=5)				
Size	t	Time	Nodes	Mem	Size	t	Time	Nodes	Mem	Size	t	Time	Nodes	Mem
50	12	0.2	22	2097	50	22	0.3	44	2097	50	18	0.1	16	2097
100	28	2.5	230	2097	100	36	0.9	78	2097	100	40	13.3	1292	3145
150	48	79.3	5538	6815	150	62	7.3	144	3145	150	56	260.4	8642	14155
200	80	39.9	1198	5767	200	88	57.3	950	5242	200	70	4330.5	74550	192937
250	84	323.6	5428	14680	250	118	36.0	350	8388	250	96	-	-	-
300	118	10470.8	94988	296747	300	146	-	-	-	300	96	-	-	-

Table 3: Same experiment as in Table2, but with one reaction node included every two ($m = 5$ instead of 8 or 9).

the size of the graphs. The program can however be stated in a few lines and first results obtained the same day the experiment is designed. The limitation on the input graph size does not guarantee to get the optimal shortest path in the original graph. This should however not be a major problem as biologists are most of the time interested in a particular portion of the metabolic graph. The rapidity of expression and resolution of such a NP(Hard) [7] problem reduces this size limitation.

Future work focuses on the limitation of running time explosion with graph size which can be obsverved in the results tables. Current results are better than those obtained with our other implementations of graph variables [21, 11]. We wish to design more efficient heuristics for labelling (a first-fail strategy from [12] has been used). Cost-based filtering will be implemented and used in order to limit the size of the graph according to an upper bound of the cost of the path. A second aspect of our future work consists in finding which constraints are needed to recover known pathways as it was shown in [20] that non-constrained shortest paths are not able to recover all of them.

7 Conclusion

This paper introduces the CP(Graph) computation domain with graph domain variables in order to state and solve subgraph extraction problems. CP(Graph) provides finites domains and finite sets of nodes and arcs along with the graph domain variables as this is more expressive than nodes or arcs alone.

The kernel constraints, a minimal set of constraints in order to build other graph constraints and problems, are introduced with their achieved consistency and complexity. Graph constraints are built using the kernel constraints and we sketch a global propagator for some of them. CP(Graph) provides a framework for the integration of existing and new global constraint on graphs. We describe a path constraint based on [7, 8], with domain variables for the source and sink. Finally we showed that CP(Graph) can be used to simply express and solve a problem in biochemical network analysis requiring up to now a dedicated and sophisticated algorithm.

In the proposed CP(Graph) prototype, graph domain variables are represented by a finite set of nodes and a finite set of arcs. A dedicated data-structure for graph domain variables will be designed and compared to the current set implementation. It will most probably consist in a graph data-structure for both bounds of the graph domain variable. The integration of CP(Graph) in an existing constraint solver will then be pursued by integrating graph variables as native variables of the system. We are working on a Gecode [22] implementation.

The application of CP(Graph) to bioinformatics problems will be pursued. This should result in the need for more memory effective graph domain variables and better branching strategies as big graphs (*e.g.* 16000 nodes, 45000 arcs) are under consideration in this field. CP(Graph) will also be compared to other implementations of combinatorial graph problems using constraint programming.

CP(Graph) allows to state problems about multiple graphs. An important problem among those is the graph isomorphism problem. We are adapting the global constraints of (mono/iso)-morphism and subgraph (mono/iso)-isomorphism of two graph domain variables from the techniques developed in [23].

References

1. Deville, Y., Gilbert, D., van Helden, J., Wodak, S.: An overview of data models for the analysis of biochemical networks. Briefings in Bioinformatics **4(3)** (2003) 246–259

2. Gervet, C.: Interval propagation to reason about sets: Definition and implementation of a practical language. CONSTRAINTS Journal **1(3)** (1997) 191–244
3. Laurière, J.: A Language and a Program for Stating and Solving Combinatorial Problems. Artificial Intelligence **10** (1978) 29–127
4. Gervet, C.: New structures of symbolic constraint objects: sets and graphs. In: Third Workshop on Constraint Logic Programming (WCLP'93), Marseille (1993)
5. Lepape, C., Perron, L., Regin, J.C., Shaw, P.: A robust and parallel solving of a network design problem. In: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming. Volume LNCS 2470. (2002) 633–648
6. Beldiceanu, N.: Global constraints as graph properties on structured network of elementary constraints of the same type. Technical Report T2000/01, SICS (2000)
7. Sellmann, M.: Cost-based filtering for shorter path constraints. In: Proceedings of the 9th International Conference on Principles and Practise of Constraint Programming (CP). Volume LNCS 2833., Springer-Verlag (2003) 694–708
8. Cambazard, H., Bourreau, E.: Conception d'une contrainte globale de chemin. In: 10e Journ. nat. sur la résolution pratique de problèmes NP-complets (JNPC'04). (2004) 107–121
9. Beldiceanu, N., Flener, P., Lorca, X.: The tree constraint. In Bartak, R., Milano, M., eds.: Proceedings of CP-AI-OR'05. Volume LNCS 3524., Springer-Verlag (2005)
10. Courcelle, B.: On the expression of graph properties in some fragments of monadic second-order logic. In: Descriptive complexity and finite models, Providence, AMS (1997) 38–62
11. Grégoire Dooms, Yves Deville, Pierre Dupont: A mozart implementation of cp(bionet). In Van Roy, P., ed.: Multiparadigm Programming in Mozart/Oz. Number LNCS 3389, Springer-Verlag (2004) 237–250
12. Pesant, G., Gendreau, M., Potvin, J., Rousseau, J.: An exact constraint logic programming algorithm for the travelling salesman with time windows. Transp. Science **32** (1996) 12–29
13. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Inf. Comput. **85** (1990) 12–75
14. Quesada, L., Roy, P.V., Deville, Y.: The reachability propagator. Research Report 2005-07, (UCL/INGI)
15. Bessière, C., Van Hentenryck, P.: To be or not to be ... a global constraint. In: Proceedings of the 9th International Conference on Principles and Practise of Constraint Programming (CP). Volume LNCS 2833., Springer-Verlag (2003) 789–794
16. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. Journal ACM **48(4)** (2001) 723–760
17. Michel, L., Van Hentenryck, P.: Maintaining longest paths incrementally. In: Proceedings of the International Conference on Constraint Programming (CP-2003), (Springer-Verlag)
18. Mozart Consortium: The mozart programming system version 1.2.5 (December 2002) `http://www.mozart-oz.org/`.
19. 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 Research **32** (2004) D443–D448
20. Croes, D.: Recherche de chemins dans le réseau métabolique et mesure de la distance métabolique entre enzymes. PhD thesis, ULB, Brussels (2005) (in preparation).
21. Dooms, G., Deville, Y., Dupont, P.: Recherche de chemins contraints dans les réseaux biochimiques. In Mesnard, F., ed.: Programmation en logique avec contraintes, actes des JFPLC 2004, Hermes Science (June 2004) 109–128
22. Gecode: Generic Constraint Development (2005) `http://www.gecode.org/`.
23. Zampelli, S., Deville, Y., Dupont, P.: Approximate constrained subgraph matching. In: Proceedings of the 11th International Conference on Principles and Practise of Constraint Programming (CP), Springer-Verlag (2005)