Combining Two Structured Domains for Modeling Various Graph Matching Problems

Yves Deville[1], Grégoire Dooms[2], Stéphane Zampelli[1]

[1] Department of Computing Science and Engineering, Université catholique de Louvain, B-1348 Louvain-la-Neuve - Belgium
{Yves.Deville,Stephane.Zampelli}@uclouvain.be

[2] Department of Computer Science, Brown University, Box 1910, Providence, RI 02912, USA
gdooms@cs.brown.edu

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 (optional nodes and forbidden arcs), covering graph/subgraph mono/isomorphism problems. In this paper, we show how the integration of two domains of computation over countable structures, *graphs* and *maps*, can be used for modeling and solving various graph matching problems from the simple graph isomorphism to approximate graph matching. To achieve this, we extend map variables allowing the domain and range to be non-fixed and constrained. We describe how such extended maps are designed then realized on top of finite domain and finite set variables with specific propagators. We show how a single monomorphism constraint is sufficient to model and solve those multiples graph matching problems. Furthermore, our experimental results show that our CP approach is competitive with a state of the art algorithm for subgraph isomorphism.

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 stringent. 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 isomorphism has been proposed by Sorlin et al. [10], and by Rudolf [3] and Valiente et al. [2] for graph monomorphism. Subgraph isomorphism in the context of the SBDD method for symmetry breaking is shortly described in [11]. We also proposed in [9] a CSP model for approximate graph matching, but without graph and map variables. Our propagators for monomorphism are based on these works. A declarative view of matching has also been proposed in [12] in the context of XML queries.

In constraint programming, two domains of computation over countable structures have received recent attention : graphs and maps. In CP(Graph) [13], graph variables, and constraints on these variables are described (see also [14, 15] for similar ideas). CP(Graph) can be used to express and solve combinatorial graph problems modeled as constrained subgraph extraction problems. In [16, 17], function variables are proposed, but the domain and range are limited to ground sets. Such function variables are useful for modeling problems such as warehouse location.

In this paper, we propose an extension to function variables by generalizing them to non-fixed range and domain (source and target set). We call this extension CP(Map) and 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:

– Extension of function variables, where the domain and range of the mapping are not limited to ground sets, but can be finite set variables. Introduction of the *MapVar* and *Map* constraints which allow to use the non-fixed feature of our map variables.
– Demonstration of how a single constraint is able to express a wide range of graph matching problems thanks to three high-level structured variables. In particular, we show how switching a parameter from a fixed graph to a graph interval opens a new spectrum of matching problems. We show how additional constraints imposed on this graph interval enable the expression of hybrid problems such as approximate graph matching. The beauty and originality of this approach resides in that those problems are either new or were always treated separately, illustrating the expressive power and generality of constraint programming.
– Experimental evaluation of our CP approach. We show that this modeling exercise is not only aesthetic but is actually competitive with the current state of the art in subgraph isomorphism (vflib). The genericity of the approach does not hinder the efficiency of the solver. On a standard benchmark set, we show that our approach solves in a given time limit a fourth of the instances which cannot be solved by vflib while only spending between 9% and 22% more time on instances solved by the two competing approaches.

The next section describes the basic idea behind the CP(Graph) framework. CP(Map), our extension to function variables in CP is described in Section 3. Approximate graph matching is defined in Section 4, and its modeling within CP(Graph+Map) is handled in Section 5. Section 6 analyses experimental results, and Section 7 concludes this paper.

2 CP(Graph)

Graphs have been around since the first years of constraint programming. Some problems involving undetermined graphs have been formulated using either binary variables, sets ([14, 15]) or integers (successor variables e.g. in [18, 19]). CP(Graph) [13] unifies those models by recognizing a common structure: Graph variables are variables whose domain ranges over a set of graphs and as with set variables [20, 16], this set of graphs is represented by a graph interval $[\underline{D}(G), \overline{D}(G)]$ where $\underline{D}(G)$, the greatest lower bound (glb) and $\overline{D}(G)$, the least upper bound (lub) are two graphs with $\underline{D}(G)$ a subgraph of $\overline{D}(G)$ (we write $\underline{D}(G) \subseteq \overline{D}(G)$). These two bounds are referred to as the lower and the upper bound. The lower bound $\underline{D}(G)$ is the set of all nodes and arcs which *must* be part of the graph in a solution while the upper bound $\overline{D}(G)$ is the set of all nodes and arcs which could be part of the graph in some solution. The domain of a graph variable with $D(G) = [\underline{D}(G), \overline{D}(G)]$ is the set of graphs g with $\underline{D}(G) \subseteq g \subseteq \overline{D}(G)$. Here, g is used to denote a constant graph and G is used to denote a graph variable. This notation is used throughout this paper: in CSP, lowercase letters denote constants and uppercase letters denote domain variables.

Graph variables can be implemented using a dedicated data-structure or translated into set variables, integer variables or binary variables. For instance, a graph variable G can be modeled as a set of nodes N and a set of arcs E with an additional constraint enforcing the relation $E \subseteq N \times N$. Whatever the graph variable implementation, two basic constraints $Nodes(G, SN)$ and $Arcs(G, SA)$ allow to access respectively the set of nodes and the set of arcs of the graph variable. To simplify the notation the expression $Nodes(G)$ is used to represent a set variable constrained to be equal to the set of nodes of G. A similar notation is used for arcs.

Various constraints have been defined over such graph variables (or their preceding specialized models); see for instance the cycle [18], tree [21], path [22, 23], minimum spanning tree [24] or spanning tree optimization constraint [25]. In the remainder of this article, we only use the two simple constraints $Subgraph(G_1, G_2)$ (also denoted $G_1 \subseteq G_2$) and $InducedSubgraph(G_1, G_2)$ (also denoted $G_1 \subseteq^* G_2$). $G_1 \subseteq G_2$ holds if G_1 is a subgraph of G_2, its propagator enforces that the lower and upper bounds of G_1 are subgraphs of the lower bound and upper bounds of G_2 respectively. The constraint $G_1 \subseteq^* G_2$ states that G_1 is the node-induced subgraph of G_2. It holds if G_1 is a subgraph of G_2 such that for each arc a of G_2 whose end-nodes are in G_1, a is also in G_1.

3 CP(Map)

The value of a map variable is a mapping from a domain set to a range set. The domain of a map variable is thus a set of mappings. Map variables were first introduced in CP in [16] where Gervet defines 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 [17] 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 [26], but the domain and range of a map variable are limited to ground sets. Relation and map variables are also described in [27] 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 [28], REFINE [29] and NP-SPEC [30]. To the best of our knowledge, map variables were 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.

In the remaining of this section, we show how a CP(Map) extension can be realized on top of finite domain and finite set variables.

3.1 The Map domain

We consider the domain of total surjective functions. Given two elements $m_1 : s_1 \rightarrow t_1$ and $m_2 : s_2 \rightarrow t_2$, where s_1, s_2, t_1, t_2 are sets, we have $m_1 \subseteq m_2$ iff $s_1 \subseteq s_2 \wedge t_1 \subseteq t_2 \wedge \forall x \in s_1 : m_1(x) = m_2(x)$. We also have that $m = \mathrm{glb}(m_1, m_2)$ is a map $m : s \rightarrow t$ with $s = \{x \in s_1 \cap s_2 \mid m_1(x) = m_2(x)\}$, $t = \{v \mid \exists x \in s : m_1(x) = v\}$, and $\forall x \in s : m(x) = m_1(x) = m_2(x)$. The lub between two elements m_1, m_2 exists only if $\forall x \in s_1 \cap s_2 : m_1(x) = m_2(x)$. In that case the lub is a map $m : s \rightarrow t$ with $m(x) = m_1(x)$ if $x \in s_1$, and $m(x) = m_2(x)$ if $x \in s_2$, $s = s_1 \cup s_2$, and $t = \{v \mid \exists x \in s : m(x) = v\}$. The domain of total surjective functions is then a meet semi lattice, that is a semi lattice where every pairs of elements has a glb.

3.2 Map variables and the MapVar constraint

A map variable is declared with the constraint $MapVar(M, S, T)$, where M is the map variable and S, T are finite set variables of Cardinal [31]. 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. As usual, the domain of a set variable S is represented by a set interval $[\underline{D}(S), \overline{D}(S)]$, the set of sets s with $\underline{D}(S) \subseteq s \subseteq \overline{D}(S)$.

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 $T = \{1\}$.

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

- Surjective function : $SurjectFct(M, S, T) \equiv MapVar(M, S, T)$.
- Bijective function : $BijectFct(M, S, T) \equiv SurjectFct(M, S, T)$
  $\wedge \forall i, j \in S : i \neq j \Rightarrow M(i) \neq M(j)$.
- Injective function : $InjectFct(M, S, T) \equiv T' \subseteq T \wedge BijectFct(M, S, T')$
- Total function : $TotalFct(M, S, T) \equiv T' \subseteq T \wedge SurjectFct(M, S, T')$
- Partial function : $PartialFct(M, S, T) \equiv S' \subseteq S \wedge TotalFct(M, S', T)$

In order to access individual elements of the map, we define the constraint $Map(M, X, V)$, where X and V are finite domain variables. Given a map variable declared with $MapVar(M, S, T)$, the constraint $Map(M, X, V)$ holds when $X \in S \wedge V \in T \wedge M(X) = V$. We also define the constraint $M1 \subseteq M2$.

3.3 Implementing Map Variables in a Finite Domain Solver

When a map variable M is declared by $MapVar(M, S, T)$, a finite domain (FD) variable M_x is associated to each element x of the upper bound of the source set ($\overline{D}(S)$).

The semantics of these FD variables is simple : M_x represents $M(x)$, the image of x through the function M. Since the source set S can be non-fixed, x might eventually not be in S and its image would not be defined. A special value $\perp$ is used for this purpose. The relationship between the domain of each variable M_x and the set variables S and T can be stated as follows :

– (1) $S = \{x \mid M_x \neq \perp\}$ (M is total)
– (2) $T = \{v \mid \exists x : M_x = v \neq \perp\}$ (M is surjective)

Given $MapVar(M, S, T)$, the domain of M is the set of total surjective functions $m : s \rightarrow t$ with $s \in D(S)$, $t \in D(T)$, $\forall x \in s : m(x) \in D(M_x)$, and $\forall x \notin s : \perp \in D(M_x)$.

As can be seen on Figure 1, these variables are stored in an array and accessed by value x through a dictionary data structure (e.g. hashmap) $index$ used to store the index in the array of each value of $\overline{D}(S)$. The initial domain of each FD variable is $\overline{D}(T) \cup \{\perp\}$.

3.4 Additional Constraints and Propagators

Given two map constraints $MapVar(M1, S1, T1)$ and $MapVar(M2, S2, T2)$ the constraint $M1 \subseteq M2$ is implemented as $S1 \subseteq S2 \wedge T1 \subseteq T2 \wedge \forall x \in S1 : M1_x = M2_x$. The last conjunct can be implemented as a set of propagation rules :

– $x \in \underline{D}(S1) \rightarrow M1_x = M2_x$
– for each $x \in \overline{D}(S1) \setminus \underline{D}(S1) : M1_x \neq M2_x \rightarrow x \notin S1$.

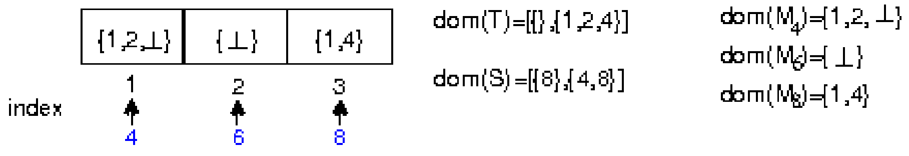

Fig. 1. Implementation of $MapVar(M, S, T)$ (with initial domain $dom(S) = [\{8\}, \{4, 6, 8\}]$ and $dom(T) = [\{\}, \{1, 2, 4\}]$), assuming (other) constraints already achieved some pruning.

The constraint $Map(M, X, V)$ is translated to $Element(index(X), I, V) \wedge X \in S \wedge V \in T$, where S and T are the source and target sets of M, I is the array representing the FD variables M_x, and $index(X)$ is a finite domain obtained by taking the index of each value of the domain of X using the $index$ dictionary.

The implementation of $BijectFct(M, S, T)$ is realized through $MapVar(M, S, T) \wedge AllDiffExceptVal(I, \perp) \wedge |S| = |T|$, where I is the array representing the FD variables M_x, and $AllDiffExceptVal$ holds when all the FD variables in I are different when their value is not $\perp$ [32].

Given $MapVar(M, S, T)$, the propagation between M, S and T is based on their relationship described in the previous section, and is achieved by maintaining the following invariants :

– $\overline{D}(S) = \{x \mid D(M_x) \neq \{\perp\}\}$
– $\underline{D}(S) = \left\{x \in \overline{D}(S) \mid \perp \notin D(M_x)\right\}$
– $\overline{D}(T) = \{v \mid v \neq \perp \wedge \exists x : v \in D(M_x)\}$
– $\underline{D}(T) \supseteq \{v \mid v \neq \perp \wedge \exists x : D(M_x) = \{v\}\}$

The last invariant is not an equality because when a value is known to be in T, it is not always possible to decide which element in I should be assigned to v.

Propagations rules are then easily derived from these invariants (two rules per invariant) :

$$\begin{aligned}
M_x = \perp &\rightarrow x \notin \overline{D}(S) \\
x \notin \overline{D}(S) &\rightarrow M_x = \perp \\
x \in \underline{D}(S) &\rightarrow M_x \neq \perp \\
M_x \neq \perp &\rightarrow x \in \underline{D}(S) \\
v \notin \overline{D}(T) \wedge v \neq \perp &\rightarrow v \notin D(M_x) \\
NbOccur(I, v) = 0 \wedge v \neq \perp &\rightarrow v \notin \overline{D}(T) \\
M_x = v \neq \perp &\rightarrow v \in \underline{D}(T) \\
v \in \underline{D}(T) \wedge NbOccur(I, v) = 1 \wedge v \in D(M_x) &\rightarrow M_x = v
\end{aligned}$$

where $NbOccur(I, v)$ denotes the number of occurrences of v in the domains of the FD variables in I. Each of these propagation rules can be implemented in $O(1)$ (assuming a bit representation of sets). The implementation of propagators also exploits the cardinality information associated with set variables.

3.5 A global constraint based on matching theory

The above propagators do not prune the M_x FD variables (except the $\perp$ value). We show here how flow and matching theory can be used to design a complete filtering algorithm for the $MapVar(M, S, T)$ constraint. The algorithm is similar to that of the GCC and Alldiff constraints but is based on a slightly different notion: the V-matchings (see [33]). In the remainder of this section we show that V-matchings characterize the structure of the $MapVar$ constraint. Note that it also has similarities with the Nvalue, Range and Roots constraints ([34, 35]).

Definition 1. *The* variable-value *graph of a $MapVar(M, S, T)$ constraint is a bipartite graph where the two classes of nodes are the elements of $\overline{D}(S)$ on one side and the elements of $\overline{D}(T)$ plus $\perp$ on the other side. An arc (x, v) is part of the graph iff $v \in D(M_x)$.*

Definition 2. *In a bipartite graph $g = (N_1 \cup N_2, A)$, a* matching *M is a subset of the arcs such that no two arcs share an endpoint : $\forall (u_1, v_1) \neq (u_2, v_2) \in M : u_1 \neq u_2 \wedge v_1 \neq v_2$. A matching M* covers *a set of nodes V, or M is a V-matching of g iff $\forall x \in V : \exists (u, v) \in M : u = x \vee v = x$*

The following property states the relationship between matching in the bipartite graphs and solutions of the $MapVar$ constraint.

Property 1. Given the constraint $MapVar(M, S, T)$ and its associated variable-value graph g, assuming the constraint is consistent, we have :

- (1) Any solution $m : s \to t$ contains a t-matching of g, and any t-matching can be extended to a solution.
- (2) An arc (x, v) belongs to a $\underline{D}(T)$-matching of g, *iff* there exists a solution m with $m(x) = v$

Proof. (1) The solution m is surjective; every node of t must have at least one incident arc. If we choose one incident arc per node in t, we have a t-matching as m is a function.

Given a t matching, let $m : s \to t$ be the bijective function corresponding to this matching. Adding arcs to t leads to a surjective function. Let $s' = \underline{D}(S) \cup s$, and $t' = \underline{D}(T) \cup t$. Since the constraint is consistent, $\forall x \in s' \setminus s \ \exists (x, v) \in g : v \neq \perp$, and $\forall v \in t' \setminus t \ \exists (x, v) \in g$. Adding all these arcs leads to a surjective function which is a solution.

(2) ($\Rightarrow$) This is a special case of the second part of (1).

($\Leftarrow$)Let $m : s \to t$ be a solution with $m(x) = v$. We then have $(x, v) \in g$. By (1), the graph g contains a t-matching M which is also a $\underline{D}(T)$-matching as $\underline{D}(T) \subseteq t$. If $(x, v) \in M$ we are done. Assume $(x, v) \notin M$. Then x is free with respect to M because $M(x) = v$. As $v \in t$, v is covered by M; there is a variable node w such that $(w, v) \in M$. Then, x, v, w is an even alternating path starting in a free node. Replacing (w, v) by (x, v) leads to another t-matching, hence a $\underline{D}(T)$-matching of g. ■

From Property 1, an arc-consistency filtering algorithm can be derived : compute the set A of arcs belonging to some $\underline{D}(T)$-matching of the bipartite graph; if $(x, v) \notin A$, remove v from $D(M_x)$. The computation of this set can be done using techniques such as described in [33], with a complexity of $O(mn)$, where n is the size of T, and m is the number of arcs in the variable-value graph.

4 Approximate graph matching and other matching problems

In this section, we define different matching problems ranging from graph homomorphism to approximate subgraph matching. The following definitions apply for directed as well as undirected graphs.

A **graph homomorphism** between a pattern graph $P = (N_p, A_p)$ and a target graph $G = (N, A)$ is a total function $f : N_p \rightarrow N$ respecting the *morphism constraint* $(u, v) \in A_p \Rightarrow (f(u), f(v)) \in A$. The graph P is homomorphic to G through the function f. In a **graph monomorphism**, the function f must be injective. In a **graph isomorphism** the function f must be bijective, and the condition $(u, v) \in A_p \Rightarrow (f(u), f(v)) \in A$ is replaced by $(u, v) \in A_p \Leftrightarrow (f(u), f(v)) \in A$. **Subgraph** isomorphisms is defined over an induced subgraph of the target graph. Notice that subgraph homo/mono-morphism are meaningless as graph homo/mono-morphism already maps N_p to a subset of N. All these problems, except graph isomorphism are NP-complete.

A useful extension is *approximate* subgraph matching, where the pattern graph and the found subgraph in the target graph may differ with respect to their structure [9]. We choose an approach where the approximations are declared by the user in the pattern graph through optional nodes and forbidden arcs.

In the previous graph matching problems, all the nodes of the pattern must be matched. An interesting extension consists in allowing optional nodes in the pattern graph. Those nodes need not necessarily be matched. If they are, all arcs incident to them are considered part of the pattern and the matching constraints apply to them. In other words, the pattern that is used in the morphism problem is an induced subgraph of the pattern containing optional nodes.

In graph isomorphism, if two nodes in the pattern are not related by an arc, this absence of arc is an implicit forbidden arc in the matching. It would be interesting to declare explicitly which arcs are *forbidden*, leading to problems between monomorphism and isomorphism.

In Figure 2, mandatory nodes are represented as filled nodes, and optional nodes are represented as empty nodes. Mandatory arcs are represented with plain line, and arcs incident to optional nodes are represented with dashed lines. Forbidden arcs are represented with a plain line crossed.

In that figure, node 6 cannot be matched to node f because only one of the arcs $(6, 4)$ and $(6, 5)$ in the pattern can be matched in the target. The right side of the figure presents two solutions of the matching problem. The nodes and arcs not matched in the target graph are greyed.

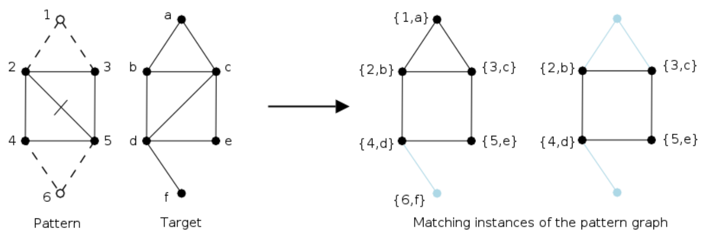

Fig. 2. Example of approximate matching.

A pattern graph with optional nodes and forbidden arcs forms an *approximate pattern graph*, and the corresponding matching is called an *approximate subgraph matching*[9]. We focus here on approximate graph monomorphism.

Definition 1 *An **approximate pattern graph** is a tuple (N_p, O_p, A_p, F_p) where (N_p, A_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 $A_p \cap F_p = \emptyset$.*

Definition 2 *An **approximate subgraph matching** between an approximate pattern graph $P = (N_p, O_p, A_p, F_p)$ and a target graph $G = (N, A)$ is a* partial *function $f : N_p \rightarrow N$ 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. $\forall\, i, j \in dom(f) : (i, j) \in A_p \Rightarrow (f(i), f(j)) \in A$
4. $\forall\, i, j \in dom(f) : (i, j) \in F_p \Rightarrow (f(i), f(j)) \notin A$

The notation $dom(f)$ represents the domain of f. Elements of $dom(f)$ are called the selected nodes of the matching. According to this definition, if $F_p = \emptyset$ the matching is a subgraph monomorphism, and if $F_p = N_p \times N_p \setminus A_p$, the matching is an isomorphism.

5 Modeling graph matching and related problems

In this section, we show how CP(Graph+Map) can be used for modeling and solving a wide range of graph matching problems.

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

– homomorphism versus monomorphism versus isomorphism;
– graph versus subgraph matching;
– exact versus approximate matching

These different problems illustrated in Table 1. All these problems can be modeled and solved through a morphism constraint on a map variable and two graph variables.

5.1 The basic morphism constraints

The two important morphism constraints introduced in this paper are the $SurjMC(P, G, M)$ and $BijMC(P, G, M)$ constraints, which holds when M is a total surjective / bijective mapping from P to G respecting the morphism constraint.

$$\begin{aligned} SurjMC(P, G, M) &\equiv SurjectFct(M, Nodes(P), Nodes(G)) \wedge MC(P, G, M) \\ BijMC(P, G, M) &\equiv BijectFct(M, Nodes(P), Nodes(G)) \wedge MC(P, G, M) \\ \text{with} \quad MC(P, G, M) &\equiv \forall (i, j) \in Arcs(P) : (M(i), M(j)) \in Arcs(G) \end{aligned}$$

We now show how these two morphism constraints can be used to solve the different classes of problems.

5.2 Exact matching

Let p be a pattern graph and g be a target graph. The graphs p and g are ground objects in CP(Graph+Map). Graph homo and monomorphism can easily be modeled as shown in Table 1. Homomorphim (resp. monomorphism) requires a surjective (resp. bijective) function between p and a subgraph of g, respecting the morphism constraint. We use here a graph variable instead of a graph constant for the target graph (G with $D(G) = [\emptyset, g]$)

Graph isomorphism requires a bijective function between p and g respecting two morphism constraints : one between the graphs, and a second between the complementary graphs. This requires a complementary graph constraint $CompGraph(G, Gc)$ which holds if $Nodes(G) = Nodes(Gc) = N$ and $Arcs(Gc) = (N \times N) \setminus Arcs(G)$. For conciseness, we also use the functional notation $Comp(G) = Gc$. In the subgraph isomorphism problem, there should exist a isomorphism between p and an induced subgraph of g.

Exact matching	
homomorphism	$G \subseteq g \wedge SurjMC(p, G, M)$
monomorphism	$G \subseteq g \wedge BijMC(p, G, M)$
isomorphism	$BijMC(p, g, M) \wedge BijMC(Comp(p), Comp(g), M)$
subgraph isomorph.	$G \subseteq^* g \wedge BijMC(p, G, M) \wedge BijMC(Comp(p), Comp(G), M)$
Optional nodes	
homomorphism	$P \in [p_{man}, p] \wedge P \subseteq^* p \wedge G \subseteq g \wedge SurjMC(P, g, M)$
monomorphism	$P \in [p_{man}, p] \wedge P \subseteq^* p \wedge G \subseteq g \wedge BijMC(P, g, M)$
isomorphism	$P \in [p_{man}, p] \wedge P \subseteq^* p \wedge BijMC(P, g, M)$ $\wedge BijMC(Comp(P), Comp(G), M)$
subgraph isomorph.	$G \subseteq^* g \wedge P \in [p_{man}, p] \wedge P \subseteq^* p \wedge BijMC(P, G, M)$ $\wedge BijMC(Comp(P), Comp(g), M)$
Forbidden arcs	
monomorphism	$G \subseteq^* g \wedge BijMC(p, G, M) \wedge BijMC(p_{forb}, Comp(G), M)$

Table 1. Constraints for the matching problems

5.3 Optional nodes and forbidden arcs

To cope with the optional nodes in the pattern graph, we replace the fixed graph pattern by a constrained graph variable, as illustrated in Table 1. Let p be the pattern graph with optional nodes, and p_{man} be the subgraph of p induced by the mandatory nodes of p. Graph monomorphisms with optional nodes amounts to find an intermediate graph 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, this is done through the use of the induced subgraph relation ($\subseteq^*$).

When all the nodes of the pattern graph are optional in the graph monomorphism, we have the *maximum common subgraph* problem by adding the size of P as an objective function. Similarly for subgraph isomorphism, this leads to the *maximum common induced subgraph* problem.

Allowing the specification of a set of forbidden arcs amounts to a simple generalization of the isomorphism problem, lying between monomorphism and isomorphism. As in the model for isomorphism, forbidden arcs are handled through a morphism constraint on the complement of the target graph. This time, only a specified set p_{forb} of arcs are forbidden. Isomorphism constitutes a special case where $p_{forb} = Arcs(Comp(p))$. This illustrated for the monomorphism problem in Table 1

The problem of approximate subgraph matching as defined in section 5, simply combines the use of optional nodes and forbidden arcs. Given an approximate pattern graph (N_p, O_p, A_p, F_p) where (N_p, A_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, A), we define the following CP(Graph+Map) constants :

– p: the pattern graph (N_p, A_p),
– p_{man}: the subgraph of p induced by the mandatory nodes $N_p \setminus O_p$ of p,
– g: the target graph (N, A),
– p_{forb} : the graph (N_p, F_p) of the forbidden arcs.

The modeling of approximate matching is then a combination of graph monomorphism with optional nodes, and forbidden arcs.

$$\begin{aligned} &G \subseteq^* g \wedge P \in [p_{man}, p] \wedge P \subseteq^* p \wedge BijMC(P, G, M) \\ &\qquad \wedge Nodes(Pc) = Nodes(P) \wedge Pc \subseteq^* p_{forb} \wedge BijMC(Pc, Comp(G), M) \end{aligned}$$

5.4 Global constraints

The main difference between the $SurjMC(P, G, M)$ and $BijMC(P, G, M)$ constraints is an alldiff constraint ensuring the bijective property of the mapping M. A direct implementation of these constraints based on their definition would be very inefficient. A global constraint for

$$MC(P, G, M) \equiv \forall (i, j) \in Arcs(P) : (M(i), M(j)) \in Arcs(G)$$

has been designed based on [2, 9], and generalized in the context of graph intervals and our extension to function variables. This global constraint is *algorithmically* global as it achieves the same consistency as the original conjunction of constraints, but more efficiently [36].

Redundant constraint, such as proposed in [2, 9] have also been developed to enhance the pruning. We also specialized global constraints for the different matching families. For instance, a global constraint for filtering subgraph isomorphism was developed and was used to solve difficult instances in [37]. Regarding the approximate matching with optional nodes, the $Mono$ propagator is specialized and assumes that a $P \subseteq^* p$ constraint is posted too, allowing a more efficient pruning. For the isomorphism and for approximate matching with forbidden arcs, a single propagator combining the two $Mono$ propagator is also used, following the ideas developed in [9].

6 Experimental results

This section assesses the performance of the proposed CP(Graph+Map) framework for graph matching. We compare our proposed solution with `vflib` [38, 39], the current state of the art algorithm for subgraph isomorphism, improving over Ullman's algorithm [40].

The CP(Graph+Map) framework has been implemented over the `Gecode` system (`http://www.gecode.org`), including graph variables and propagators, map variables and propagators, together with matching propagators.

Our benchmark set consists of graphs made of different topological structures as explained in [2]. These graphs were generated using the Stanford GraphBase [41], consisting of 1225 undirected instances, and 405 directed instances. The graphs range from 10 to 125 nodes for undirected graphs, and from 10 to 462 for directed graphs.

The experiments consist in performing subgraph monomorphism over the 1225 undirected instances, and subgraph isomorphism over the 405 instances. All solutions are searched. Following the methodology used in [2], we ran the two competing algorithms for five minutes on each of the problem instances. A run is called *solved* if it finishes under five minutes or *unsolved* otherwise. All benchmarks were performed on an Intel Xeon 3 Ghz.

Table 6 shows the experimental results. We report the percentage of solved instances (sol.), the percentage of unsolved instances (unsol), the total running time (tot.T), the mean running time (av.T) and memory (av.M) and the mean running time and memory over instances solved by both approaches (resp. "av.T com." and "av.M com.").

The CP(Graph+Map) model solves more problem instances than the specialized `vflib` algorithm. This difference is significant for subgraph monomorphism (61% vs. 48%). It is interesting to notice that around 4% of the instances solved by `vflib` were not solved by our CP model. This shows that on some instances, standard algorithms can be better, but that globally, CP(Graph+Map) solves more instances. It is clear that the CP approach consumes more memory. The comparison of the average time is clearly in favour of CP(Graph+Map) as it solves more instances. It is more interesting to compare the mean execution time on the commonly solved instances. This shows that the time overhead induced by the CP approach is minimal on the commonly solved instances : about 9% for monomorphism over undirected graphs and 22% for isomorphism over directed graphs.

We conclude that our approach is beneficial to someone willing to pay an average time overhead of 9% to 22% on "simple" instances to be able to solve a fourth of the instances of the benchmark which cannot be solved in the time limit by the other method.

7 Conclusion

In this paper, we showed how the integration of two domains of computation over countable structures, *graphs*[13]. and *maps*, [16], can be used for modeling and solving a wide spectrum of of graph matching problems with any combination of the following properties : monomorphism or isomorphism, graph or subgraph matching, exact or

All solutions; subgraph monomorphism over undirected graphs (5 min. limit)							
	solved	unsolved	tot.T min	av.T sec	av.M kb	av.T com. sec	av.M com. kb
vflib	48%	51%	3273	160	11.91	4.96	97.6
CP(Graph+Map)	61%	38%	2479	121	9115.46	5.43	8243

All solutions; subgraph isomorphism over directed graphs (5 min. limit)							
	solved	unsolved	tot.T min	av.T sec	av.M kb	av.T com. sec	av.M com. kb
vflib	92%	7%	181	26.95	114.28	4.11	4.22
CP(Graph+Map)	96%	3%	109	16.22	2859.85	5.04	2754

Table 2. Comparison of the two methods on monomorphism and isomorphism problems.

approximate matching (user-specified approximation [9]). To achieve this, we needed to generalize the map variables with non-fixed source and target sets (of the Cardinal kind [31]).

We showed how a single constraint able to use both fixed and non-fixed graph variables is sufficient to model all these graphs matching problems. Furthermore we showed that this constraint programming approach is competitive with the state of the art algorithm for subgraph isomorphism vflib based on the Ullman graph matching algorithm; by solving substantially more instances (our approach solves more complex instances) and requiring a small overhead over the simple instances.

Future work includes the definition of consistency for map variables, the analysis of the impact of our flow-based filtering algorithm for map variables, the design of a more efficient algorithm (we target $O(\sqrt{m}n)$) for this global constraint and the extension of graph matching to other graph comparison problems such as subgraph bisimulation [42].

Acknowledgments

The authors want to thank the anonymous reviewers for the helpful comments. Thanks to Pierre Dupont for his comments on an earlier version of this paper. This research is supported by the Walloon Region, project Transmaze (WIST516207) and by the Interuniversity Attraction Poles Programme (Belgian State, Belgian Science Policy).

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 Verlag, S., ed.: International Conference on Principles and Practice of Constraint Programming. (2005) 832–836
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. Puget, J.F.: Symmetry breaking revisited. Constraints **10** (2005) 23–46
12. 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
13. Dooms, G., Deville, Y., Dupont, P.: Cp(graph): Introducing a graph computation domain in constraint programming. In Verlag, S., ed.: International Conference on Principles and Practice of Constraint Programming. (2005) 211–225
14. Gervet, C.: New structures of symbolic constraint objects: sets and graphs. In: Third Workshop on Constraint Logic Programming (WCLP'93), Marseille (1993)
15. Chabrier, A., Danna, E., Pape, C.L., Perron, L.: Solving a network design problem. Annals of Operations Research **130** (2004) 217–239
16. Gervet, C.: Interval propagation to reason about sets: Definition and implementation of a practical language. Constraints **1** (1997) 191–244
17. 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
18. Beldiceanu, N., Contjean, E.: Introducing global constraints in CHIP. Mathematical and Computer Modelling **12** (1994) 97–123
19. Pesant, G., Gendreau, M., Potvin, J., Rousseau, J.: An exact constraint logic programming algorithm for the travelling salesman with time windows. Transportation Science **32** (1996) 12–29
20. Puget, J.F.: Pecos a high level constraint programming language. In: Proceedings of Spicis 92. (1992)
21. Beldiceanu, N., Flener, P., Lorca, X.: The tree constraint. In: CPAIOR. (2005) 64–78
22. 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
23. Cambazard, H., Bourreau, E.: Conception d'une contrainte globale de chemin. In: 10e Journées nationales sur la résolution pratique de problèmes NP-complets (JNPC'04). (2004) 107–121
24. Dooms, G., Katriel, I.: The minimum spanning tree constraint. In: Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming. (2006) 152–166
25. Dooms, G., Katriel, I.: The "not-too-heavy" spanning tree constraint. In: Proceedings of CPAIOR 2007. (2007)
26. Hnich, B.: Function variables for Constraint Programming. PhD thesis, Uppsala University, Department of Information Science (2003)

27. Frisch, A.M., Jefferson, C., Hernandez, B.M., Miguel, I.: The rules of constraint modelling. In: Proceedings of IJCAI 2005. (2005)
28. Lauriere, J.L.: A language and a program for stating and solving combinatorial problems. Artificial Intelligence **10** (1978) 29–128
29. Smith, D.: Structure and design of global search algorithms. Technical Report Tech. Report KES.U.87.12, Kestrel Institute, Palo Alto, Calif. (1987)
30. 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
31. Azevedo, F.: Cardinal: A finite sets constraint solver. Constraints **12** (2007) 93–129
32. Beldiceanu, N.: Global constraints as graph properties on structured network of elementary constraints of the same type. Technical Report T2000/01, SICS (2000)
33. Thiel, S.: Efficient Algorithms for Constraint Propagation and for Processing Tree Descriptions. PhD thesis, University of Saarbrucken (2004)
34. Bessière, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: Filtering algorithms for the nvalue constraint. In: CPAIOR. (2005) 79–93
35. Bessière, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: The range and roots constraints: Specifying counting and occurrence problems. In: IJCAI. (2005) 60–65
36. 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
37. Zampelli, S., Deville, Y., Solnon, C., Sorlin, S., Dupont, P.: Filtering for subgraph isomorphism. In: Proc. 13th Conf. of Principles and Practice of Constraint Programming. Lecture Notes in Computer Science, Springer (2007) 728–742
38. Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. In http://amalfi.dis.unina.it/graph/db/vflib 2.0/doc/vflib.html, ed.: 3rd IAPR-TC15 Workshop on Graph-based Representations. (2001)
39. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: Performance evaluation of the vf graph matching algorithm. In: ICIAP, IEEE Computer Society (1999) 1172–1177
40. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM **23** (1976) 31–42
41. Knuth, D.E.: The Stanford GraphBase. A Platform for Combinatorial Computing. ACM, NY (1993)
42. Dovier, A., Piazza, C.: The subgraph bisimulation problem. IEEE Transaction on Knowledge and Data Engineering **15** (2003) 1055–1056