# Filtering for Subgraph Isomorphism

Stéphane Zampelli[1], Yves Deville[1], Christine Solnon[2],
Sébastien Sorlin[2], Pierre Dupont[1]

[1] Université catholique de Louvain, Department of Computing Science and
Engineering, Place Sainte-Barbe 2, 1348 Louvain-la-Neuve (Belgium)
{sz,yde,pdupont}@info.ucl.ac.be
[2] LIRIS, CNRS UMR 5205, University of Lyon I, 43 Bd du 11 Novembre, 69622
Villeurbanne Cedex (France)
{christine.solnon,sebastien.sorlin}@liris.cnrs.fr

**Abstract.** A subgraph isomorphism problem consists in deciding if there exists a copy of a pattern graph in a target graph. We introduce in this paper a filtering algorithm dedicated to this problem. The main idea is to label every node with respect to its relationships with other nodes of the graph, and to define a partial order on these labels in order to express compatibility of labels for subgraph isomorphism. This partial order over labels is used to filter domains. Labelings can also be strengthened by adding information from the labels of the neighbors. Such a strengthening can be applied iteratively until a fixpoint is reached. Practical experiments illustrate that our new filtering approach is more effective on difficult instances of scale free graphs than state-of-the-art algorithms and other CP approaches.

## 1 Introduction

Graphs are widely used in real-life applications to represent structured objects, e.g., molecules, images, or biological networks. In many of these applications, one looks for a copy of a pattern graph into a target graph [1]. This problem, known as subgraph isomorphism, is NP-complete [2] in the general case.

There exists dedicated algorithms for solving subgraph isomorphism problems, such as [3, 4]. However, such dedicated algorithms can hardly be used to solve more general problems, with additional constraints, or approximate subgraph isomorphism problems, such as the one introduced in [5].

An attractive alternative to these dedicated algorithms is Constraint Programming (CP), which provides a generic framework for solving constraint satisfaction problems (CSP). Indeed, subgraph isomorphism problems may be formulated as CSP in a straightforward way [6, 7]. To make CP competitive with dedicated approaches for these problems, [8] has introduced a global monomorphism constraint, and an associated filtering algorithm, together with redundant *Alldiff* constraints. [5] has extended this work to approximate subgraph isomorphism, and has shown that CP is competitive with dedicated approaches.

**Contribution.** In this paper, we introduce a new filtering algorithm for the subgraph isomorphism problem that exploits the global structure of the graph to achieve a stronger partial consistency. This work takes inspiration from the partition refinement procedure used in Nauty [9] and Saucy [10] for finding graph automorphisms: the idea is to label every node by some invariant property, such as node degrees, and to iteratively extend labels by considering labels of adjacent nodes. Similar labelings are used in [11, 12] to define filtering algorithms for the graph isomorphism problem: the idea is to remove from the domain of a variable associated to a node $v$ every node the label of which is different from the label of $v$. The extension of such a label-based filtering to subgraph isomorphism problems mainly requires to define a partial order on labels in order to express compatibility of labels for subgraph isomorphism: this partial order is used to remove from the domain of a variable associated to a node $v$ every node the label of which is not compatible with the label of $v$. We show that this extension is more effective on difficult instances of scale free graphs than state-of-the-art subgraph isomorphism algorithms and other CP approaches.

**Outline of the paper.** Section 2 presents the subgraph isomorphism problem and related CP models. Section 3 describes the theoretical framework of our filtering: it first introduces the concept of labeling, and shows how labelings can be used for filtering; it then shows that labelings can be iteratively strengthened by adding information from labels of neighbors. Section 4 introduces the practical framework and describes how to compute a label strengthening. An exact algorithm as well as an approximate version are provided. Experimental results are described in Section 5.

## 2 Subgraph Isomorphism

### 2.1 Definitions

An (undirected) *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 couple of nodes.

A *subgraph isomorphism problem* between a pattern graph $G_p = (N_p, E_p)$ and a target graph $G_t = (N_t, E_t)$ consists in deciding whether $G_p$ is isomorphic to some subgraph of $G_t$. More precisely, one should find an injective function $f : N_p \rightarrow N_t$ such that $\forall (u, v) \in N_p \times N_p, (u, v) \in E_p \Rightarrow (f(u), f(v)) \in E_t$. The problem is also called subgraph monomorphism problem or subgraph matching in the literature. The function $f$ is called a *subgraph matching function.*

In the following, we assume $G_p = (N_p, E_p)$ and $G_t = (N_t, E_t)$ to be the underlying instance of subgraph isomorphism problem. We also define $Node = N_p \cup N_t$, $Edge = E_p \cup E_t$, $n_p = \#N_p$, $n_t = \#N_t$, $n = \#Node$, $d_p$ and $d_t$ the maximal degree of the graphs $G_p$ and $G_t$, and $d = max(d_p, d_t)$.

### 2.2 CP Models for Subgraph Isomorphism

A subgraph isomorphism problem can be formulated as a CSP in a straightforward way [6–8]. A variable $x_u$ is associated with every node $u$ of the pattern graph and its domain is the set of target nodes. A global Alldiff constraint [13] ensures that the matching function is injective. Edge matching is ensured by a set of binary constraints:

$$\forall (u,v) \in N_p \times N_p,\ c2(x_u, x_v) \equiv ((u,v) \in E_p \Rightarrow (x_u, x_v) \in E_t)\ .$$

## 3 Theoretical Framework

This section introduces a new filtering algorithm for subgraph isomorphism. We will show in the next section how filtering can be achieved in practice from this theoretical work.

### 3.1 Subgraph Isomorphism Consistent Labelings

**Definition 1.** *A* labeling $l$ *is defined by a triple* $(\mathbb{L}, \preceq, \alpha)$ *such that*

- $\mathbb{L}$ *is a set of labels that may be associated to nodes;*
- $\preceq \subseteq \mathbb{L} \times \mathbb{L}$ *is a partial order on* $\mathbb{L}$*;*
- $\alpha : \text{Node} \rightarrow \mathbb{L}$ *is a total function assigning a label* $\alpha(v)$ *to every node* $v$*.*

A labeling induces a compatibility relation between nodes of the pattern graph and the target graph.

**Definition 2.** *The set of* compatible couples of nodes *induced by a labeling* $l = (\mathbb{L}, \preceq, \alpha)$ *is defined by* $CC_l = \{(u,v) \in N_p \times N_t \mid \alpha(u) \preceq \alpha(v)\}$

This compatibility relation can be used to filter the domain of a variable $x_u$ associated with a node $u$ of the pattern graph by removing from it every node $v$ of the target graph such that $(u,v) \notin CC_l$.

The goal of this work is to find a labeling that filters domains as strongly as possible *without removing solutions* to the subgraph isomorphism problem, *i.e.*, if a node $v$ of the pattern graph may be matched to a node $u$ of the target graph by a subgraph matching function, then the label of $v$ must be compatible with the label of $u$. This property is called subgraph isomorphism consistency.

**Definition 3.** *A labeling* $l$ *is* subgraph isomorphism consistent *(SIC) iff for any subgraph matching function* $f$*, we have* $\forall v \in N_p, (v, f(v)) \in CC_l$*.*

In the context of graph isomorphism, such as in [9], as opposed to subgraph isomorphism studied here, an SIC labeling is often called an *invariant*. In this case, the partial ordering is replaced by an equality condition: two nodes are compatible if they have the same label.

Many graph properties, that are “invariant” to subgraph isomorphism, may be used to define SIC labelings such as, e.g., the three following SIC labelings:

- $l_{deg} = (\mathbb{N}, \leq, deg)$ where $deg$ is the function that returns node degree;
- $l_{distance_k} = (\mathbb{N}, \leq, distance_k)$ where $distance_k$ is the function that returns the number of nodes that are reachable by a path of length smaller than $k$;
- $l_{clique_k} = (\mathbb{N}, \leq, clique_k)$ where $clique_k$ is the function that returns the number of cliques of size $k$ that contains the node.

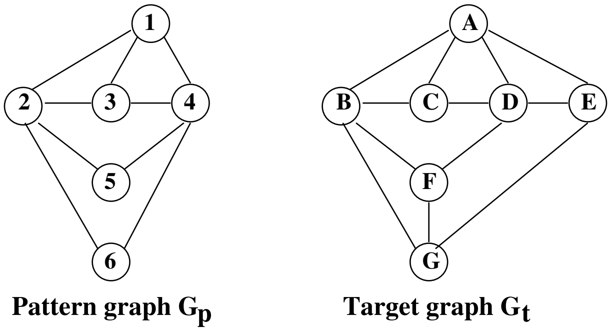

**Fig. 1.** Instance of subgraph isomorphism problem.

**Example.** Let us consider for example the subgraph isomorphism problem displayed in Fig. 1. Note that this instance has no solution as $G_p$ cannot be mapped into a subgraph of $G_t$. The labeling $l_{deg} = (\mathbb{N}, \leq, deg)$ assigns the following labels to nodes.

$$\begin{aligned} deg(A) = deg(B) = deg(D) = deg(2) = deg(4) &= 4 \\ deg(C) = deg(E) = deg(F) = deg(G) = deg(1) = deg(3) &= 3 \\ deg(5) = deg(6) &= 2 \end{aligned}$$

Hence, the set of compatible couples induced by this labeling is

$$\begin{aligned} CC_{l_{deg}} &= \{(u, v) \mid u \in \{2, 4\}, v \in \{A, B, D\}\} \\ &\cup \{(u, v) \mid u \in \{1, 3, 5, 6\}, v \in \{A, B, C, D, E, F, G\}\} \end{aligned}$$

This set of compatible couples allows one to remove values $C$, $E$, $F$ and $G$ from the domains of the variables associated with nodes 2 and 4.

### 3.2 Strengthening a Labeling

We propose to start from an elementary SIC labeling that is easy to compute such as the $l_{deg}$ labeling defined above, and to iteratively strengthen this labeling. The strength of labelings is defined with respect to the induced compatible couples as follows.

**Definition 4.** *Let $l$ and $l'$ be two labelings. $l'$* is strictly stronger than *$l$ iff $CC_{l'} \subset CC_l$, and $l'$* is equivalent to *$l$ iff $CC_{l'} = CC_l$.*

A stronger labeling yields a better filtering, as it contains less compatible couples.

To strengthen a labeling, the idea is to extend the label of a node by adding information from the labels of its neighbors. This information is a multiset (as several neighbors may have the same label). We shall use the following notations for multisets.

**Definition 5.** *Given an underlying set $A$, a* multiset *is a function $m : A \rightarrow \mathbb{N}$, such that $m(a)$ is the multiplicity (i.e., the number of occurrences) of $a$ in $m$. The multiset $m$ can also be represented by the bag $\{a_0, \ldots, a_0, a_1, \ldots\}$ where elements are repeated according to their multiplicity.*

For example, the multiset $m$ that contains 2 occurrences of $a$, 3 occurrences of $b$, and 1 occurrence of $c$ is defined by $m(a)$=2, $m(b)$=3, $m(c)$=1, and $\forall x \notin \{a, b, c\}, m(x)$=0. This multiset may also be represented by $\{a, a, b, b, b, c\}$.

Given a partial order on a set $A$, we extend the partial order to multisets over $A$ as follows.

**Definition 6.** *Given two multisets $m$ and $m'$ over a set $A$, and a partial order $\preceq \subseteq A \times A$, we define $m \preceq m'$ iff there exists a total injective mapping $t : m \rightarrow m'$ such that $\forall a_i \in m, a_i \preceq t(a_i)$.*

In other words, $m \preceq m'$ iff for every element of $m$ there exists a different element of $m'$ which is greater or equal. For example, if we consider the classical ordering on $\mathbb{N}$, we have $\{3, 3, 4\} \preceq \{2, 3, 5, 5\}$, but $\{3, 3, 4\}$ is not comparable with $\{2, 5, 6\}$. Note that comparing two multisets is not trivial in the general case, especially if the order relation on the underlying set $A$ is not total. This point will be handled in the next section.

We now define the labeling extension procedure.

**Definition 7.** *Given a labeling $l = (\mathbb{L}, \preceq, \alpha)$, the* neighborhood extension *of $l$ is the labeling $l' = (\mathbb{L}', \preceq', \alpha')$ such that:*

- *every label of $\mathbb{L}'$ is composed of a label of $\mathbb{L}$ and a multiset of labels of $\mathbb{L}$, i.e., $\mathbb{L}' = \mathbb{L} \cdot (\mathbb{L} \rightarrow \mathbb{N})$;*
- *the labeling function $\alpha'$ extends every label $\alpha(v)$ by the multiset of the labels of the neighbors of $v$, i.e., $\alpha'(v) = \alpha(v) \cdot m$ where $\forall l_i \in \mathbb{L}$,*
  *$m(l_i) = \#\{u \mid (u, v) \in Edge \wedge \alpha(u) = l_i\}$;*
- *the partial order on the extended labels of $\mathbb{L}'$ is defined by $l_1 \cdot m_1 \preceq' l_2 \cdot m_2$ iff $l_1 \preceq l_2$ and $m_1 \preceq m_2$.*

The next theorem states that the neighborhood extension of a SIC labeling is a stronger (or equal) SIC labeling.

**Theorem 1.** *Let $l = (\mathbb{L}, \preceq, \alpha)$ be a labeling, and $l' = (\mathbb{L}', \preceq', \alpha')$ be its neighborhood extension. If $l$ is an SIC labeling, then (i) $l'$ is also SIC, and (ii) $l'$ is stronger than or equal to $l$.*

*Proof.* (i): Let $f$ be a subgraph matching function and $v \in N_p$. We show that $\alpha'(v) \preceq' \alpha'(f(v))$, that is $\alpha(v) \preceq \alpha(f(v))$ and $m \preceq m'$, with $m$ (resp. $m'$) the multiset of the labels of the neighbors of $v$ in $G_p$ (resp. of $f(v)$ in $G_t$):

– $\alpha(v) \preceq \alpha(f(v))$ because $l$ is SIC;
– $m \preceq m'$ because $m'$ contains, for each neighbor $u$ of $v$, the label $\alpha(f(u))$ of the node matched to $u$ by the subgraph matching function $f$; as $l$ is SIC, $\alpha(u) \preceq \alpha(f(u))$, and thus $m \preceq m'$.

(ii) : This is a direct consequence of the partial order on the extended labels in $\mathbb{L}'$ (Definition 7) : $\alpha(u) \preceq \alpha(v)$ is one of the conditions to have $\alpha'(u) \preceq \alpha'(v)$. □

**Example.** Let us consider again the subgraph isomorphism problem displayed in Fig. 1, and the labeling $l_{deg} = (\mathbb{N}, \leq, deg)$ defined in 3.1. The neighborhood extension of $l_{deg}$ is the labeling $l' = (\mathbb{L}', \preceq', \alpha')$ displayed below. Note that we only display compatibility relationships $l_i \preceq l_j$ such that $l_i$ is the label of a node of the pattern graph and $l_j$ is the label of a node of the target graph as other relations are useless for filtering purposes.

$$\begin{aligned}
\alpha'(A) &= 4 \cdot \{3,3,4,4\} \\
\alpha'(B) = \alpha'(D) &= 4 \cdot \{3,3,3,4\} \\
\alpha'(2) = \alpha'(4) &= 4 \cdot \{2,2,3,3\} \preceq' 4 \cdot \{3,3,4,4\} \text{ and } 4 \cdot \{3,3,3,4\} \\
\alpha'(C) &= 3 \cdot \{4,4,4\} \\
\alpha'(E) = \alpha'(F) &= 3 \cdot \{3,4,4\} \quad \preceq' 4 \cdot \{3,3,4,4\}, 3 \cdot \{4,4,4\} \text{ and } 3 \cdot \{3,4,4\} \\
\alpha'(1) = \alpha'(3) &= 3 \cdot \{3,4,4\} \quad \preceq' 4 \cdot \{3,3,4,4\}, 3 \cdot \{4,4,4\} \text{ and } 3 \cdot \{3,4,4\} \\
\alpha'(G) &= 3 \cdot \{3,3,4\} \\
\alpha'(5) = \alpha'(6) &= 2 \cdot \{4,4\} \qquad \preceq' 4 \cdot \{3,3,4,4\}, 3 \cdot \{4,4,4\} \text{ and } 3 \cdot \{3,4,4\}
\end{aligned}$$

Hence, the set of compatible couples induced by this extended labeling is

$$\begin{aligned}
CC_{l'} = &\{(u,v) \mid u \in \{2,4\}, v \in \{A,B,D\}\} \\
\cup\ &\{(u,v) \mid u \in \{1,3,5,6\}, v \in \{A,C,E,F\}\}
\end{aligned}$$

As compared to the initial labeling $l_{deg}$, this set of compatible couples allows one to further remove values $B$, $D$ and $G$ from the domains of the variables associated with nodes 1, 3, 5 and 6.

### 3.3 Iterative Labeling Strengthening

The strengthening of a labeling described in the previous section can be repeated by relabeling nodes iteratively, starting from a given SIC labeling $l$.

**Definition 8.** *Let $l = (\mathbb{L}, \preceq, \alpha)$ be an initial SIC labeling. We define the sequence of SIC labelings $l^i = (\mathbb{L}^i, \preceq^i, \alpha^i)$ such that $l^0 = l$ and $l^{i+1}$ = neighborhood extension of $l^i$ $(i \geq 0)$.*

A theoretical filter can be built on this sequence. Starting from an initial SIC labeling function $l = l^0$, we iteratively compute $l^{i+1}$ from $l^i$ and filter domains with respect to the set of compatible couples induced by $l^i$ until either a domain becomes empty (thus indicating that the problem has no solution) or reaching some termination condition.

A termination condition is to stop iterating when the sequence reaches a fixpoint, *i.e.*, a step where any further relabeling cannot change the strength of the labeling. We show in [14] that such a fixpoint is reached in $O(n_p \cdot n_t)$ steps, when both the set of compatible couples and the number of different labels are not changed between two consecutive steps.

**Example.** Let us consider again the subgraph isomorphism problem displayed in Fig. 1, and let us suppose that the sequence of SIC labelings is started from $l^0 = l_{deg} = (\mathbb{N}, \leq, deg)$ as defined in 3.1. After the first iteration, the neighborhood extension $l^1$ of $l^0$ is the labeling displayed in the example of section 3.2. To improve reading, we rename these labels as follows:

$$\begin{array}{rl}
\alpha^1(A) = 4 \cdot \{3,3,4,4\} & \text{renamed } m_1 \\
\alpha^1(B) = \alpha^1(D) = 4 \cdot \{3,3,3,4\} & \text{renamed } m_2 \\
\alpha^1(2) = \alpha^1(4) = 4 \cdot \{2,2,3,3\} & \text{renamed } m_3 \preceq^1 \{m_1, m_2\} \\
\alpha^1(C) = 3 \cdot \{4,4,4\} & \text{renamed } m_4 \\
\alpha^1(E) = \alpha^1(F) = \alpha^1(1) = \alpha^1(3) = 3 \cdot \{3,4,4\} & \text{renamed } m_5 \preceq^1 \{m_1, m_4\} \\
\alpha^1(G) = 3 \cdot \{3,3,4\} & \text{renamed } m_6 \\
\alpha^1(5) = \alpha^1(6) = 2 \cdot \{4,4\} & \text{renamed } m_7 \preceq^1 \{m_1, m_4, m_5\}
\end{array}$$

From labeling $l^1$, we compute the following extended labels and partial order:

$$\begin{array}{rll}
\alpha^2(A) = m_1 \cdot \{m_2, m_2, m_4, m_5\} & \text{renamed } n_1 & \\
\alpha^2(B) = m_2 \cdot \{m_1, m_4, m_5, m_6\} & \text{renamed } n_2 & \\
\alpha^2(C) = m_4 \cdot \{m_1, m_2, m_2\} & \text{renamed } n_3 & \\
\alpha^2(D) = m_2 \cdot \{m_1, m_4, m_5, m_5\} & \text{renamed } n_4 & \\
\alpha^2(E) = m_5 \cdot \{m_1, m_2, m_6\} & \text{renamed } n_5 & \\
\alpha^2(F) = m_5 \cdot \{m_2, m_2, m_6\} & \text{renamed } n_6 & \\
\alpha^2(G) = m_6 \cdot \{m_2, m_5, m_5\} & \text{renamed } n_7 & \\
\alpha^2(1) = \alpha^2(3) = m_5 \cdot \{m_3, m_3, m_5\} & \text{renamed } n_8 & \preceq^2 \{n_1, n_3\} \\
\alpha^2(2) = \alpha^2(4) = m_3 \cdot \{m_5, m_5, m_7, m_7\} & \text{renamed } n_9 & \preceq^2 \{n_4\} \\
\alpha^2(5) = \alpha^2(6) = m_7 \cdot \{m_3, m_3\} & \text{renamed } n_{10} & \preceq^2 \{n_1, n_3, n_5, n_6\}
\end{array}$$

The set of compatible couples induced by this labeling is

$$\begin{aligned}
CC_{l^2} = \{&(2,D),(4,D),(1,A),(1,C),(3,A),(3,C)\} \\
\cup\ \{&(u,v) \mid u \in \{5,6\}, v \in \{A,C,E,F\}\}
\end{aligned}$$

This set of compatible couples allows one to remove values $A$ and $B$ from the domains of the variables associated with nodes 2 and 4. Hence, the domains of these two variables only contain one value ($D$), and thanks to the *alldiff* constraint on the variables, an inconsistency is detected.

```
Algorithm 1: Filtering procedure
   Input: two graphs $G_p = (N_p, E_p)$ and $G_t = (N_t, E_t)$ such that $N_p \cap N_t = \emptyset$,
          an initial SIC labeling $l^0 = (\mathbb{L}^0, \alpha^0, \preceq^0)$,
          initial domains $D : N_p \rightarrow \mathcal{P}(N_t)$,
          a limit $k$ on the number of iterations
   Output: filtered domains
 1 for every node $u \in N_p$ do: $D(u) \leftarrow D(u) \cap \{v \in N_t | \alpha^0(u) \preceq^0 \alpha^0(v)\}$
 2 $i \leftarrow 1$
 3 while $\forall u \in N_p, D(u) \neq \emptyset$ and $i \leq k$ and fixpoint not reached do
 4     for every node $u \in N_p \cup N_t$ do
 5         $m_u^i \leftarrow$ multiset containing an occurrence of $\alpha^{i-1}(v), \forall (u,v) \in E_p \cup E_t$
 6         $\alpha^i(u) \leftarrow \alpha^{i-1}(u) \cdot m_u^i$
 7     $\mathbb{L}^i \leftarrow \{\alpha^i(u) | u \in N_p \cup N_t\}$; rename labels in $\mathbb{L}^i$ and $\alpha^i$
 8     $\preceq^i \leftarrow \{(\alpha^i(u), \alpha^i(v)) \mid u \in N_p \ \wedge \ v \in D(u) \ \wedge \ test(m_u^i, m_v^i, \preceq^{i-1})\}$
 9     for every node $u \in N_p$ do: $D(u) \leftarrow D(u) \cap \{v \in N_t | \alpha^i(u) \preceq^i \alpha^i(v)\}$
10     $i \leftarrow i + 1$
11 return $D$
```

## 4 Practical Framework

Algorithm 1 describes the overall filtering procedure. Starting from an initial SIC labeling, that may be, e.g., $l_{deg}$, this procedure first filters domains with respect to this initial labeling (line 1) and then iteratively extends this labeling (lines 4–8) and filters domains with respect to the new extended labeling (line 9) until some domain becomes empty, or a maximum number of iterations have been performed, or a fixpoint is reached.

Labeling extension (lines 4–8) is decomposed into three steps:

- lines 4–6: $\alpha^i$ is computed from $\alpha^{i-1}$; this step is done in $\mathcal{O}(\#Edge)$;
- line 7: labels of $\mathbb{L}^i$ are renamed; this step is done in $\mathcal{O}(d \cdot \#Node)$;
- line 8: the partial order $\preceq^i$ is computed, i.e., for every couple of nodes $(u, v)$ such that $u$ is a node of the pattern graph and $v$ is a node of the target graph which was compatible with $u$ at step $i-1$, we test for the compatibility of the multisets $m_u^i$ and $m_v^i$ to determine if the labels of $u$ and $v$ are still compatible at step $i$. Testing the compatibility of two mulisets is not trivial. We show in 4.1 how to do this exactly in $\mathcal{O}(d^{5/2})$, so that line 8 has a time complexity of $\mathcal{O}(n_p \cdot n_t \cdot d^{5/2})$. We then show in 4.2 how to compute an order inducing a weaker filtering in $\mathcal{O}(n_t \cdot d \cdot (n_p + d_t \cdot \log n_t))$. These two variants are experimentally compared in Section 5.

The filtering step (line 9) is done in $\mathcal{O}(n_p \cdot n_t)$.

### 4.1 Exact computation of the partial order

Given two multisets $m_u$ and $m_v$, and a partial order $\preceq$, the function *test*$(m_u, m_v, \preceq)$ determines if $m_u \preceq m_v$, i.e., if there exists for each label occurrence in $m_u$ a distinct label occurrence in $m_v$ which is greater or equal according to $\preceq$.

*Property 1.* Let $G = (N = (N_u, N_v), E)$ be the bipartite graph such that $N_u$ (resp. $N_v$) associates a different node with every label occurrence in the multiset $m_u$ (resp. $m_v$), and $E$ contains the set of edges $(i, j)$ such that $i \preceq j$. We have $m_u \preceq m_v$ iff there exists a matching that covers $N_u$ in $G$.

Hopcroft [15] proposes an algorithm for solving this problem in $\mathcal{O}(|N_u| \cdot |N_v| \cdot \sqrt{|N|})$. As the sizes of $m_u$ and $m_v$ are bounded by the maximal degree $d$, the test function can be done in $O(d^{5/2})$.

### 4.2 Computation of an approximated order

If $\preceq$ is a total order, the function *test*$(m_u, m_v, \preceq)$ can be implemented more efficiently, by sorting each multiset and matching every label of $m_u$ with the smallest compatible label of $m_v$. In this case, the complexity of *test* is $\mathcal{O}(d \cdot \log d)$.

When $\preceq$ is not a total order, one may extend it into a total order $\leq$. This total order can then be used in the *test* function to determine if $m_u \leq m_v$. However, the total order introduces new label compatibilities so that *test*$(m_u, m_v, \leq)$ may return true while *test*$(m_u, m_v, \preceq)$ returns false. As a consequence, using this approximated order may induce a weaker filtering.

In this section, we first introduce the theoretical framework that defines a new neighborhood labeling extension based on a total order and proves its validity; then we show how it can be achieved in practice.

**Neighborhood labeling extension based on a total order.** The next definition gives a simple condition on the total order to ensure its consistency with respect to the partial order, i.e., to ensure that *test*$(m_u, m_v, \preceq) = \text{True} \Rightarrow$ *test*$(m_u, m_v, \leq) = \text{True}$.

**Definition 9.** *Let $l = (\mathbb{L}, \preceq, \alpha)$ be a labeling. A* consistent total order for $l$ *is a total order $\leq$ on $\mathbb{L}$ such that $\forall u \in n_p, \forall v \in n_t, \alpha(u) \preceq \alpha(v) \Rightarrow \alpha(u) \leq \alpha(v)$*

We extend the order $\leq$ on multisets like for partial orders in Definition 6, i.e., $m \leq m'$ iff there exists an injective function $t : m \rightarrow m'$ such that $\forall a_i \in m, a_i \leq t(a_i)$. Hence, $m \preceq m' \Rightarrow m \leq m'$. Let us note however that this extension of $\leq$ to multisets only induces a partial order on multisets as some multisets may not be comparable.

We can then define a new neighborhood extension procedure, based on a consistent total order.

**Definition 10.** *Let $l = (\mathbb{L}, \preceq, \alpha)$ be a labeling, and $\leq$ be a consistent total order for $l$. The* neighborhood extension of $l$ based on $\leq$ *is the labeling $l'_\leq = (\mathbb{L}', \preceq'_\leq, \alpha')$ where $\mathbb{L}'$ and $\alpha'$ are defined like in Definition 7, and the order relation $\preceq'_\leq \subseteq \mathbb{L}' \times \mathbb{L}'$ is defined by*

$$l_1 \cdot m_1 \preceq'_\leq l_2 \cdot m_2 \textit{ iff } l_1 \preceq l_2 \wedge m_1 \leq m_2$$

The next theorem shows that the neighborhood extension $l'_\leq$ based on $\leq$ may be used in our iterative labeling process, and that it is stronger or equal to $l$. However, it may be weaker than the neighborhood extension based on the partial order $\preceq$. Indeed, the total order induces more compatible couples of labels than the partial order.

**Theorem 2.** *Let $l = (\mathbb{L}, \preceq, \alpha)$, $l' = (\mathbb{L}', \preceq', \alpha')$, and $l'_\leq = (\mathbb{L}', \preceq'_\leq, \alpha')$, be three labelings such that $l'$ is the neighborhood extension of $l$ and $l'_\leq$ is the neighborhood extension of $l$ based on a consistent total order $\leq$.*

*If $l$ is an SIC labeling, then (i) $l'_\leq$ is SIC, (ii) $l'_\leq$ is stronger than (or equal to) $l$, and (iii) $l'$ is stronger than (or equal to) $l'_\leq$.*

*Proof.* (ii) and (iii): For labeling $l'$, we have $l_1 \cdot m_1 \preceq' l_2 \cdot m_2$ iff $l_1 \preceq l_2 \wedge m_1 \preceq m_2$. As $\leq$ is consistent w.r.t. $\preceq$, we have $m \preceq m' \Rightarrow m \leq m'$. Hence, $CC_{l'} \subseteq CC_{l'_\leq} \subseteq CC_l$. (i) is a direct consequence of (iii), as $l'$ is SIC (Theorem 1). □

Different consistent total orders may be derived from a given partial order, leading to prunings of different strength: the less new couples of compatible nodes are introduced by the total order, the better the filtering. However, we conjecture in [14] that finding the best consistent total order is NP-hard. Hence, we propose a heuristic algorithm that aims at computing a total order that introduces few new compatible couples without guarantee of optimality. Let us note $L_p$ (resp. $L_t$) the set of labels associated with nodes of the pattern graph $G_p$ (resp. target graph $G_t$). We shall suppose without loss of generality[1] that $L_p \cap L_t = \emptyset$. The idea is to sequence the labels of $L_p \cup L_t$, thus defining a total order on these labels, according to the following greedy principle: starting from an empty sequence, one iteratively adds some labels of $L_p \cup L_t$ at the end of the sequence, and removes these labels from $L_p$ and $L_t$, until $L_p \cup L_t = \emptyset$.

To choose the labels added in the sequence at each iteration, our heuristic is based on the fact that the new couples of compatible nodes are introduced by new couples of compatible labels $(e_p, e_t)$ such that $e_p \in L_p$ and $e_t \in L_t$. Hence, the goal is to sequence as late as possible the labels of $L_p$. To this aim, we first compute the set of labels $e_t \in L_t$ for which the number of labels $e_p \in L_p, e_p \preceq e_t$ is minimal. To break ties, we then choose a label $e_t$ such that the average number of labels $e'_t \in L_t, e_p \preceq e'_t$, for every label $e_p \in L_p, e_p \preceq e_t$, is minimal. Then, we introduce in the sequence the selected label $e_t$, preceded by every label $e_p \in L_p$ such that $e_p \preceq e_t$.

The time complexity of this heuristic algorithm is in $\mathcal{O}(n_t \cdot \log n_t \cdot d_p \cdot d_t)$.

[1] If a label $e$ both belongs to $L_p$ and $L_t$, it is always possible to rename $e$ into $e'$ in $L_t$ (where $e'$ is a new label), and to add a relation $e'' \preceq e'$ for every label $e'' \in L_p$ such that $e'' \preceq e$.

**Practical computation of an approximate partial order.** In practice, one has to compute a total order $\leq^{i-1}$ that approximates the partial order $\preceq^{i-1}$ at each iteration $i$ of Algorithm 1. This must be done between lines 7 and 8. Then each call to the test function, line 8, is performed with the total order $\leq^{i-1}$ instead of the partial order $\preceq^{i-1}$.

In this case, the time complexity of the computation of $\preceq^{i}$ (line 8) is in $\mathcal{O}(n_t \cdot n_p \cdot d \cdot \log d)$. This complexity can be reduced to $\mathrm{O}(n_t \cdot n_p \cdot d)$ by first sorting all the multisets. When adding the time complexity of the computation of the total order by our heuristic algorithm, we obtain an overall complexity in $\mathcal{O}(n_t \cdot d \cdot (n_p + d_t \cdot \log n_t))$.

### 4.3 Filtering within a Branch and Propagate framework

In this section, we introduce two optimizations that may be done when filtering is integrated within a branch and propagate search, where a variable assignment is done at each step of the search.

A first optimization provides an entailment condition for the filtering. If the initial labeling $l^0$ is such that the maximum label of the pattern graph is smaller or equal to the minimum label of the target graph, every label of nodes of the pattern graph is compatible with all the labels of nodes of the target graph so that no domain can be reduced by our filtering procedure.

A second optimization is done when, during the search, the variable associated with a pattern node is assigned to a target node. In this case, the neighborhood extension procedure is modified by forcing the two nodes to have a same new label which is not compatible with other labels as follows:

**Definition 11.** *Let $l = (\mathbb{L}, \preceq, \alpha)$ be an SIC labeling, and let $(u, v) \in N_p \times N_t$ such that $v \in x_u$. The propagation of $x_u = v$ on $l$ is the new labeling $l' = (\mathbb{L}', \preceq'$ $, \alpha')$ such that*

- *$\mathbb{L}' = \mathbb{L} \cup \{l_{uv}\}$ where $l_{uv}$ is a new label such that $l_{uv} \notin \mathbb{L}$;*
- *$\preceq' = \preceq \cup \{(l_{uv}, l_{uv})\}$ so that the new label $l_{uv}$ is not comparable with any other label except itself;*
- *$\alpha'(u) = \alpha'(v) = l_{uv}$ and $\forall w \in Nodes \setminus \{u, v\}, \alpha'(w) = \alpha(w)$*

This labeling $l'$ is used as a starting point of a new sequence of labeling extensions. Note that this propagation is done every time a domain is reduced to a singleton.

## 5 Experimental Results

**Considered instances.** We evaluate our approach on graphs that are randomly generated using a power law distribution of degrees $P(d = k) = k^{-\lambda}$: this distribution corresponds to scale-free networks which model a wide range of real networks, such as social, Internet, or neural networks [16]. We have made experiments with different values of $\lambda$, ranging between 1 and 5, and obtained similar results. Hence, we only report experiments on graphs generated with the standard value $\lambda = 2.5$.

We have considered 6 classes of instances, each class containing 20 different instances. For each instance, we first generate a connected target graph which node degrees are bounded between $d_{min}$ and $d_{max}$. Then, a connected pattern graph is extracted from the target graph by randomly selecting a percentage $p_n$ (resp. $p_e$) of nodes (resp. edges).

All instances of classes A, B, and C are non directed feasible instances that have been generated with $d_{min} = 5$, $d_{max} = 8$, and $p_n = p_e = 90\%$. Target graphs in A (resp. B and C) have 200 (resp. 600 and 1000) nodes.

All instances of class D are directed feasible instances that have been generated with $d_{min} = 5$, $d_{max} = 8$, and $p_n = p_e = 90\%$. Target graphs have 600 nodes. Edges of target graphs have been randomly directed. To solve these directed instances, the filtering procedure is adapted by extending labelings with two multisets that respectively contain labels of successors and predecessors.

All instances of classes E and F are non directed instances that have been generated with $d_{min} = 20$, $d_{max} = 300$, and $p_n = 90\%$. Target graphs have 300 nodes. Instances of class E are feasible ones that have been generated with $p_e = 90\%$. Instances of class F are non feasible ones: for these instances, pattern graphs are extracted from target graphs by randomly selecting 90% of nodes and 90% of edges, but after this extraction, 10% of new edges have been randomly added.

For all experimentations reported below, each run searches for all solutions of an instance.

**Comparison of different variants of our filtering algorithm.** Algorithm 1 has been implemented in Gecode (http://www.gecode.org), using CP(Graph) and CP(Map) [17, 18] which provide graph and function domain variables. The global subgraph isomorphism constraint has been combined with $c2$ constraints (as defined in Section 2.2) and a global AllDiff constraint which are propagated by forward checking.

Table 1 compares different variants of Algorithm 1, obtained by either computing an exact partial order or an approximated one (as described in 4.1 and 4.2), and by considering different limits $k$ on the number of iterations. In all variants, the initial labeling $l^0$ is the labeling $l_{deg}$ defined in 3.1. Note that the order of $l_{deg}$ is a total order so that in this case the exact and approximated variants are equivalent for $k = 1$.

Let us first compare the exact and approximated variants. The number of failed nodes with Approx./$k = 2$ is greater than Exact/$k = 2$, but it is smaller than with Exact/$k = 1$. This shows us that the total order computed by our heuristic algorithm is a quite good approximation of the partial order. When considering CPU-times, we note that Approx./$k = 2$ is significantly quicker than Exact/$k = 2$.

Table 1 also shows that the best performing variant differs when considering different classes of instances. Instances of class D are best solved when $k = 0$,

| | Solved instances (%) | | | | | | Average time | | | | | | Average failed nodes | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Exact | | | Approx. | | | Exact | | | Approx. | | | Exact | | | Approx. | | |
| | k=0 | k=1 | k=2 | k=2 | k=4 | k=8 | k=0 | k=1 | k=2 | k=2 | k=4 | k=8 | k=0 | k=1 | k=2 | k=2 | k=4 | k=8 |
| A | 100 | 100 | 100 | 100 | 100 | 100 | 2.2 | **0.6** | 23.4 | 1.3 | 1.9 | 3.2 | 440 | 14 | 0 | 13 | 0 | 0 |
| B | 100 | 100 | 100 | 100 | 100 | 100 | 61.4 | **5.6** | 144.2 | 24.5 | 28.7 | 59.6 | 1314 | 8 | 0 | 3 | 0 | 0 |
| C | 45 | 100 | 45 | 100 | 100 | 100 | 439.2 | **26.3** | 495.8 | 101.8 | 110.4 | 227.8 | 1750 | 13 | 0 | 2 | 0 | 0 |
| D | 100 | 100 | 100 | 100 | 100 | 100 | **0.7** | 2.6 | 99.6 | 7.5 | 24.7 | 56.3 | 2 | 0 | 0 | 0 | 0 | 0 |
| E | 80 | 60 | 0 | 75 | 80 | **85** | 126.7 | 98.6 | - | 35.2 | 18.8 | 36.7 | 4438 | 159 | - | 39 | 13 | 7 |
| F | 23 | 20 | 0 | 38 | 63 | **68** | 186.4 | 109.9 | - | 45.0 | 10.5 | 3.9 | 18958 | 3304 | - | 2323 | 481 | 107 |

**Table 1.** Comparison of different variants of Algorithm 1: Exact (resp. Approx.) refers to the implementation of *test* described in 4.1 (resp. 4.2); $k$ gives the maximum number of iterations. Each line successively reports the percentage of instances that have been solved within a CPU time limit of 600s on an Intel Xeon 3,06 Ghz with 2Go of RAM; the average run time for the solved instances; and the average number of failed nodes in the search tree for the solved instances.

i.e., with the simple $l_{deg}$ labeling: these instances are easy ones, as adding a direction on edges greatly narrows the search space. Instances of classes A, B and C are more difficult ones, as they are not directed; these instances are best solved when $k = 1$, i.e., after one iteration of the exact labelling extension. Instances of classes E and F, which have significantly higher node degrees, are very difficult ones. For these instances, and more particularly for those of class F which are not feasible ones and which appear to be even more difficult, iterative labeling extensions actually improve the solution process and the best results are obtained when $k = 8$.

As a conclusion, these experimentations show us that (1) the approximated variant offers a good compromise between filtering's strength and time, and (2) the optimal limit $k$ on the number of iterations depends on the difficulty of instances. The best average results are obtained with Approx./$k = 4$.

**Comparison with state-of-the-art approaches.** We now compare the variant Approx./k=4 of Algorithm 1 with a state-of-the-art algorithm coming from a C++ library called `vflib` [4], and with CP. We consider two different CP models:

- $c2$ is the model using $c2$ constraints described in Section 2.2;
- $c2 + c3$ is the model that additionnaly uses $c3$ constraints introduced in [8].

These two models are combined with a global Alldiff constraint. For $c2$ and Alldiff constraints, two levels of consistency are considered, i.e., Forward Checking (denoted by FC) and Arc Consistency (denoted by AC). Propagation of $c3$ follows [8]. All CP models have been implemented in Gecode using CP(Graph) and CP(Map).

Table 2 compares all these approaches and shows us that, except for easy instances of class D which are best solved by `vflib`, all other classes of instances are best solved by Approx./k=4. When comparing the different CP models, we note that adding redundant $c3$ constraints significantly improves the solution

| | Solved instances (%) | | | | | | Average time | | | | | | Average failed nodes | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | vflib | c2 | | c2+c3 | | App. | vflib | c2 | | c2+c3 | | App. | vflib | c2 | | c2+c3 | | App. |
| | | FC | AC | FC | AC | k=4 | | FC | AC | FC | AC | k=4 | | FC | AC | FC | AC | k=4 |
| A | 35 | 100 | 100 | 100 | 100 | 100 | 251.4 | 57.1 | 38.7 | 26.9 | 22.3 | **1.9** | - | 165239 | 19 | 67 | 0 | 0 |
| B | 0 | 0 | 0 | 0 | 0 | **100** | - | - | - | - | - | **28.7** | - | - | - | - | - | 0 |
| C | 0 | 0 | 0 | 0 | 0 | **100** | - | - | - | - | - | **110.4** | - | - | - | - | - | 0 |
| D | 100 | 100 | 100 | 5 | 0 | 100 | **0.8** | 7.9 | 81.7 | 542.7 | - | 24.7 | - | 2402 | 0 | 0 | 0 | 0 |
| E | 0 | 0 | 5 | 33 | 20 | **80** | - | - | 362.0 | 319.5 | 397.6 | **18.8** | - | - | 154 | 21 | 7 | 13 |
| F | 0 | 0 | 0 | 10 | 5 | **63** | - | - | - | 381.7 | 346.5 | **10.5** | - | - | - | 52 | 14 | 481 |

**Table 2.** Comparison of state-of-the-art approaches. Each line successively reports the percentage of instances that have been solved within a CPU time limit of 600s on an Intel Xeon 3,06 Ghz with 2Go of RAM; the average run time for the solved instances; and the average number of failed nodes in the search tree for the solved instances.

process except for the easy instances of class D which are better solved with simpler models.

## 6 Conclusion

We introduced a new filtering algorithm for the subgraph isomorphism problem that exploits the global structure of the graph in order to achieve a stronger partial consistency. This work extends a filtering algorithm for graph isomorphism [12] where a total order defined on some graph property labelling is strengthened until a fixpoint. The extension to subgraph isomorphism has been theorically founded. The order is partial and can also be iterated until a fixpoint. However, using such a partial order is ineffective. Instead, one can map this partial order to a total order. Performing such a mapping is hard, and can be efficiently approximated throught a heuristic algorithm. Experimental results show that our propagators are efficient against state-of-the-art propagators and algorithms.

Future work includes the development of dynamic termination criteria for the iterative labeling, the experimental study of other degree distributions, the analysis of alternative initial labelings.

## Acknowledgments

The authors want to thank the anonymous reviewers for the helpful comments. 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**(3) (2004) 265–298

2. Garey, M., Johnson, D.: Computers and Intractability. Freeman and Co., New York (1979)
3. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM **23**(1) (1976) 31–42
4. Cordella, L., Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen (2001) 149–159
5. Zampelli, S., Deville, Y., Dupont, P.: Approximate constrained subgraph matching. In: Principles and Pratice of Constraint Programming. Volume 3709 of Lecture Notes in Computer Science. (2005) 832–836
6. Régin, J.: Développement d'Outils Algorithmiques pour l'Intelligence Artificielle. Application à la Chimie Organique. PhD thesis (1995)
7. Rudolf, M.: Utilizing constraint satisfaction techniques for efficient graph pattern matching. In: Theory and Application of Graph Transformations. Number 1764 in Lecture Notes in Computer Science, Springer (1998) 238–252
8. Larrosa, J., Valiente, G.: Constraint satisfaction algorithms for graph pattern matching. Mathematical. Structures in Comp. Sci. **12**(4) (2002) 403–422
9. McKay, B.D.: Practical graph isomorphism. Congressus Numerantium **30** (1981) 45–87
10. P. T. Darga, M. H. Liffiton, K.A.S., Markov, I.L.: Exploiting structure in symmetry detection for cnf. In: Proc. Design Automation Conference (DAC), IEEE/ACM (2004) 530–534
11. Sorlin, S., Solnon, C.: A global constraint for graph isomorphism problems. In: 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2004). Volume 3011 of LNCS., Springer-Verlag (2004) 287–301
12. Sorlin, S., Solnon, C.: A new filtering algorithm for the graph isomorphism problem. 3rd International Workshop on Constraint Propagation and Implementation, CP2006 (2006)
13. 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
14. Zampelli, S., Deville, Y., Solnon, C., Sorlin, S., Dupont, P.: Filtering for subgraph matching. Technical Report INGIRR2007-03, Université Catholique de Louvain (2007)
15. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. **2**(4) (1973) 225–231
16. Barabasi, A.L.: Linked: How Everything Is Connected to Everything Else and What It Means. Plume (2003)
17. Dooms, G., Deville, Y., Dupont, P.: Cp(graph): Introducing a graph computation domain in constraint programming. In: Principles and Pratice of Constraint Programming. Volume 3709 of Lecture Notes in Computer Science. (2005) 211–225
18. Deville, Y., Dooms, G., Zampelli, S., Dupont, P.: Cp(graph+map) for approximate graph matching. 1st International Workshop on Constraint Programming Beyond Finite Integer Domains, CP2005 (2005) 33–48