CP(Graph+Map) for Approximate Graph Matching

Yves Deville, Gregoire Dooms, Stéphane Zampelli, Pierre Dupont

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

Abstract. Graph pattern matching is a central application in many fields. In various areas, the structure of the pattern can only be approximated and exact matching is then too accurate. We focus here on approximations declared by the user within the pattern, stating which part could be discarded (optinal nodes and arcs), and also allowing maching problems between monomorphism and isomorphism through the definition of forbidden arcs.
In this paper, we show how the integration of two new domains of computation over countable structures, *graphs* and *maps*, can be used for modeling and solving approximate graph matching as well as many other matching problems. To achieve this, we introduce map variables where the domain and range can de declared as finite set variables. We describe how such extended map variables can be realized on top of finite domain and finite set variables. On top of CP(Graph+Map), we propose a monomorphims constraint suitable for various matching problems. Finally, global constraints, enhancing the pruning of the monomorphism constraint and of the different matching problems are proposed.

1 Introduction

Graph pattern matching is a central application in many fields [1]. Many different types of algorithms have been proposed, ranging from general methods to specific algorithms for particular types of graphs. In constraint programming, several authors [2, 3] have shown that graph matching can be formulated as a CSP problem, and argued that constraint programming could be a powerful tool to handle its combinatorial complexity.

In many areas, the structure of the pattern can only be approximated and exact matching is then far too accurate. Approximate matching is a possible solution, and can be handled in several ways. In a first approach, the matching algorithm may allow part of the pattern to mismatch the target graph (e.g. [4–6]). The matching problem can then be stated in a probabilistic framework (see, e.g. [7]). In a second approach, the approximations are declared by the user within the pattern, stating which part could be discarded (see, e.g. [8, 9]). This approach is especially useful in fields, such as bioinformatics, where one faces

a mixture of precise and imprecise knowledge of the pattern structures. In this approach, which will be followed in this paper, the user is able to choose parts of the pattern open to approximation.

Within the CSP framework, a model for graph monomorphism has been proposed by Rudolf [3] and Valiente et al. [2]. Our modeling is based on these works. Sorlin [10] proposed a filtering algorithm based on paths for graph isomorphism and part of our approach can be seen as a generalization of this filtering. A declarative view of matching has also been proposed in [11].

In constraint programming, two new domains of computation over countable structures have been introduced: graphs and maps. In CP(Graph) [12], a new type of domain variables, graph domain variables, and constraints on these variables are proposed. CP(Graph) can then be used to express and solve combinatorial graph problems modeled as constrained subgraph extraction problems. In CP(Map) (e.g. [13, 14]), map variables are proposed, but the domain and range are limited to ground sets. Such a high level object is useful for modeling problems such as warehouse location.

In this paper, we show how approximate graph matching can be modeled and solved, within the CSP framework, on top of CP(Graph+Map).

Contributions The main contributions of this work are the following:

- Introduction of map variables, where the domain and range of the mapping are not limited to ground sets, but can be finite set variables;
- Description of how a CP(Map) extension can be realized on top of finite domain and finite set variables;
- Integration of CP(Graph+Map) for modeling and solving approximate graph matching;
- Definition of a monomorphism constraint suitable for modeling and solving different classes of matching problems: monomorphism and isomorphism, graph and subgraph matching, exact and approximate matching;
- Definition of global constraints for the monomorphism constraint and for the different matching problems.

The first section introduces the CP(Graph) framework. The introduction of map variables in CP is described in Section 2. Approximate graph matching is defined in Section 4, and its modeling within CP(Graph+Map) is handled in Section 5. Section 6 proposes global constraints for a more effective pruning.

2 CP(Graph) [12]

Constants and Variables A (directed) graph $g = (sn, sa)$ is a set of nodes sn, and a set of arcs $sa \subseteq sn \times sn$. An extension to undirected graphs is presented in [12].

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 1. This table

presents the notations for constants and domain variables of each type. It also shows one particular aspect of graphs: the inherent constraint stating that an arc can only be present if both end nodes are present.

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

Table 1. The different variables and constants of CP(Graph) along with their notations. Note only the graph has an inherent constraint.

There exists a partial ordering among graphs, defined by graph inclusion: given $g_1 = (sn_1, sa_1)$ and $g_2 = (sn_2, sa_2)$, $g_1 \subseteq g_2$ (g_1 is a subgraph of 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. 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 $s_U = lub(S)$.

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 suppose it is possible to post a constraint for each value in a set variable S, that is $\forall i \in S : C(i)$. 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 $C(i)$. While the former filters more, the latter uses less memory.

CP(Graph) is built over the finite set computation domain [15, 16, 13]. 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 [17]. The usage of graphs as structures of symbolic constraint objects was proposed in 1993 by Gervet [15]. In that work, a graph domain is modeled as an endomorphic relation domain. In 2002, Lepape et al. defined path variables [18] which were used to solve constrained path finding problems in a network design context.

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 [19]. 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.

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 some constraints related to constrained subgraph extraction and the matching 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

$$Subgraph(G_1, G_2) \equiv Nodes(G_1) \subseteq Nodes(G_2) \wedge Arcs(G_1) \subseteq Arcs(G_2)$$

which gives bound consistent pruning as $\subseteq$ is also bound consistent.

$InNeighbors(G, N, SN)$ 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.

$$InNeighbors(G, N, SN) \equiv SN \subseteq Nodes(G) \wedge \forall (n_1, n_2) \in Arcs(G) : n_2 = N \Leftrightarrow n_1 \in SN$$

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

$InducedSubGraph(G_1, G_2)$ holds if G_1 is an induced subgraph of G_2.

$$InducedSubGraph(G_1, G_2) \equiv SubGraph(G_1, G_2) \wedge SN = Nodes(G_2) \setminus Nodes(G_1) \wedge \forall (n1, n2) \in Arcs(G_1) : (n_1 \in SN \vee n_2 \in SN) \text{ XOR } (n1, n2) \in Arcs(G2)$$

The graph constraints can be combined to model numerous NP(Hard) problems as illustrated in [12]. This article also presents consistency and propagation rules of the kernel constraints, as well as some global constraints and a propotype implementation of CP(Graph).

3 CP(Map)

Map variables were first introduced in CP in [13] where Gervet defined relation variables. However, the domain and the range of the relations were limited to ground finite sets. Map variables were also introduced as high level type constructors, simplifying the modeling of combinatorial optimization problems. This was first defined in [14] as a relation or map variable M from set v into a set w, where supersets of v and w must be known. Such map variables are then compiled into OPL. This idea is developed in [20], but the domain and range of a map variable are limited to ground sets. Relation and map variables are also described in [21] as a useful abstraction in constraint modeling. Rules are proposed for refining constraints on these complex variables into constraints on finite domain and finite set variables. Map variables were also introduced in modeling languages such as ALICE [17], REFINE [22] and NP-SPEC [23]. As far as we know, map variables are not yet introduced directly in a CP language. One challenge is then to extend current CP languages to allow map variables as well as constraints on these variables.

We here sketch how a CP(Map) extension can be realized on top of finite domain and finite set variables. For ease of presentation, we do not consider relation variables and focus on map variables.

3.1 Map variables and kernel constraints

A map variable is declared as $MapVar(M, S, T)$, where M is the map variable and S, T are finite set variables. The domain of M is all the *total surjective* functions from s to t, where s, t are in the domain of S, T. We call S the *source set* of M, and T the *target set* of M. When M is instantiated (when its domain is a singleton), the source set and the target set of M are ground sets corresponding to the domain and the range of the mapping. An order relation on maps can be defined as follows. Given $MapVar(M_1, S_1, T_1)$ and $MapVar(M_2, S_2, T_2)$, we have $M_1 \subseteq M_2$ iff $S_1 \subseteq S_2 \wedge T_1 \subseteq T_2 \wedge \forall s \in S_1 : M_1(s) = M_2(s)$. We then obtain a meet semi lattice, that is a semi lattice in which there exists a glb between any two elements.

Example Let M be a map variable declared in $MapVar(M, S, T)$, with $dom(S) = [\{8\}, \{4, 6, 8\}]$ and $dom(T) = [\{\}, \{1, 2, 4\}]$. A possible instance of M is $\{4 \rightarrow 1, 8 \rightarrow 4\}$. On this instance, $S = \{4, 8\}$, and $T = \{1, 4\}$. Another instance is $M = \{4 \rightarrow 1, 8 \rightarrow 1\}$, $S = \{4, 8\}$, and $T = \{1\}$.

The kernel constraint on a map variable M is the constraint $Map(M, V_0, V_1)$, where V_0 and V_1 are finite domain variables. The constraint holds when $V_0 \in$

$S \wedge V_1 \in T \wedge M(V_0) = V_1$, where S and T are the source and target sets of the mapping M.

Map variables can be used for defining various kinds of mappings, such as :

- Total function from S to T: $T' \subseteq T \wedge MapVar(M, S, T')$
- Partial function from S to T: $T' \subseteq T \wedge S' \subseteq S \wedge MapVar(M, S', T')$
- Bijective function from S to T: $MapVar(M, S, T) \wedge |S| = |T| \wedge \forall i, j \in S : M(i) \neq M(j)$. The last conjunct is redundant but achieves better pruning.

Other forms of mapping can be derived by combining the above mappings.

3.2 Implementing CP(Map)

Map variables can be implemented by using a structure to model the relation. The relation can be modeled in a way similar to [13] as an array of domain variables. In that work, the source set is a ground set and an array of finite set variables is used to represent the relation. These variables model the set of values of the target set which are in relation with each of the values in the source set. There are as much values in the source set as variables in the array.

The difference between our Map variables and the relations presented in that work are the following. As maps are functions and not general relations, the domain variables stored in this indexed array are not finite sets but finite domain variables. The source set and target set are modeled by two set variables S_s and S_t. These variables are constrained to be coherent with the values of the image variables.

When a Map variable M is declared by $MapVar(M, S, T)$ with $lub(S) = \{v_1, ..., v_n\}$, an array of n FD variables $Image_i, 0 \leq i < n$ is allocated. We also allocate a dictionary data structure *index* used to store the index in the array of each value of $lub(S_s)$ (i.e. $index(v_j) = j$).

The initial domain of each variable in the *Image* array is $lub(T) \cup \{e_i\}$ where e_i is a special value used to denote the absence of image for this index. Obviously, $e(i)$ is chosen such that it is different from all values in $lub(T)$, and different from the e_j values of the other indexes. We just need an $O(1)$ operation testing if a value is a special value denoting the absence of image. Such a representation allows a simple and efficient implementation of an injective/bijective condition on M through a global *alldiff* constraint on the $Image_i$ variables. An example is provided in Figure 1.

A global constraint implementing the following constraints is also posted:

- $\forall 0 \leq i \leq |lub(S)| : Image_i \neq e_i \Leftrightarrow index^{-1}(i) \in S \Leftrightarrow Image_i \in T$
- $\forall j \in lub(T) : Occurs(Image, j) = 0 \Rightarrow j \notin T$

where $Occurs(Image, j)$ denotes the number of occurences of j in the set union of $Image_i$'s.

The constraint $Map(M, V_0, V_1)$ is translated to $Element(index(V_0), Image, V_1)$ where $index(V_0)$ is a finite domain obtained by taking the index of each value of the domain of V_0 using the *index* dictionary.

The Map variable is instantiated (its domain is a singleton) once all variables in *Image* are instantiated.

Image	{1,2,e}	{e}	{1,4}
index	1 ↑ 4	2 ↑ 6	3 ↑ 8

dom(T)=[{},{1,2,4}]

dom(S)=[{8},{4,8}]

Fig. 1. Implementation of $MapVar(M, S, T)$ (with initial domain $dom(S) = [\{8\}, \{4, 6, 8\}]$ and $dom(T) = [\{\}, \{1, 2, 4\}]$), assuming (other) constraints already achieved some pruning.

4 Approximate graph matching [9]

In this section, we define the problem of subgraph matching and describe how graph matching can be approximated through the definition of optional nodes and forbidden arcs. The following definitions apply for directed as well as undirected graphs.

A **subgraph isomorphism** between a pattern graph $G_p = (N_p, E_p)$ and a target graph $G_t = (N_t, E_t)$ is a total injective function $f : N_p \rightarrow N_t$ respecting $(u, v) \in E_p \Leftrightarrow (f(u), f(v)) \in E_t$. The graph G_p is isomorphic to G_t through function f.

A **subgraph monomorphism** between G_p and G_t is a total injective function $f : N_p \rightarrow N_t$ respecting $(u, v) \in E_p \Rightarrow (f(u), f(v)) \in E_t$. The graph G_p is monomorphic to G_t through function f.

Graph isomorphism and monomorphism can also be defined as variant of the above definitions where the function f is bijective. Alternatively, subgraph isomorphism and monomorphism can also be defined as graph isomorphism and monomorphism with a subgraph of the target graph.

A **subgraph matching** is either a subgraph isomorphism or a subgraph monomorphism. Subgraph isomorphism and monomorphism are known to be NP-complete.

A useful extension of subgraph matching is approximate subgraph matching, where the pattern graph and the found subgraph in the target graph may differ with respect to their structure. The following definitions are from [9].

Optional nodes In our framework, the approximation is *declared* upon the pattern graph. Some *nodes* are declared *optional*, i.e. nodes that may not be in the matching. Specifying optional arcs in a monomorphism problem is useless as it is equivalent to omitting the arc in the pattern. The status of the arcs depends on the optional state of their endpoints. An arc having an optional node as one of its endpoints is optional. An optional arc is not considered in the matching if one of its endpoints is not part of the matching. Otherwise, the arc must also belong to the matching.

Forbidden arcs *Arcs* may also be declared as *forbidden* between their two endpoints (u, v), meaning that if u and v are in the domain of f, then $(f(u), f(v))$

must not exist in the target graph. A pattern graph with all its complementary arcs declared as forbidden induces a subgraph isomorphism instead of a subgraph monomorphism.

A pattern graph with optional nodes and forbidden arcs forms an *approximate pattern graph.*

Definition 1 *An* ***approximate pattern graph*** *is a tuple* (N_p, O_p, E_p, F_p) *where* (N_p, E_p) *is a graph,* $O_p \subseteq N_p$ *is the set of optional nodes and* $F_p \subseteq N_p \times N_p$ *is the set of forbidden arcs, with* $E_p \cap F_p = \emptyset$.

The corresponding matching is called an *approximate subgraph matching.*

Definition 2 *An* ***approximate subgraph matching*** *between an approximate pattern graph* $G_p = (N_p, O_p, E_p, F_p)$ *and a target graph* $G_t = (N_t, E_t)$ *is a* partial *function* $f : N_p \rightarrow N_t$ *such that:*

1. $N_p \setminus O_p \subseteq dom(f)$
2. $\forall\, i, j \in dom(f) : i \neq j \Rightarrow f(i) \neq f(j)$
3. $\forall\, i, j \in dom(f) : (i, j) \in E_p \Rightarrow (f(i), f(j)) \in E_t$
4. $\forall\, i, j \in dom(f) : (i, j) \in F_p \Rightarrow (f(i), f(j)) \notin E_t$

The notation $dom(f)$ represents the domain of f. Elements of $dom(f)$ are called the selected nodes of the matching. This means that $dom(f)$ can be represented by a finite set variable. Its greatest lower bound consists of all selected nodes, and its least upper bound consists of selected nodes and nodes that could be selected.

Condition 1 requires mandatory nodes to be in the matching. Condition 2 is the injective condition, also present in the exact case. Condition 3 enforces that an arc between two selected endpoints must always be present in the target. Condition 4 forbids the presence of an arc in the matching between node $(f(u), f(v))$ if the arc (u, v) was declared forbidden and u, v are in the matching. According to this definition, if $F_p = \emptyset$ the matching is a subgraph monomorphism, and if $F_p = N_p \times N_p \setminus E_p$, the matching is an isomorphism.

Condition 3 has an important impact on the set of possible matchings, as shown in Figure 2. In this figure, mandatory nodes are represented as filled nodes, and optional nodes are represented as empty nodes. Mandatory arcs are represented with plain line, and optional arcs are represented with dashed lines. Forbidden arcs are represented with a plain line crossed. Intuitively, one could think that arc $(5, 6)$ in the pattern could be discarded, while node 6 could be selected together with arc $(4, 6)$. In fact, because of condition 3, matching of node 6 would require the arc $(5, 6)$ to be present in the target. Only two subgraphs match this pattern as shown on the right side of Figure 2. The nodes and arcs not selected in the target graph are grey.

5 Modeling approximate graph matching

In this section, we show how CP(Graph+Map) can be used for modeling and solving approximate graph matching as well as many other matching problems. This section is thus central to this paper and contitutes our main contribution.

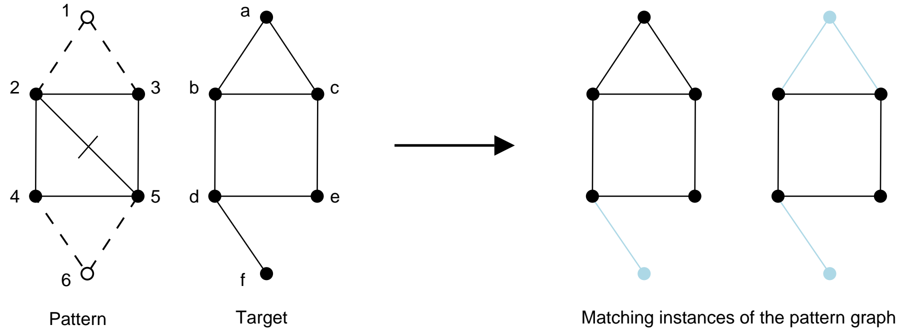

Fig. 2. Example of approximate matching.

The problem of graph matching can be stated along three different dimensions:

- monomorphism versus isomorphism;
- graph versus subgraph matching;
- exact versus approximate matching

This leads to 8 different classes of problems. All these problems can be modeled and solved through a single monomorphism constraint on graph domain variables and a map variable.

5.1 The monomorphism constraint

The constraint $Mono(P, G, M)$ holds if P is monomorphic to G through M, where P, G are graph domain variables and M is a map variable with source set $Nodes(P)$ and target set $Nodes(G)$. A $Mono(P, G, M)$ constraint thus implies an implicit $MapVar(M, Nodes(P), Nodes(G))$ with the additional constraint that M is bijective.

The $Mono(P, G, M)$ constraint can be defined as follows :

$$\begin{aligned} Mono(P, G, M) \equiv\ & MapVar(M, Nodes(P), Nodes(G)) \\ & \wedge |Nodes(P)| = |Nodes(G)| \\ & \wedge \forall i, j \in Nodes(P) : M(i) \neq M(j) \\ & \wedge (i, j) \in Arcs(P) \Rightarrow (M(i), M(j)) \in Arcs(G) \end{aligned}$$

The first three conjuncts (bijective mapping) can be implemented efficiently as described in Section 3.2. We now show how this constraint can be used to solve the different classes of problems.

5.2 Graph and subgraph monomorphism

Let p be a pattern graph and g be a target graph. The graphs p and g are thus ground objects in CP(Graph). Graph monomorphism can easily be modeled as

$$Mono(p, g, M)$$

In a subgraph monomorphism problem, there should exist a monomorphism between p and a subgraph of g. Hence the constraints:

$$Subgraph(G, g) \wedge Mono(p, G, M)$$

The graph G will thus be the matched subgraph of g. The range of M will be the nodes of the matched subgraph of g. Notice that for subgraph matching, it is essential to allow map variables with a finite set variable as target set.

5.3 Graph and subgraph isomorphism

Graph isomorphism can be modeled by two monomorphisms: one between the graphs, and a second between the complementary graphs.

We first introduce a complementary graph constraint $CompGraph(G, Gc)$ which holds if $Nodes(G) = Nodes(Gc) = N$ and $Arcs(Gc) = (N \times N) \setminus Arcs(G)$.

From the definition of graph isomorphism, an isomorphism is a monomorphism with the additional constraint that if an arc does not exist between two pattern nodes, then an arc should not exist through the mapping. This additional constraint states that the mapping should also be a monomorphism between the complementary graphs. Hence the following constraint:

$$\begin{aligned} & Mono(p, g, M) \\ & \qquad \wedge CompGraph(p, Pc) \wedge CompGraph(g, Gc) \wedge Mono(Pc, Gc, M) \end{aligned}$$

Notice that the second line of the constraint could be replaced by $|Arcs(p)| = |Arcs(g)|$, leading to a simpler constraint, but achieving less pruning. This also holds for the next constraint.

Subgraph isomorphism can then be derived easily, following the same idea than for monomorphism.

$$\begin{aligned} & Subgraph(G, g) \wedge Mono(p, G, M) \\ & \qquad \wedge CompGraph(p, Pc) \wedge CompGraph(G, Gc) \wedge Mono(Pc, Gc, M) \end{aligned}$$

5.4 Introducing optional nodes

Let us first introduce optional nodes in the pattern graph. Let p be the pattern graph with optional nodes, and p_{man} be the subgraph of p induced by the mandatory nodes of p. Approximate graph monomorphism then amounts

to find a graph p_{sol} between p_{man} and p which is monomorphic to the target graph. However, between p_{man} and p, only the subgraphs induced by p should be considered. When two optional nodes are selected in the matching, if there is an arc between these nodes in pattern graph p, this arc must be considered in the matching, according to our definition of optional nodes. We then obtain the following constraint:

$$P \in [p_{man}, p] \wedge InducedSubGraph(P, p) \wedge Mono(P, g, M)$$

Notice that for optional nodes, it is essential to allow map variables with a finite set variable as source set. This easily extends to subgraph monomorphism with optional nodes:

$$Subgraph(G, g) \wedge P \in [p_{man}, p] \wedge InducedSubGraph(P, p) \wedge Mono(P, G, M)$$

The domain of the mapping M will define the selected nodes in the pattern and the range of M will define the selected nodes in the target graph g.

Although not described here, optional nodes can also be handled in (sub)graph isomorphism.

5.5 Approximate matching

We now consider the general problem of approximate subgraph matching as defined in the previous section. Given an approximate pattern graph (N_p, O_p, E_p, F_p) where (N_p, E_p) is a graph, $O_p \subseteq N_p$ is the set of optional nodes, and $F_p \subseteq N_p \times N_p$ is the set of forbidden arcs, and a target graph (N_t, E_t), we define the following CP(graph) constants :

- p: the pattern graph (N_p, E_p),
- p_{man}: the subgraph of p induced by the mandatory nodes $N_p \setminus O_p$ of p,
- g: the target graph (N_t, E_t),
- p_{forb} : the graph (N_p, F_p) of the forbidden arcs.

The modeling of approximate matching is a combination of subgraph monomorphism with optional nodes, and subgraph isomorphism. But here the complementary matching should hold between the complementary target subgraph and the subgraph p_{forb} composed of the selected nodes in the pattern and all the forbidden arcs between these nodes.

$$\begin{aligned} &Subgraph(G, g) \wedge P \in [p_{man}, p] \wedge InducedSubGraph(P, p) \wedge Mono(P, G, M) \\ &\quad \wedge Nodes(Pc) = Nodes(P) \wedge InducedSubGraph(Pc, p_{forb}) \\ &\qquad \wedge CompGraph(G, Gc) \wedge Mono(Pc, Gc, M) \end{aligned}$$

6 Global constraints

The implementation of the $Mono(P, G, M)$ constraint described in the previous section is not very efficient. More particularly, a classical arc-consistency algorithm on the constraint

$$(i, j) \in Arcs(P) \Rightarrow (M(i), M(j)) \in Arcs(G) \tag{1}$$

would cost $O(ED^2)$ amortized time [2], where $N = |Nodes(P)|$, $E = |Arcs(P)|$, $D = |Nodes(G)|$ and d is the average degree of the target graph G. It is therefore important to design a more efficient global constraint for this constraint. Global constraint should also be developed for the different matching problems.

The proposed global constraint for (1) is based on [2, 9], but generalized in the context of CP(Graph+Map). It is defined as follows for undirected graphs :

$$\begin{aligned} &\forall i \in Nodes(glb(P))\ \forall a \in Nodes(lub(G)) \\ &\qquad |Dom(M(i)) \cap Neighb(lub(G), a)| = 0 \\ &\qquad\qquad \Rightarrow a \notin Dom(M(j))\ \forall j \in Neighb(glb(P), i) \end{aligned}$$

where $Dom(X)$ denotes the domain of the variable, and $Neighb(g, a)$ denotes the neighbours of nodes a in graph g.

The proposed propagator keeps track of relations between all the target nodes and the domain $Dom(M(i))$ in a structure $S(i, a) = |Dom(M(i)) \cap Neighb(lub(G), a)|$ representing the number of relations between a target node a and the domain of $M(i)$. Whenever the neighbors of a target node a have no relation with $Dom(M(i))$, that is when $S(i, a) = 0$, node a is pruned from all neighbors of $M(i)$. However, in the above constraint, $lub(G)$ may vary during the computation, preventing a simple update of the $S(i, a)$ data structure. One could however show that the $lub(G)$ term can be replaced by a constant $g = lub(G_0)$, that is the lub of G when the propagator is activated. Algorithm 1 shows an implementation of this global morphism constraint for undirected graph. It has a $O(NDd)$ amortized time complexity, and the structure $S(i, a)$ has O(ND) spatial complexity [2]. The preprocessing to compute $S(i, a)$ costs $O(NDd)$. The global MC constraint is thus *algorithmically* global as it achieves the same consistency than the original conjunction of constraints, but more efficiently [24]. For directed graphs, the global constraint should be split in two constraints, one for the incoming arcs, and one for the outgoing arcs, leading to two data structures for $S(i, a)$, and two loops in Algorithm 1.

Redundant constraint, such as proposed in [2, 9] could also be developed to enhance the pruning.

Specialized global constraint can also be designed for the different matching famillies. For instance, in the approximate matching with optinal nodes, the $InducedSubGraph(P, p)$ can be integrated in the Propagate_MC propagator by simply replacing $Neighb(glb(P), i)$ by $Neighb(p, i)$. For the approximate matching with optional nodes and forbidden arcs, a single propagator could also be designed following the ideas developed in [9].

Algorithm 1: Morphism Constraint

```
Propagate_MC(i,a)
// Element a exits from Dom(M(i))
for b ∈ Neighb(g, a) do
    S(i, b) ← S(i, b) − 1
    if S(i, b) = 0 then
        foreach j ∈ Neighb(glb(P), i) do
            Dom(M(j)) ← Dom(M(j)) \ {b}
```

7 Conclusion

In this paper, we showed how the integration of two new domains of computation over countable structures, *graphs* and *maps*, can be used for modeling and solving approximate graph matching as well as many other matching problems. We already described CP(Graph) in [12]. Maps were already introduced in within CP in [13], as well as in some modeling languages, but were limited to ground sets for the domain and the range of the map variables. We extended CP(Map) with domain and range of the map variable being finite set variables; we also described how such extended map variables can be realized on top of finite domain and finite set variables.

Appromimation matching is based on our work in [9] where approximations are declared by the user within the pattern, stating which part could be discarded (optinal nodes and arcs), and also allowing maching problems between monomorphism and isomorphism through the definition of forbidden arcs.

A monomorphism constraint, defined on graph and map variables has been designed and was shown to be suitable for modeling and solving different classes of matching problems: monomorphism and isomorphism, graph and subgraph matching, exact and approximate matching. Finally, we defined global constraints enhancing the pruning of the monomorphism constraint and for the different matching problems.

CP(Graph+Map) is thus a suitable framework for approximate graph matching. It allows the introduction of additional constraints on the pattern and on the target graph, leading to constrained graph matching. Graph matching can also easily be integrated within various graph analysis problem, such as constrained path finding [12].

An implementation of CP(Graph) without map and without matching is reported in [12]. A CSP implementation of our approach to approximate graph matching, but without explicit graph and map variables is reported in [9]. It is shown that the general framework is competitive with a specialized C++ Ullman (exact) matching algorithm, while also offering approximate matching. The introduction of graph and map variables should not influence the performance.

An implementation of CP(Graph+Map) is now under development. Future work includes a better integration of the graph analysis problems and the match-

ing problems, as well as extending graph matching to other graph comparison problems such as subgraph bisimulation [25].

Acknowledgment This research is supported by the Walloon Region, project BioMaze (WIST 315432). Many thanks to the anonymous reviewers for their helpful comments.

References

1. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. IJPRAI **18** (2004) 265–298
2. Larrosa, J., Valiente, G.: Constraint satisfaction algorithms for graph pattern matching. Mathematical. Structures in Comp. Sci. **12** (2002) 403–422
3. Rudolf, M.: Utilizing constraint satisfaction techniques for efficient graph pattern matching. In Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: TAGT. Volume 1764 of Lecture Notes in Computer Science., Springer (1998) 238–251
4. Wang, J.T.L., Zhang, K., Chirn, G.W.: Algorithms for approximate graph matching. Inf. Sci. Inf. Comput. Sci. **82** (1995) 45–74
5. Messmer, B.T., Bunke, H.: A new algorithm for error-tolerant subgraph isomorphism detection. IEEE Trans. Pattern Anal. Mach. Intell. **20** (1998) 493–504
6. DePiero, F., Krout, D.: An algorithm using length-r paths to approximate subgraph isomorphism. Pattern Recogn. Lett. **24** (2003) 33–46
7. Robles-Kelly, A., Hancock, E.: Graph edit distance from spectral seriation. IEEE Transactions on Pattern Analysis and Machine Intelligence **27-3** (2005) 365–378
8. Giugno, R., Shasha, D.: Graphgrep: A fast and universal method for querying graphs. In: ICPR (2). (2002) 112–115
9. Zampelli, S., Deville, Y., Dupont, P.: Approximate constrained subgraph matching. In: International Conference on Principles and Practice of Constraint Programming. (2005)
10. Sorlin, S., Solnon, C.: A global constraint for graph isomorphism problems. In Régin, J.C., Rueher, M., eds.: CPAIOR. Volume 3011 of Lecture Notes in Computer Science., Springer (2004) 287–302
11. Mamoulis, N., Stergiou, K.: Constraint satisfaction in semi-structured data graphs. In Wallace, M., ed.: CP. Volume 3258 of Lecture Notes in Computer Science., Springer (2004) 393–407
12. Dooms, G., Deville, Y., Dupont, P.: Cp(graph): Introducing a graph computation domain in constraint programming. In: International Conference on Principles and Practice of Constraint Programming. (2005)
13. Gervet, C.: Interval propagation to reason about sets: Definition and implementation of a practical language. Constraints **1** (1997) 191–244
14. Flener, P., Hnich, B., Kiziltan, Z.: Compiling high-level type constructors in constraint programming. In: PADL '01: Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages, London, UK, Springer-Verlag (2001) 229–244
15. Gervet, C.: New structures of symbolic constraint objects: sets and graphs. In: Third Workshop on Constraint Logic Programming (WCLP'93), Marseille (1993)
16. Dovier, A., Rossi, G.: Embedding extensional finite sets in CLP. In: International Logic Programming Symposium. (1993) 540–556

17. Lauriere, J.L.: A language and a program for stating and solving combinatorial problems. Artificial Intelligence **10** (1978) 29–128
18. 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
19. 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
20. Hnich, B.: Function variables for Constraint Programming. PhD thesis, Uppsala University, Department of Information Science (2003)
21. Frisch, A.M., Jefferson, C., Hernandez, B.M., Miguel, I.: The rules of constraint modelling. In: Proceedings of IJCAI 2005. (2005)
22. Smith, D.: Structure and design of global search algorithms. Technical Report Tech. Report KES.U.87.12, Kestrel Institute, Palo Alto, Calif. (1987)
23. Cadoli, M., Palopoli, L., Schaerf, A., Vasile, D.: NP-SPEC: An executable specification language for solving all problems in NP. Lecture Notes in Computer Science **1551** (1999) 16–30
24. 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
25. Dovier, A., Piazza, C.: The subgraph bisimulation problem. IIEEE Transaction on Knowledge and Data Engineering **15** (2003) 1055–1056