Declarative Approximate Graph Matching Using A Constraint Approach *

Stéphane Zampelli, Yves Deville, and Pierre Dupont

Université Catholique de Louvain,
Department of Computing Science and Engineering,
2, Place Sainte-Barbe
1348 Louvain-la-Neuve (Belgium)
{sz, yde, pdupont}@info.ucl.ac.be

Abstract. Graph pattern matching is a central application in many fields. However, in many cases, the structure of the pattern can only be approximated and exact matching is then far too accurate. This work aims at proposing a CSP approach for approximate subgraph matching where the potential approximation is declaratively included in the pattern graph as optional nodes and forbidden edges. The model, covering both monomorphism and isomorphism problem, also allows additional properties, such as distance properties, between pairs of nodes in the pattern graph. Such properties can be either stated by the user, or automatically inferred by the system. The model is built through the definition of parametric morphism constraints, allowing an efficient implementation of propagators. An Oz/Mozart implementation has been developped. Experimental results show that our general framework is competitive with a specialized C++ Ullman (exact) matching algorithm, while also offering approximate matching.

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. However, many issues should be considered such as the evaluation of the performance of a CSP approach against traditional algorithms, the development of new global constraints enhancing the pruning, the extension of exact matching to approximate matching.

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

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

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]). 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 [9] 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 [10].

Objectives This work aims at proposing a CSP approach for approximate subgraph matching where the potential approximation is declaratively included in the pattern graph as mandatory/optional nodes/edges. We also want a framework where additional properties between pairs of nodes in the pattern graph, such as distance properties, can be either stated by the user, or automatically inferred by the system. In the former case, such properties can define new approximate patterns. In the latter case, these redundant constraints enhance the pruning.

Results The main contributions of this paper are the following:

- An extension of the CSP model for pattern subgraph matching covering both monomorphism and isomorphism problems, and allowing the specification of additional constraints between pairs of nodes, as well as the derivation of redundant constraints providing more pruning.
- A definition of approximate subgraph matching, including the specification of optional nodes and forbidden edges in the pattern graph, and its associated CSP model.
- A definition of parametric morphism constraints, allowing a simple expression of the above problems.
- An implementation of propagators for the generic constraints.
- Experimentations showing that our general framework is competitive with a specialized C++ Ullman (exact) matching algorithm, while also offering approximate matching.

Outline Sections 2 and 3 introduce basic definitions of subgraph matching and describe the basic framework for monomorphism. Section 4 generalizes the basic monomorphism constraints to parametric constraints in the exact case. Instances of these parametric constraints, such as isomorphism constraints and path constraints, are described in Section 5. Section 6 introduces the approximate matching problem and describes how this problem can be solved by an approximate version of the parametric constraints. Section 7 presents a comparison of our CSP approach with Ullman based graph matching algorithm and shows that the proposed approach is competitive. Section 8 concludes this paper and presents research directions.

2 Background

Before presenting the basic CSP for exact and approximate subgraph matching, we define the notion of subgraph matching.

A **graph** $G = (N, E)$ consists of a **node set** N and an **edge set** $E \subseteq N \times N$, where an edge (u, v) is a pair of nodes. The nodes u and v are the endpoints of the edge (u, v).

The **neighborhood function** $V(a)$ is the set of neighbors of a node a in the underlying graph.

A **subgraph** of a graph $G = (N, E)$ is a graph $S = (N', E')$ where N' is a subset of N and E' is a subset of E.

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

A **subgraph monomorphism** between G_p and G_t is an injective function $f : N_p \rightarrow N_t$ respecting $(u, v) \in E_p \Rightarrow (f(u), f(v)) \in E_t$.

A **subgraph matching** is either a subgraph isomorphism or a subgraph monomorphism.

A constraint model to solve the exact subgraph matching problem has been proposed by several authors [2] [3]. This model focuses on monomorphism and will form our basic monomorphism constraints. The variables $\mathcal{X} = \{x_1, ..., x_n\}$ are the nodes of the pattern graph and their respective domain $D(x_i)$ is the set of target nodes. The assignment must respect two conditions: all variables have a different value and the structure of the pattern must be kept (monomorphism condition). The first condition is implemented with the classical *Alldiff*$(x_1, ..., x_n)$ constraint [11] [12]. The second condition is translated into a monomorphism constraint.

2.1 Monomorphism Constraint

The monomorphism constraint states that if an edge exists between two pattern nodes, then an edge must exist between their corresponding images :

$$\forall\ (i, j) \in E_p : (f(i), f(j)) \in E_t\ .$$

The corresponding basic monomorphism constraint is defined as :

$$MC(x_i, x_j) \equiv (i, j) \in E_p \Rightarrow (x_i, x_j) \in E_t\ .$$

In the rest of the paper, $N = |N_p|$, $E = |E_p|$, $D = |N_t|$ and d is the average degree of the target graph. A classical AC-consistency algorithm would cost $O(ED^2)$ amortized time [2]. By using the problem structure, its amortized complexity can be reduced to $O(NDd)$ [2]. We note n the average degree of the pattern graph.

A global constraint $MC(x_1, ..., x_n)$ can be formulated, instead of having one constraint MC per node pair:

$$MC(x_1, ..., x_n) = \bigwedge_{i,j} MC(x_i, x_j) \ .$$

The global basic monomorphism constraint $MC(x_1, ..., x_n)$ can be expressed as:

$$\forall\ i \in N_p\ \forall\ a \in N_t : |D(x_i) \cap V_t(a)| = 0 \Rightarrow a \notin D(x_j)\ \ \forall\ j \in V_p(i) \ ,$$

where $V_p(i) = \{j \in N_p \mid (i,j) \in E_p\}$ and $V_t(i) = \{j \in N_t \mid (i,j) \in E_t\}$.

The proposed propagator keeps track of relations between all the target nodes and the domain $D(x_i)$ in a structure $S(i,a) = |D(x_i) \cap V_t(a)|$ representing the number of relations between a target node a and $D(x_i)$. Whenever the neighbors of a target node a have no relation with $D(x_i)$, that is when $S(i,a) = 0$, node a is pruned from all neighbors of x_i. The Algorithm 2.1 shows an implementation of the global morphism constraint. 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.

Algorithm 1: Morphism Constraint

```
Propagate_MC(i,a)
// Element a exits from D(x_i)
for b ∈ V_t(a) do
    S(i,b) ← S(i,b) − 1
    if S(i,b) = 0 then
        foreach j ∈ V_p(i) do
            D(x_j) ← D(x_j) \ {b}
```

2.2 Local Alldiff Constraint

A redundant constraint pruning the search space has been proposed in [2]. This constraint is a local Alldiff constraint [11], enforcing that the number of candidates available in the union of the domains of x_i's neighbors should not be less than the actual number of x_i neighbors in the pattern graph:

$$LA(x_i) \equiv |\cup_{j \in V_p(i)} D(x_j) \cap V_t(x_i)| \geq |V_p(i)| \ . \tag{1}$$

An algorithmic global constraint $LA(x_1, ..., x_n)$ can be formulated, instead of having one constraint LA per node :

$$LA(x_1, ..., x_n) \equiv \bigwedge_i LA(x_i) \ .$$

A structure $CT(i,a) = |\cup_{j \in V_p(i)} D(x_j) \cap V_t(a)|$ is updated through the use of an intermediate structure $R(i,a) = |\{j \in V_p(i) \mid a \in D(x_j)\}|$. The structure

$R(i,a)$ counts the number of neighbors of x_i which have a in their domain. Whenever $R(i,a)$ equals 0, $CT(i,b)$ diminishes by 1 for all b in the neighbor of a in the target graph. The expression $|V_p(i)|$ can be obtained in $O(1)$. Algorithm 2 describes an implementation of the $LA(x_1,...,x_n)$ constraint.

The amortized complexity of this redundant constraint is $O(NDd)$ and its space complexity is $O(ND)$. The preprocessing time to build the $CT(i,a)$ and $R(i,a)$ structures is $O(NDd)$.

Algorithm 2: Local alldiff constraint

```
Propagate_LA(i,a)
// Element a exits from D(x_i)
for j ∈ V_p(i) do
    R(j,a) ← R(j,a) − 1
    if R(j,a) = 0 then
        foreach b ∈ V_t(a) do
            CT(j,b) ← Ct(j,b) − 1
            if CT(j,b) < |V_t(j)| then
                D(x_j) ← D(x_j) \ {b}
```

3 Generic Subgraph Matching Constraints

In this section we present new parameterized global constraints able to handle different type of constraints. These constraints will be instantiated to different matching constraints.

The MC constraint is redefined as a parametric constraint, taking pattern pair node relations A and target pair node relations B as parameter :

$$MC_p(x_1,...,x_n,A,B) \equiv \bigwedge_{i,j} (i,j) \in A \Rightarrow (x_i,x_j) \in B \ .$$

The propagator of this parametric MC constraint is given by Algorithm 1, where the neighborhood functions $V_p(\cdot)$ and $V_t(\cdot)$ are specialized to the considered instance of the constraint. More precisely, $V_p(i,A) = \{j \in Np \mid (i,j) \in A\}$ and $V_t(a,B) = \{b \in Np \mid (a,b) \in B\}$ respectively. As a consequence, $S(i,a)$ is redefined as $|D(x_i) \cap V_t(a,B)|$.

The LA constraint can also be parameterized with the relations A and B :

$$LA_p(x_i,A,B) \equiv |\cup_{j \in V_p(j,A)} D(x_j) \cap V_t(x_i,B)| \geq |V_p(i,A)|$$
$$LA_p(x_1,...,x_n,A,B) \equiv \bigwedge_i LA(x_i,A,B)$$

Algorithm 2, with suitable specific structures, is a possible implementation for instances of this constraint.

The problem of subgraph monomorphism can then be expressed as :

$$\text{alldiff}(x_1,...,x_n) \wedge MC_p(x_1,...,x_n,E_p,E_t) \wedge LA_p(x_1,...,x_n,E_p,E_t)$$

4 Specifying Additional Constraints

Additional constraints for the matching problem can be expressed as instances of the parametric morphism constraint.

4.1 Isomorphism as Monomorphism Matching

Isomorphism condition states that if an edge does not exist between two pattern nodes, then an edge should not exist between their corresponding images :

$$\forall\ (i,j) \notin E_p : (f(i), f(j)) \notin E_t\ .$$

The problem of subgraph isomorphism can be stated easily by using complementary edge sets $\overline{E}_p = \{(i,j) \in N_p \times N_p \mid (i,j) \notin E_p\ \}$ and $\overline{E}_t = \{(i,j) \in N_t \times N_t \mid (i,j) \notin E_t\ \}$ as parameters :

$$\text{alldiff}(x_1, ..., x_n)\ \wedge\ MC_p(x_1, ..., x_n, E_p, E_t)\ \wedge\ MC_p(x_1, ..., x_n, \overline{E}_p, \overline{E}_t)$$
$$\wedge\ LA_p(x_1, ..., x_n, E_p, E_t)\ \wedge\ LA_p(x_1, ..., x_n, \overline{E}_p, \overline{E}_t)$$

4.2 Path and Shortest Path Distance Constraint

In this section we formulate a new constraint between pair of nodes based on the path and shortest path distance. It can be seen as a generalization of other works based on shortest path distance as filtering and checking methods [9] [13], where only initial filtering and checking is achieved. In our method, the path constraints does this initial filtering but also propagates.

Definition 1 *A node a is at distance k from node b in a graph if and only if there exists a shortest path of distance k between them. dist*(a, b) *denotes the shortest path distance between a and b.*

A shortest path monomorphism constraint (for a given distance k) can be formulated as $MC_{dist}(x_1, ..., x_n, k) \equiv \bigwedge_{i,j} dist(i,j) = k \Rightarrow dist(x_i, x_j) \leq k$.

Similarly, a shortest path isomorphism constraint (for a given distance k) can be formulated as $MC_{dist}(x_1, ..., x_n, k) \equiv \bigwedge_{i,j} dist(i,j) = k \Rightarrow dist(x_i, x_j) = k$.

Suppose $E_p^k = \{(i,j) \in N_p \times N_p \mid dist(i,j) = k\}$ and $E_t^k = \{(a,b) \in N_t \times N_t \mid dist(a,b) \leq k\}$. Then MC_{dist} is equivalent to $MC_{dist}(x_1, ..., x_n, k) \equiv MC(x_1, .., x_n, E_p^k, E_t^k)$.

The expression $path(i,j,k)$ denotes that there is a path of length k between i and j. The path constraint can be formulated as $MC_{path}(x_1, ..., x_n) \equiv \bigwedge_{i,j} path(i,j,k) \Rightarrow path(x_i, x_j, k)$.

Suppose $E_p^k = \{(i,j) \in N_p \times N_p \mid path(i,j,k)\}$ and $E_t^k = \{(a,b) \in N_t \times N_t \mid path(a,b,k)\}$. We can see that MC_{path} is equivalent to $MC_{path}(x_1, ..., x_n) \equiv MC(x_1, .., x_n, E_p^k, E_t^k)$.

The number of (E_p^k, E_t^k) couples is bound by the diameter of the pattern graph, which is, in the worst case, $O(N)$. The time complexity of all these new

constraints is thus $O(N^2Dd)$ and their spatial complexity $O(N^2D)$. Preprocessing time to compute path and shortest-path distance adjacency matrices is $O(D^3)$.

Shortest path constraints lead to poor pruning when k increases since the average degree of the graphs E_p^k and E_t^k is $O(d^k)$. All path constraints are however redundant, meaning they are necessary conditions of the matching. Only a subset of these constraints can be chosen. One could select only path of distance two and three, resulting in a $O(ND3d) = O(NDd)$ time complexity and a $O(3ND) = O(ND)$ spatial complexity. One could use the path distance only from one specific node to all the another nodes. We call this kind of node an *orbit*. Each orbit will cost an additional $O(NDd)$. One could select specific path distance constraints between chosen nodes.

5 Approximate Subgraph Matching

5.1 Problem Definition

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.

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 edges in a monomorphism problem is useless as it is equivalent to omitting the edge in the pattern. The status of the edges depends on the optional state of their endpoints. An edge having an optional node as one of its endpoints is optional. An optional edge is not considered in the matching if one of its endpoints is not part of the matching. Otherwise, the edge must also be a part of the matching.

Forbidden edges *Edges* 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 (u, v) must not exist in the target graph. A pattern graph with all its complementary edges declared as forbidden induces a subgraph isomorphism instead of a subgraph monomorphism.

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

Definition 2 *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 edges, with $E_p \cap F_p = \emptyset$.*

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

Definition 3 *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 $f : N_p \to N_t$ 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 lower bound f_{lb} consists of all selected nodes, and its upper bound f_{glb} 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 edge between two selected endpoints must always be present in the target. Condition 4 forbids the presence of an edge in the matching between node $(f(u), f(v))$ if the edge (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 1. In this figure, mandatory nodes are represented as filled nodes, and optional nodes are represented as empty nodes. Mandatory edges are represented with plain line, and optional edges are represented with dashed lines. Forbidden edges are represented with a plain line crossed. Intuitively, one could think that edge $(5, 6)$ in the pattern could be discarded, while node 6 could be selected together with edge $(4, 6)$. In fact, because of condition 3, matching of node 6 would require the edge $(5, 6)$ to be present in the target. Only two subgraphs match this pattern as shown on the right side of Figure 1. The nodes and edges not selected in the target graph are grey.

Fig. 1. Example of approximate matching.

In an approximate subgraph matching, the number of possible solutions may be higher than in exact matching. One could therefore add some optimization criteria on the results, such as maximizing the number of edges in the matching.

5.2 Parametric Constraints for Optional Nodes

Morphism constraint on this approximate matching should handle the optional nodes, but also be parameterized, because expressiveness and pruning should be kept. The approximate morphism constraint can be defined as follows :

$$MC_{pa}(x_1, ..., x_n, A, B) \equiv \bigwedge_{i,j}(i,j) \in A \wedge i, j \in dom(f) \Rightarrow (x_i, x_j) \in B$$

The former constraint states that a morphism relation between two pattern nodes x_i and x_j must be forced if and only if they are present in the domain of f. Using a MC-like implementation, the MC_{pa} constraint can be rewritten as :

$$\forall\, i \in N_p\ \forall\, a \in N_t : (\ |D(x_i) \cap V_t(a)| = 0 \\ \wedge\ i \in dom(f)\ ) \Rightarrow a \notin D(x_j) \quad \forall\, j \in V_p(i)\ .$$

The additional condition $i \in dom(f)$ states that only selected nodes should propagate under the morphism condition. The propagation of the morphism constraint of an optional i is computed but performed only when i is in the domain of f.

As depicted in Figure 2, all selected nodes propagate in their neighborhood but optional nodes propagate only when they are selected.

Fig. 2. Pruning method for the approximate morphism condition

MC_{pa} is a simple extension of the implementation of MC_p one (Algorithm 1). If i is not selected and there exists a such that $S(i,a) = 0$, the propagator waits for node i to be selected to trigger the actual propagation.

5.3 Constraints for Forbidden Edges

A constraint for the forbidden edges (condition 4 in the matching) can be obtained by using parameterized MC_{pa} :

$$MC_{pa}(x1, ..., x_n, F_p, \overline{E}_t)$$

The constraints for the approximate matching problem are then :

$$\text{alldiff}(x_1, ..., x_n)\ \wedge\ MC_{pa}(x1, ..., x_n, E_p, E_t)\ \wedge\ MC_{pa}(x1, ..., x_n, F_p, \overline{E}_t)\ .$$

Using these two MC_{pa} constraints has a major drawback. The neighborhood function $\overline{V}_p(i) = \{j \mid (i,j) \in F_p\}$ and especially $\overline{V}_t(a) = \{b \mid (a,b) \notin E_t\}$ may

increase time complexity, because most of the time is spent in the loop upon $\overline{V}_t(a)$. Whatever the average degree of the target graph is, one of the constraints has a $O(ND^2)$ complexity. Moreover, a second structure $\overline{S}(i,a) = |D(x_i) \cap \overline{V}_t(a)|$ has to be created. These two constraints can however be expressed within a single propagator, thanks to $\overline{S}(i,a) = 0 \Leftrightarrow S(i,a) = |D(x_i)|$. Indeed, $S(i,a) + \overline{S}(i,a) = |D(x_i) \cap V(a)| + |D(x_i) \cap \overline{V}(a)| = |D(x_i) \cap (V(a) \cup \overline{V}(a))| = |D(x_i) \cap N_t| = |D(x_i)|$.

The two propagators implementing the two instances of the MC_{pa} constraint can be implemented in an unique propagator $MCFA$ described in Algorithm 3. Record that $S(i,a) = num$ represents the number of relations between target node a and $D(x_i)$. Since $S(i,a)$ is computed over $D(x_i)$ that may be different from the actual $D(x_i)$, a counter $size$ is added to $S(i,a)$ structure, representing the size of $D(x_i)$ over which value num is computed.

Algorithm 3: Morphism and Forbidden Edges Constraint

```
PropagateMCFA(i,a,E_p1,E_p2,E_t)
// Element a exits from D(x_i)
S(i,a,E_t).size ← S(i,a,E_t).size − 1
for b ∈ V(a,E_t) do
    S(i,b,E_t).num ← S(i,b,E_t).num − 1
    num = S(i,b,E_t).num
    size = S(i,b,E_t).size
    if size == num then
        PropaNeigh(i,b,E_p2)
    if num == 0 then
        PropaNeigh(i,b,E_p1)
Procedure PropaNeigh(i,b,E_p)
Wait until i ∈ dom(f)
for j ∈ V(i,E_p) do
    D(x_j) ← D(x_j) \ b
```

The LA constraint may also be adapted for approximate matching. Constraint LA infers propagation on its x_i variable on the basis of x_i neighborhood.

Definition 4 *The selected neighborhood function $V_p^+(i)$, with respect to a finite set variable $D = [f_{lb}, f_{gb}]$ representing $dom(f)$ of a node i in an approximate pattern graph is the set $\{\ j \mid j \in V_p(i) \wedge j \in f_{lb}\}$.*

The function $V_p^+(i)$ creates an LA_{pa}^+ constraint, playing the same role as in the exact case :

$$LA_{pa}^+(x_i, A, B) \equiv |\cup_{j \in V_p^+(i,A)} D(x_j) \cap V_t(x_i, B)| \geq |V_p^+(i, A)|$$
$$LA_{pa}^+(x_1, ..., x_n, A, B) \equiv \bigwedge_i LA_{pa}^+(x_i, A, B)\ .$$

Similarly to the LA_p constraint, LA^+_{pa} plays a pruning role. It can be implemented by maintaining the neighborhood variable, with an $O(d)$ time complexity, whenever the domain of x_i is pruned. The structure $R^+(i,a) = |\{\ j \in V^+_p(i,A) \mid a \in D(x_j)\ \}|$ depends not only on the domain of the neighborhood of x_i but also on the neighborhood variable. Whenever the lower bound of $V^+_p(i,A)$ changes, the structure $R^+(i,\cdot)$ must be updated in $O(D)$, resulting in a $O(ND^2)$ amortized complexity. Moreover, $R^+(i,a)$ may be incremented from zero to one, resulting in an increment of $CT^+(i,a) = |\cup_{j \in V^+_p(i,A)} D(x_j) \cap V_t(a,B)|$, which is not monotone. Nevertheless, when condition $CT^+(i,a) < |V^+_p(i,A)|$ is fulfilled, a can be safely pruned from x_i, because if there is not enough candidates for a subgroup of the neighborhood, node i cannot be mapped to node a, even if the condition still holds for the group.

5.4 Extending approximate pattern

Until now parametric constraints has been used for designing global or redundant constraints. In fact they can also be instantiated to constraints declaring distance constraints between specific pattern nodes. Such properties state new information on the pattern graph. For example, a constraint $PathAtMost(x_i, x_j, k)$ could state that there exists a shortest path of distance k or less between node i and j :

$$PathAtMost(x_i, x_j, k) \equiv dist(x_i, x_j) \leq k\ .$$

Similarly, a constraint $Path(x_i, x_j, k)$ could state that there exists a path of length k between node i and j :

$$Path(x_i, x_j, k) \equiv path(x_i, x_j, k)\ .$$

A matching declaring this $Path$ constraint between two pattern nodes i and j states the existence of a path of length k, in the target graph, between nodes $f(i)$ and $f(j)$. Such additional constraints enriches the approximation on the pattern graph. It is clear that parametric constraints can be instantiated to other types of constraints as long as they are properties concerning pair nodes of the pattern and that those properties can be precomputed or dynamically computed on the target graph. For example, richer path constraints could state that there exists a path of length k containing two nodes of a given type.

6 Experiments

Our CSP model for approximate subgraph matching has been implemented inside the CSP framework of Oz/Mozart (`www.mozart-oz.org`). Both parametric propagators $MC_{pa}(x_1, ..., x_n, A, B)$ and $LA_{pa}(x_1, ..., x_n, A, B)$ were implemented as well as $MCFA$. Various transformations of E_p and E_t were automated to instantiate propagators for the forbidden edges and the distance constraints. We

also included facility constraints to declare distance constraints between specific pattern nodes.

First part of the experimental tests aims at comparing the CSP approach with a dedicated algorithm for subgraph matching. The selected algorithm is an improvement of Ullmann's algorithm [14] called vflib, described into [13]. The C++ implementation provided by the authors is used. We have also reimplemented the vflib algorithm in Oz/Mozart.

Two distinct sets of graphs were selected. The first set comes from [15] and consists of 3000 directed instances divided in three classes : first one has probability $\eta = 0.01$ (noted r001 in Table 1) that an edge is present between two distinct node n and n', second one has a 0.05 probability (noted r005) and third one has a 0.1 probability (noted r01). Those graphs were used to evaluate vflib algorithm performance [13]. In our experiments, target graph size (the number of nodes in the target graph) ranges from 20 to 200, pattern graph size is 20% of the target graph size, and all solutions are searched. From a topological point of view, a N nodes graph generated with a probability of η has a mean degree of ηN and each node has a degree close to this mean degree. We call that kind of graphs *uniform* because a subgraph has a structure close to another subgraph in the same graph. The second set contains graphs having different topological structures as explained in [2]. These graphs were generated using the Stanford GraphBase [16] and are all graphs tested in [2], consisting of 406 directed instances.

Tables 1,2 and 3 show the results for the first graph database and the GraphBase graphs. The subgraph matching is a monomorphism. Total and mean time reported concern solved instances only. Following the methodology used into [2], we put a time limit on any given run. In Table 1, left column describes the number of problems solved within a time limit of 5 minutes and right column within 10 minutes, for each set of a given target graph size 60, 80, 100, and 200. All benchmarks were performed on an Intel Xeon 3 Ghz. The three algorithms (original C++ vflib, vflib in Oz, and our CSP in Oz) solve all instances for graph size 20 and 40 within time limit. On the first graph database, one can also measure the overhead of implementing vflib algorithm in Oz. The CSP approach is outperformed by the vflib algorithm for the first graph database in Table 1, but outperforms the vflib algorithm for the Stanford GraphBase set. This comes from the fact that topological properties in the first graph database set are different from GraphBase set. In uniform graphs, the probability that a variable has an empty domain thanks to conjunction of $MC(x_i, x_j)$ constraints is low. This explains that CSP performances decrease as target size increases in Table 1. The vflib algorithm is effective in this case. When measuring performance on the set of graphs which is not uniform, CSP outperforms the vflib algorithm when looking either for one solution or for all solutions.

Benefits of the unique $MCFA$ propagator instead of the conjunction of the two MC_{pa} are shown in Table 4. The subgraph isomorphism problem is solved by using forbidden edges. Left column shows a set of runs with both MC_{pa} handling the isomorphism. Right column shows the same set of runs with the

unique propagator $MCFA$ handling the isomorphism. As expected, the $MCFA$ propagator solves more problems, and mean time over solved problems decreases.

In most cases redundant path constraints do not reduce the total time as average degree increases with distance. Path constraints are useful for enhancing the expressiveness of the pattern graph. We tested influence of specifying an additional distance constraint between two nodes (left graph in Figure 3). The pattern graph has size 20. As expected, the greater the distance, more time is needed to find all solutions as the search space is higher. This experiment underlies the feasibility using additional distance constraints between nodes, viewed as expressive constraints instead of redundant constraints, in the pattern graph if the distance is not too high (≤ 3). Such an approximate pattern may be especially usefull in domains such as bioinformatics. Approximate matching has been evaluated by declaring two constraints of distance 3 shortest path on the pattern graph and 40% of its nodes as optional. The pattern graph is matched against 100 distinct instances of a target graph made of 100 nodes, searching for all solutions. Two curves are shown in the right graph in Figure 3 in a logarithmic scale. The lower one shows the matching of the pattern graph without its distance constraints and optional nodes. The upper one represents the running time of the approximate matching. A constant factor exists between the two sets of runs and a majority of the executions are below 10 seconds.

Table 1. Comparison over uniform directed graphs.

vflib C++ 5 min.						
	r001		r005		r01	
	solved	unsol	solved	unsol	solved	unsol
60	100	0	100	0	96	4
80	100	0	94	6	98	2
100	100	0	88	12	99	1
200	74	26	84	16	97	3

vflib C++ 10 min.						
	r001		r005		r01	
	solved	unsol	solved	unsol	solved	unsol
60	100	0	100	0	96	4
80	100	0	94	6	98	2
100	100	0	89	11	99	1
200	81	19	87	13	99	1

ozvflib 5 min.						
	r001		r005		r01	
	solved	unsol	solved	unsol	solved	unsol
60	100	0	85	15	76	24
80	100	0	76	24	83	17
100	100	0	56	44	88	12
200	16	84	53	47	69	31

ozvflib 10 min.						
	r001		r005		r01	
	solved	unsol	solved	unsol	solved	unsol
60	100	0	89	11	81	19
80	100	0	79	21	85	15
100	100	0	62	38	91	9
200	18	82	57	43	79	21

CSP MC 5 min.						
	r001		r005		r01	
	solved	unsol	solved	unsol	solved	unsol
60	100	0	96	4	86	14
80	100	0	84	16	91	9
100	100	0	82	18	93	7
200	33	67	77	23	51	49

CSP MC 10 min.						
	r001		r005		r01	
	solved	unsol	solved	unsol	solved	unsol
60	100	0	96	4	91	9
80	100	0	86	14	93	7
100	100	0	86	14	99	1
200	40	60	87	13	86	14

Table 2. Comparison over GraphBase directed graphs.

One solution 5 min.				
	solved	unsol	total time	mean time
vflib C++	80,5%	19,5%	8.89 min.	0.02 min.
ozvflib	78,5%	21,5%	17.67 min.	0.04 min.
CSP	87%	13%	36.64 min.	0.09 min.

All solutions 5 min.				
	solved	unsol	total time	mean time
vflib C++	63,7%	36,3%	12.01 min.	0.02 min.
ozvflib	59,8%	40,2%	11.52 min.	0.02 min.
CSP	68,7%	31,3%	31.4 min.	0.07 min.

Table 3. Comparison over GraphBase undirected graphs.

One solution 5 min.				
	solved	unsol	total time	mean time
vflib C++	64,4%	35,6%	8.14 min.	0.006 min.
ozvflib	58,2%	41,8%	8.6 min.	0.04 min.
CSP	64,4%	35,6%	18.24 min.	0.01 min.

All solutions 5 min.				
	solved	unsol	total time	mean time
vflib C++	48,3%	51,7%	9.31 min.	0.007 min.
ozvflib	39,5%	60,5%	4.43 min.	0.003 min.
CSP	57,7%	42,3%	11.39 min.	0.009 min.

Table 4. MC_{pa} and MC_{fa} versus $MCFA$

CSP MC_p and MC_{fa} 5 min.						
	r001		r005		r01	
	solved	unsol	solved	unsol	solved	unsol
100	100	0	100	0	96	4
200	79	21	62	38	10	90

CSP $MCFA$ 5 min.						
	r001		r005		r01	
	solved	unsol	solved	unsol	solved	unsol
100	100	0	100	0	99	1
200	83	17	80	20	34	66

7 Conclusion

In this paper, we proposed a CSP approach for approximate subgraph matching. The model handles both monomorphism and isomorphism problems. It also allows the specification of additional constraints between pairs of nodes (such as distance constraints), as well as the derivation of redundant constraints providing more pruning. Approximation is specified through optional nodes and forbidden edges, as well as additional constraints. The CSP model is expressed through two parametric constraints, allowing a simple and versatile modeling of various classes of matching problems. Propagators of the constraints have been described, supported by an Oz/Mozart implementation. Experimentations showed that our CSP approach for exact matching is competitive with a specialized C++ Ullman matching algorithms, and illustrated its versatility for approximate subgraph matching.

The proposed framework for declarative approximate subgraph matching open various research directions. Better heuristics could be developed when searching for an approximate matching. Our algorithm for exact matching could also be compared with other algorithms dedicated to the largest common subgraph problem. We also intend to apply our approximate matching algorithm for the analysis of biochemical networks. New approximations could be defined on the pattern graph, along with new constraints and propagators. Finally, as

Fig. 3. Influence of distance over running time and approximate matching running times.

the (approximate) matching is expressed as a combination of (parameterized) constraints, subgraph matching could be integrated in a constraint language handling graph variables, such as CP(Graph) [17].

References

1. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. IJPRAI **18**(3) (2004) 265–298
2. Larrosa, J., Valiente, G.: Constraint satisfaction algorithms for graph pattern matching. Mathematical. Structures in Comp. Sci. **12**(4) (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**(1-2) (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**(5) (1998) 493–504
6. DePiero, F., Krout, D.: An algorithm using length-r paths to approximate subgraph isomorphism. Pattern Recogn. Lett. **24**(1-3) (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. 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
10. 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
11. Regin, J.C.: A filtering algorithm for constraints of difference in CSPs. In: Proc. 12th Conf. American Assoc. Artificial Intelligence. Volume 1., Amer. Assoc. Artificial Intelligence (1994) 362–367
12. van Hoeve, W.J.: The alldifferent constraint: A survey. CoRR **cs.PL/0105015** (2001)
13. 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
14. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM **23**(1) (1976) 31–42
15. Foggia, P., Sansone, C., Vento, M.: A database of graphs for isomorphism and sub-graph isomorphism benchmarcking. CoRR **cs.PL/0105015** (2001)
16. Knuth, D.E.: The Stanford GraphBase. A Platform for Combinatorial Computing. acm, ny (1993)
17. Dooms, G.: Cp(graph): Introducing a graph computation domain in constraint programming (accepted paper). CP2005 (2005)