# Solving subgraph isomorphism problems with constraint programming

Stéphane Zampelli, Yves Deville, Christine Solnon

**Noname manuscript No.**
(will be inserted by the editor)

# Solving Subgraph Isomorphism Problems with Constraint Programming

**Stéphane Zampelli · Yves Deville · Christine Solnon**

Received: date / Accepted: date

**Abstract** The 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 global constraint and an associated filtering algorithm to solve this problem within the context of constraint programming. The main idea of the filtering algorithm 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 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 constraint programming approaches.

**Keywords** Subgraph isomorphism · Global constraint · Filtering algorithm

## 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 [3]. This problem, known as subgraph isomorphism, is NP-complete [13] in the general case.

There exist dedicated algorithms for solving subgraph isomorphism problems, such as [25,5]. 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 [28].

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 CSPs in a

Ecole des Mines de Nantes, 4, rue Alfred Kastler. B.P. 20722, 44307 Nantes Cedex 3 (France), E-mail: stephane.zampelli@emn.fr · University of Louvain, Department of Computing Science and Engineering, Place Sainte-Barbe 2, 1348 Louvain-la-Neuve E-mail: yves.deville@uclouvain.be · Université de Lyon, Université Lyon 1, LIRIS, CNRS UMR5205, Nautibus, 43 Bd du 11 novembre, 69622 Villeurbanne cedex, France, E-mail: christine.solnon@liris.cnrs.fr

straightforward way [19,21]. To make CP competitive with dedicated approaches for these problems, [17] has introduced a filtering algorithm. [28] has extended this work to approximate subgraph isomorphism, and has shown that CP is competitive with dedicated approaches.

*Contribution and outline of the paper*

In this paper, we introduce a new global constraint and an associated filtering algorithm for the subgraph isomorphism problem. The filtering exploits the global structure of graphs to achieve a stronger partial consistency than when propagating edge constraints separately. This work takes inspiration from the partition refinement procedure used in [18,12,8] 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 [22,24] to define filtering algorithms for the graph isomorphism problem: the idea is to remove from the domain of a variable associated with 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 with a node $v$ every node the label of which is not compatible with the label of $v$.

This paper extends preliminary results presented in [29,27].

In Section 2, we introduce the subgraph isomorphism problem and we give an overview of existing approaches for solving this problem. The basic idea of a labeling-based filtering is described in Section 3: we first introduce the concept of labeling, and show how labelings can be used for filtering; then we show that labelings may be iteratively strengthened by adding information from labels of neighbors. The global constraint and a first filtering algorithm are described in Section 4. We introduce in Section 5 another filtering algorithm which ensures a lower consistency at a lower cost. Experimental results are described in Section 6.

## 2 Subgraph Isomorphism

### 2.1 Definitions and notations

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 couple of nodes. In this paper, we implicitly consider undirected graphs, such that $(u, v) \in E \Rightarrow (v, u) \in E$. The extension of our work to directed graphs, which is straightforward, is discussed in 3.4.

A *subgraph isomorphism problem* (SIP) 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 $f : N_p \rightarrow N_t$ that associates a different node of the target graph with every node of the pattern graph and that matches edges of the pattern graph, i.e., $\forall (u, v) \in N_p \times N_p, (u, v) \in E_p \Rightarrow (f(u), f(v)) \in E_t$. Note that the subgraph is not necessarily induced so that two pattern nodes that are not linked by an edge may be matched to target nodes which are linked by an edge. The problem is also called subgraph monomorphism problem or

subgraph matching in the literature. The function $f$ is called a *subgraph isomorphism 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 and we assume that $N_p \cap N_t = \emptyset$. We denote $\#S$ the cardinality of a set $S$. We also define $Node = N_p \cup N_t$, $Edge = E_p \cup E_t$, $n_p = \#N_p$, $n_t = \#N_t$, $n = \#Node$, $e = \#Edge$, $d_p$ and $d_t$ the maximal degrees of the graphs $G_p$ and $G_t$, and $d = max(d_p, d_t)$. The set of neighbors of a node $i$ is denoted $adj(i)$ and is defined by $adj(i) = \{j \mid (i,j) \in Edge\}$.

### 2.2 Dedicated approaches

The subgraph isomorphism problem is NP-complete. Ullmann's algorithm [25] is one of the first algorithm dedicated to the graph and subgraph isomorphism problem itself. It includes a preprocessing step based on node degrees and basically performs a depth-first search algorithm. The search space is pruned by maintaining arc consistency on edge constraints. Ullmann's algorithm has been the baseline for more advanced algorithms and it is still cited in recent applications (see for example [14]).

Another well known algorithm is `vflib` [6], which is considered as the state-of-the-art for subgraph isomorphism. The key points of `vflib` are the incremental building of connected graphs with cheap pruning rules and a clever choice of data structures leading to a lower memory footprint. `vflib` builds incrementally the assignment through a backtracking procedure, stopping whenever its extension is impossible. Assignments are extended by growing two connected graphs inside the pattern and the target graph respectively, and by checking conditions in the neighborhood of the current subgraphs. [26] also solves subgraph isomorphism with this approach. The popularity of `vflib` is due to several papers [6,4,5] demonstrating the superiority of `vflib` against Ullman's algorithm, the public availability of its source code, and the systematic experiments over a synthetic database for graph matching problems [11].

### 2.3 CP Models for Subgraph Isomorphism

A subgraph isomorphism problem may be formulated as a CSP in a straightforward way [19,21,17]. A variable $x_u$ is associated with every node $u$ of the pattern graph, and its domain $D(x_u)$ is the set of target nodes. A global *AllDiff* constraint [20] ensures that the matching function is injective. Edge matching is ensured by a set of $\#E_p$ binary constraints denoted by $c2$ and defined as follows:

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

*Example 1* Let us consider the subgraph isomorphism problem displayed in Fig. 1. The corresponding CSP is

$$\begin{aligned}
X &= \{x_1, x_2, x_3\} \\
D(x_i) &= \{a, b, c, d\}, \forall x_i \in X \\
C &= (x_1, x_2) \in \{(a,b),(b,a),(a,d),(d,a),(d,c),(c,d),(b,c),c,b)\} \wedge \\
&\quad (x_2, x_3) \in \{(a,b),(b,a),(a,d),(d,a),(d,c),(c,d),(b,c),c,b)\} \wedge \\
&\quad (x_3, x_1) \in \{(a,b),(b,a),(a,d),(d,a),(d,c),(c,d),(b,c),c,b)\} \wedge \\
&\quad \mathit{AllDiff}(\{x_1, x_2, x_3\})
\end{aligned}$$

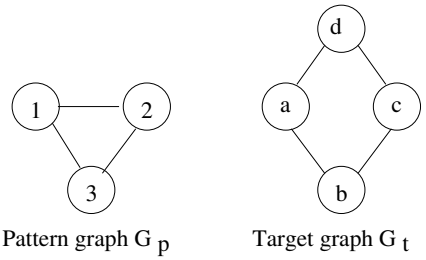



**Fig. 1** Instance of subgraph isomorphism problem.

Different levels of consistency may be considered for these constraints, i.e., forward checking (FC) or arc consistency (AC). For *AllDiff*, FC is performed on the binary decomposition of the constraint while AC is hyperarc consistency on the n-ary global constraint.

Larrosa and Valiente have proposed in [17] a filtering algorithm (called nRF+) dedicated to the subgraph isomorphism problem. The idea is to check, for every pattern node $i$ and every target node $a \in D(x_i)$, that all variables associated with adjacent nodes of $i$ can be assigned to different target nodes that are all adjacent to $a$. More precisely, they define the set

$$\mathcal{F}(i,a) = \emptyset \text{ if } \exists j \in adj(i), D(x_j) \cap adj(a) = \emptyset$$
$$\mathcal{F}(i,a) = \cup_{j \in adj(i)} (D(x_j) \cap adj(a)) \text{ otherwise}$$

and remove $a$ from $D(x_i)$ whenever $\#\mathcal{F}(i,a) < \#adj(i)$. Larrosa and Valiente have shown that this filtering algorithm combined with binary c2 constraints and a global *AllDiff* constraint allows CP to efficiently solve difficult instances.

*Example 2* Let us consider again the subgraph isomorphism problem displayed in Fig. 1, and let us suppose that the domains of $x_2$ and $x_3$ have been reduced to $\{a, b, c\}$. In this case, we have

$$\mathcal{F}(1,a) = (D(x_2) \cap adj(a)) \cup (D(x_3) \cap adj(a)) = \{b\}$$

As $\#adj(1) = 2$, $a$ is removed from $D(x_1)$. For the same reason, $c$ is removed from $D(x_1)$. Then, $b$ is removed from $D(x_2)$ and $D(x_3)$ as the neighbors of $b$ no longer belong to the domains of neighbors of 2 and 3.

## 3 Labeling for subgraph isomorphism

This section introduces a labeling process which is used in section 4 to define a filtering algorithm for a global constraint dedicated to the SIP.

Basically, this labeling process aims at filtering, for every pattern node $u$, the set of target nodes that may be matched with $u$ by a subgraph isomorphism function. To do that, the idea is to associate a label with every node and to define a partial order over these labels. This partial order defines node compatibility, i.e., a target node $u$ may be matched to a pattern node $v$ only if $(u, v)$ belongs to the partial order. Of course, this partial order must be consistent with respect to subgraph isomorphism, i.e., two nodes that may be matched by a subgraph isomorphism function must be compatible.

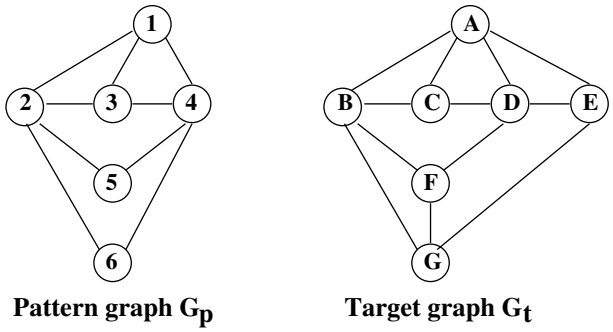



**Fig. 2** Instance of subgraph isomorphism problem.

We define such subgraph isomorphism consistent labelings in 3.1. We then show in 3.2 how a labeling may be strengthened by adding information from labels of neighbors. We show in 3.3 that this strengthening step may be iterated until some fixpoint is reached, thus defining a sequence of labelings the strength of which increases at each iteration. In sections 3.1 to 3.3, we always consider undirected graphs, such that $(u, v) \in E \Rightarrow (v, u) \in E$. The extension of our approach to directed graphs, which is straightforward, is discussed in 3.4.

### 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 with nodes;
- $\preceq \subseteq \mathbb{L} \times \mathbb{L}$ is a partial order on $\mathbb{L}$;
- $\alpha : Node \rightarrow \mathbb{L}$ 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 may 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 $(u, v) \notin CC_l$.

*Example 3* Let us consider the subgraph isomorphism problem displayed in Fig. 2. Note that this instance has no solution as $G_p$ cannot be mapped into a subgraph of $G_t$. Let us define the labeling $l_{deg} = (\mathbb{N}, \leq, deg)$ such that $deg$ is the function which returns node degree, i.e., $\forall u \in Nodes, deg(u) = \#adj(u)$. This labeling 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

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

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.

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 $u$ of the pattern graph may be matched to a node $v$ of the target graph by a subgraph isomorphism function, then the label of $u$ must be compatible with the label of $v$. This property is called subgraph isomorphism consistency.

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

In the context of graph isomorphism, such as in [18], 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.

### 3.2 Strengthening a Labeling

We propose to start from an elementary SIC labeling which is easy to compute such as the $l_{deg}$ labeling defined in Ex. 3, 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 $CC_{l'} \subset CC_l$. $l'$ *is equivalent to* $l$ iff $CC_{l'} = CC_l$.

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

To strengthen a labeling, the idea is to extend the label of a node by adding information from the label 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 $m : A \to \mathbb{N}$, 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.

*Example 4* The multiset $m$ which contains exactly 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 it to a partial order on 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 $t : m \to m'$ 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.

*Example 5* 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 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., $\mathbb{L}' = \mathbb{L} \cdot (\mathbb{L} \to \mathbb{N})$;
- 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 $m_1 \preceq m_2$.

*Example 6* Let us consider again the subgraph isomorphism problem displayed in Fig. 2, and the labeling $l_{deg} = (\mathbb{N}, \leq, deg)$ defined in Ex. 3. The neighborhood extension of $l_{deg}$ is the labeling $l' = (\mathbb{L}', \preceq', \alpha')$ such that

- labels of $\mathbb{L}'$ are composed of numbers from $\mathbb{N}$ followed by multisets over $\mathbb{N}$;
- the labeling function $\alpha'$ is defined as follows

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

- the partial order $\preceq'$ is such that

$$\begin{array}{lll}
m_3 \preceq' m_1 & m_5 \preceq' m_1 & m_7 \preceq' m_1\\
m_3 \preceq' m_2 & m_5 \preceq' m_4 & m_7 \preceq' m_4\\
 & m_5 \preceq' m_5 & m_7 \preceq' m_5
\end{array}$$

Note that we only display compatibility relationships $m_i \preceq m_j$ such that $m_i$ is the label of a node of the pattern graph and $m_j$ is the label of a node of the target graph as other relations are useless for filtering purposes.

The set of compatible couples induced by this extended labeling is

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

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.

The next theorem states that the neighborhood extension of a SIC labeling is a stronger (or equivalent) 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 equivalent to $l$.*

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

- $\alpha(v) \preceq \alpha(f(v))$ because $l$ is SIC;
- $m \preceq m'$ because $f$ induces an injection from $m$ to $m'$ that respects $\alpha$.

(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 $\alpha'(u) \preceq \alpha'(v)$. □

### 3.3 Iterative Labeling Strengthening

The strengthening of a labeling described in the previous subsection 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 $l^{i+1}$ 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. Theorem 2 shows that a fixpoint is reached when both the set of compatible couples and the number of different labels are not changed between two consecutive steps.

**Definition 9** Given a labeling $l = (\mathbb{L}, \preceq, \alpha)$, we note labels($l$) the set of labels associated with nodes, i.e., labels($l$) = $image(\alpha)$.

**Theorem 2** *Let $l = (\mathbb{L}, \preceq, \alpha)$ be a SIC labeling. The following properties hold:*

1. *each iteration can only remove compatible couples and increase the number of different labels, i.e.,*

$$\forall k \geq 0, CC_{l^{k+1}} \subseteq CC_{l^k} \text{ and } \#labels(l^{k+1}) \geq \#labels(l^k)$$

2. *if at a given iteration $k$ the set of compatible couples and the number of labels are not changed, then they will not change at any further iteration, i.e.,*

$$\forall k \geq 0, \text{ if } CC_{l^{k+1}} = CC_{l^k} \text{ and } \#labels(l^{k+1}) = \#labels(l^k),$$
$$\text{then } \forall j > k, CC_{l^j} = CC_{l^k} \text{ and } \#labels(l^j) = \#labels(l^k)$$

3. *the fixpoint is reached in at most $n_p + n_t + n_p.n_t$ steps, i.e.,*

$$\exists k \leq n_p + n_t + n_p.n_t, CC_{l^{k+1}} = CC_{l^k} \text{ and } \#labels(l^{k+1}) = \#labels(l^k)$$

*Proof* (1) The inclusion is a direct consequence of property (ii) in Theorem 1. For the cardinality of the labels, by Definition 7, we have $\alpha^{k+1}(v) = \alpha^k(v).m$, with $m$ some multiset. Hence $\alpha^{k+1}(u) = \alpha^{k+1}(v)$ only if $\alpha^k(u) = \alpha^k(v)$.

(2) The set of labels at step $k$ defines a partition $P_k$ of $N_p \cup N_t$ (such that two nodes belong to a same part if they have the same label). The relation between a set $S_1$ of $P_k$ and a set $S_2$ of $P_{k+1}$ must be either $S_1 \cap S_2 = \emptyset$ or $S_2 \subseteq S_1$. This follows from Definition 7: $\alpha^{k+1}(v) = \alpha^k(v).m$ implies that a node $v$ must be in a partition $\alpha^{k+1}(v)$ finer than the partition $\alpha^k(v)$. Hence, when $\#labels(l^k) = \#labels(l^{k+1})$, the underlying partitions are the same for all subsequent steps. In particular the number of labels will never change. It remains to show that
$\forall\ (u, v) \in N_p \times N_t$
if $(\alpha^k(u) \preceq^k \alpha^k(v)) \Leftrightarrow (\alpha^{k+1}(u) \preceq^{k+1} \alpha^{k+1}(v))$
and $(\alpha^k(u) = \alpha^k(v)) \Leftrightarrow (\alpha^{k+1}(u) = \alpha^{k+1}(v))$
then $(\alpha^{k+1}(u) \preceq^{k+1} \alpha^{k+1}(v)) \Leftrightarrow (\alpha^{k+2}(u) \preceq^{k+2} \alpha^{k+2}(v))$.
We know that the underlying partitions and the set of compatible couples are the same for step $k$ and step $k+1$. This means that there exists an isomorphism preserving $\preceq$ between $labels(l^k)$ and $labels(l^{k+1})$.

(3) is a consequence of property (1) of Theorem 2; the fixpoint is reached in at most $\#labels(l) + \#CC_l$ steps, that is $(n_p + n_t) + n_p.n_t$ steps. □

*Example 7* Let us consider again the subgraph isomorphism problem displayed in Fig. 2, and let us suppose that the sequence of SIC labelings is started from $l^0 = l_{deg} = (\mathbb{N}, \leq, deg)$ as defined in Ex. 3. After the first iteration, the neighborhood extension $l^1$ of $l^0$ is the labeling displayed in Ex. 6. From labeling $l^1$, we compute the following extended labels and partial order:

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

The set of compatible couples induced by this labeling is

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

From labeling $l^2$, the following extended labels and partial order are computed:

$$\begin{array}{rll}
\alpha^3(A) = n_1 \cdot \{\{n_2,n_3,n_4,n_5\}\} & \text{renamed } o_1 & \\
\alpha^3(B) = n_2 \cdot \{\{n_1,n_3,n_6,n_7\}\} & \text{renamed } o_2 & \\
\alpha^3(C) = n_3 \cdot \{\{n_1,n_2,n_4\}\} & \text{renamed } o_3 & \\
\alpha^3(D) = n_4 \cdot \{\{n_1,n_3,n_5,n_6\}\} & \text{renamed } o_4 & \\
\alpha^3(E) = n_5 \cdot \{\{n_1,n_4,n_7\}\} & \text{renamed } o_5 & \\
\alpha^3(F) = n_6 \cdot \{\{n_2,n_4,n_7\}\} & \text{renamed } o_6 & \\
\alpha^3(G) = n_7 \cdot \{\{n_2,n_5,n_6\}\} & \text{renamed } o_7 & \\
\alpha^3(1) = \alpha^3(3) = n_8 \cdot \{\{n_8,n_9,n_9\}\} & \text{renamed } o_8 & \preceq^3 \emptyset \\
\alpha^3(2) = \alpha^3(4) = n_9 \cdot \{\{n_8,n_8,n_{10},n_{10}\}\} & \text{renamed } o_9 & \preceq^3 \{o_4\} \\
\alpha^3(5) = \alpha^3(6) = n_{10} \cdot \{\{n_9,n_9\}\} & \text{renamed } o_{10} & \preceq^3 \emptyset
\end{array}$$

The set of compatible couples induced by this labeling is

$$CC_{l^3} = \{(2,D),(4,D)\}$$

As no node is compatible with nodes 1, 3, 5 and 6, the domains of the associated variables become empty and an inconsistency is detected.

### 3.4 Extension to directed graphs

In directed graphs, the set of neighbors of a node $u$ is partitioned into successors and predecessors. The iterative labeling strengthening procedure may be extended to directed graphs in a very straightforward way, by computing two different multisets that respectively contain the labels of successors and predecessors, instead of one multiset that contains the labels of neighbors. More precisely, in the case of directed graphs, we define the neighborhood extension of a labeling as follows.

**Definition 10** 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 two multisets of labels of $\mathbb{L}$, i.e., $\mathbb{L}' = \mathbb{L} \cdot (\mathbb{L} \to \mathbb{N}) \cdot (\mathbb{L} \to \mathbb{N})$;
- the labeling function $\alpha'$ extends every label $\alpha(v)$ by the two multisets of the labels of the predecessors and successors of $v$ respectively, i.e., $\alpha'(v) = \alpha(v) \cdot m_{pred} \cdot m_{succ}$ where $\forall l_i \in \mathbb{L}$,
  $m_{pred}(l_i) = \#\{u \mid (u,v) \in Edge \wedge \alpha(u) = l_i\}$, and
  $m_{succ}(l_i) = \#\{u \mid (v,u) \in Edge \wedge \alpha(u) = l_i\}$;
- the partial order on the extended labels of $\mathbb{L}'$ is defined by $l_1 \cdot m_{pred_1} \cdot m_{succ_1} \preceq' l_2 \cdot m_{pred_2} \cdot m_{succ_2}$ iff $l_1 \preceq l_2$, $m_{pred_1} \preceq m_{pred_2}$, and $m_{succ_1} \preceq m_{succ_2}$.

This neighborhood extension procedure may be iterated until a fixpoint is reached exactly the same way as for undirected graphs.

## 4 ILF filtering algorithm

The labeling process introduced in the previous section can be used to define a filtering algorithm for a global constraint dedicated to subgraph isomorphism. Hence, we introduce in 4.1 a global constraint for the subgraph isomorphism problem. Then, we describe in 4.2 an associated filtering algorithm called ILF. We compare in 4.3 the achieved level of consistency with the ones obtained by forward checking and arc consistency on the standard CP model based on *AllDiff* and $c2$ constraints and the filtering procedure nRF+ described in 2.3.

### 4.1 Definition of the SIP global constraint

The SIP global constraint is defined by the relation $sip(G_p, G_t, Lvar)$ where $G_p = (N_p, E_p)$ and $G_t = (N_t, E_t)$ are two graphs and $Lvar$ is a set of couples which associates a different variable of the CSP with each different node of $N_p$, *i.e.*, $Lvar$ is a set of $\#N_p$ couples of the form $(x_u, u)$ where $x_u$ is a variable of the CSP and $u$ is a node of $N_p$.

Semantically, the global constraint $sip(G_p, G_t, Lvar)$ is satisfied by an assignment of the variables of $Lvar$ if there exists a subgraph isomorphism function $f : N_p \rightarrow N_t$ such that, $\forall (x_u, u) \in Lvar$, $x_u = f(u)$.

This global constraint is not semantically global according to [2] as it may be represented by a semantically equivalent set of *AllDiff* and $c2$ constraints as described in 2.3. However, the *sip* global constraint allows us to exploit the global semantic of SIPs to filter variable domains more efficiently.

### 4.2 Description of the ILF filtering procedure

The filtering procedure associated with the global *sip* constraint is called *Iterative Labeling Filtering* (ILF) and is described in Algorithm 1. This algorithm takes in input a global SIP constraint and initial variable domains; it is also parameterized by an initial labeling $l^0$ and the number $k$ of labeling extension iterations. The different steps of this algorithm are described in sections 4.2.1 to 4.2.4. Its correctness and complexity are discussed in 4.2.5 and 4.2.6.

#### *4.2.1 Filtering of $D$ w.r.t. a labeling $l^i$ (line 3)*

The successively computed labelings $l^i$ are used to filter domains by removing nodes which have incompatible labels. More precisely, for every couple $(x_u, u) \in Lvar$ we remove from $D(x_u)$ every value $v$ such that $\alpha^i(u) \not\preceq^i \alpha^i(v)$.

This step is done in $\mathcal{O}(n_p \cdot n_t)$.

#### *4.2.2 Reduction of the target graph w.r.t. domains (line 5)*

If a target node $u$ does not belong to any domain, then this node and its incident edges are discarded from the target graph as no pattern node may be matched with $u$. More precisely, we define $N_t \downarrow D = N_t \cap (\cup_{(x_u,u)\in Lvar} D(x_u))$. Each time domains

**Algorithm 1**: ILF($sip(G_p, G_t, Lvar), D, l^0, k$)

**Input**:
a global SIP constraint $sip(G_p, G_t, Lvar)$
the initial domain $D(x_u)$ of every variable $x_u$ such that $(x_u, u) \in Lvar$
an initial SIC labeling $l^0 = (L^0, \preceq^0, \alpha^0)$
a limit $k$ on the number of iterations
**Output**: filtered domains

1 $i \leftarrow 0$
2 **loop**
3 | filter $D$ w.r.t. $l^i$
4 **exit when** $\exists (x_u, u) \in Lvar, D(x_u) = \emptyset$ **or** $i = k$ **or** *fixpoint reached*
5 | reduce $G_t$ w.r.t. $D$
6 | strengthen $l^i$ w.r.t. singleton domains
| `/* Computation of the neighborhood extension` $l^{i+1}$ `of` $l^i$ `*/`
7 | **for** *every node* $u \in N_p \cup N_t$ **do**
8 | | $m_u \leftarrow \{\{\alpha^i(v) \mid (u, v) \in E_p \cup E_t\}\}$
9 | | $\alpha^{i+1}(u) \leftarrow \alpha^i(u) \cdot m_u$
10 | $\mathbb{L}^{i+1} \leftarrow \{\alpha^{i+1}(u) | u \in N_p \cup N_t\}$
11 | rename labels in $\mathbb{L}^{i+1}$ and $\alpha^{i+1}$
12 | $\preceq^{i+1} \leftarrow \{(\alpha^{i+1}(u), \alpha^{i+1}(v)) \mid (x_u, u) \in Lvar \ \wedge \ v \in D(x_u) \ \wedge \ test(m_u, m_v, \preceq^i)\}$
13 | $i \leftarrow i + 1$
14 **return** $D$

have been reduced, the target graph $G_t$ is reduced to the subgraph of $G_t$ induced by $N_t \downarrow D$.

Note that nodes and edges that are discarded during filtering must be restored when some backtracking occurs.

This step is done in $\mathcal{O}(n_t \cdot d)$ where $d$ is the maximal node degree.

#### *4.2.3 Strengthening of a labeling $l^i$ w.r.t. singleton domains (line 6)*

If a domain $D(x_v)$ is reduced to a singleton $\{u\}$, then nodes $u$ and $v$ are labeled with a new label $l_{uv}$ which is not compatible with any other label, except itself, thus preventing other pattern nodes from being matched with $u$. More precisely, we add a new label $l_{uv}$ to $\mathbb{L}^i$, we assign $\alpha^i(u)$ and $\alpha^i(v)$ to $l_{uv}$, and we add $(l_{uv}, l_{uv})$ to $\preceq^i$.

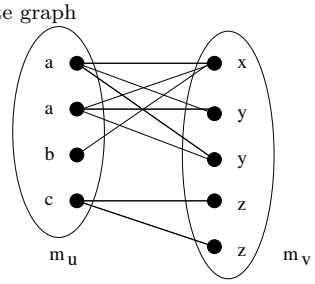
This step is done in $\mathcal{O}(n_p)$.

#### *4.2.4 Computation of the neighborhood extension $l^{i+1}$ of $l^i$ (lines 7 to 12)*

The neighborhood extension step is composed of the following steps:

- lines 7–9: $\alpha^{i+1}$ is computed from $\alpha^i$.
  This step is done in $\mathcal{O}(e)$ as it basically involves collecting the labels of the neighbours of every node.
- lines 10–11: Labels of $\mathbb{L}^{i+1}$ are collected and renamed in order to keep their size in $\mathcal{O}(d)$.
  This step is done in $\mathcal{O}(d \cdot n)$ as there are at most $n$ labels the length of which is bounded by $d$. We use a hashtable to associate new names to labels: each time a label is computed, if it already belongs to the hashtable then the label is renamed consequently, otherwise a new name is created and the label is added to the hashtable.

Let $m_u = \{\{a, a, b, c\}\}$ and $m_v = \{\{x, y, y, z, z\}\}$ be two multisets to be compared, and let the partial order be $\preceq = \{(a, x), (a, y), (b, x), (c, z)\}$. To decide if $test(m_u, m_v, \preceq)$ is true or false, we build the following bipartite graph




and use Hopcroft algorithm to find a matching which covers $m_u$. On this example, $\{(a, y), (a, y), (b, x), (c, z)\}$ is a matching that covers $m_u$ so that $test(m_u, m_v, \preceq)$ returns true.

**Fig. 3** Modeling the compatibility test problem as a matching problem.

– line 12: The partial order $\preceq^{i+1}$ is computed.
For every couple of nodes $(u, v) \in N_p \times N_t$ such that $v$ was compatible with $u$ at step $i$, we call the function $test(m_u, m_v, \preceq^i)$ to determine if $m_u \preceq^i m_v$, 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 $\preceq^i$. As different nodes may have the same label, we call the test function only when encountering a couple $(m_u, m_v)$ for the first time, and we memorize the result in a table so that we can reuse it if the same couple is encountered later.
To implement the *test* function, we have used the matching algorithm of Hopcroft [16]. Indeed, $m_u \preceq^i m_v$ iff there exists a matching that covers $m_u$ in the bipartite graph $G = ((m_u, m_v), E)$ such that $E = \{(x, y) \in m_u \times m_v \mid x \preceq^i y\}$, i.e., a label occurrence $x$ of $m_u$ is linked by an edge with a label occurrence $y$ of $m_v$ iff $x \preceq^i y$ (see example in Fig. 3). The complexity of this algorithm is in $\mathcal{O}(\#m_u \cdot \#m_v \cdot \sqrt{\#m_u + \#m_v})$. As the sizes of $m_u$ and $m_v$ are bounded by the maximal degree $d$, the *test* function can be executed in $O(d^{5/2})$.
Hence, the computation of the partial order is done in $\mathcal{O}(n_p \cdot n_t \cdot d^{5/2})$.

#### *4.2.5 Correctness of the filtering procedure*

*Property 1* Let $l^0$ be an SIC labeling and $k \in \mathbb{N}$. The ILF($sip(G_p, G_t, Lvar), D, l^0, k$) procedure is correct, i.e., for every subgraph isomorphism function $f$ from $N_p$ to $N_t$ such that $\forall (x_u, u) \in Lvar$, $f(u) \in D(x_u)$, ILF does not remove $f(u)$ from $D(x_u)$.

*Proof.* Indeed, we have shown in section 3 that the iterative labeling strengthening procedure is correct, i.e., if it is started from an SIC labeling $l^0$, then every successive $l^i$ labeling is also an SIC labeling that can be used to filter domains without removing solutions. Our filtering procedure is based on this iterative labeling strengthening procedure, but it also integrates information brought by variable domains to filter more values:

– the target graph $G_t$ is reduced to the subgraph induced by variable domains;
– labelings are strengthened with respect to singleton domains.

It is clear that this extra filtering does not remove values that may belong to a solution.

#### *4.2.6 Complexity of the filtering procedure*

At each iteration, the most expensive step is the computation of the partial order during the neighborhood extension procedure, which is done in $\mathcal{O}(n_p \cdot n_t \cdot d^{5/2})$. The procedure stops iterating either if the bound $k$ on the number of iterations is reached, or if a domain becomes empty, or if the fixpoint is reached. We have shown in 3.3 that the fixpoint is reached in $\mathcal{O}(n_p.n_t)$ iterations.

*Property 2* The ILF procedure has an overall time complexity of

$$\mathcal{O}(min(k, n_p \cdot n_t) \cdot n_p \cdot n_t \cdot d^{5/2})$$

We propose in Section 5 an approximated filtering procedure which ensures a lower partial consistency at a lower cost.

### 4.3 Level of consistency of ILF

The level of consistency achieved by the ILF filtering procedure described in the previous section depends on the initial labeling $l^0$ which is a parameter of this procedure. We first introduce and compare in 4.3.1 different SIC initial labelings that may be considered. We then compare the level of consistency ensured by ILF with forward checking (FC) and arc consistency (AC) on the standard SIP model and the filtering algorithm nRF+ introduced in section 2.3.

#### *4.3.1 Initial SIC labeling $l^0$*

The iterated neighborhood extension process is started from an initial SIC labeling $l^0$ which is a parameter. Different initial labelings may be considered, that may have different strength.

The weakest initial labeling that may be used is the labeling $l_\emptyset$ which associates the same label $\emptyset$ to every node, i.e., $l_\emptyset = (\mathbb{L}_\emptyset, \alpha_\emptyset, \preceq_\emptyset)$ such that

- $\mathbb{L}_\emptyset = \{\emptyset\}$,
- $\forall u \in Nodes, \alpha_\emptyset(u) = \emptyset$, and
- $\preceq_\emptyset = \{(\emptyset, \emptyset)\}$.

A stronger initial labeling is the $l_{deg}$ labeling defined in Ex. 3. Note however that, when $l^0 = l_\emptyset$, the labeling $l^1$ computed after the first iteration of ILF exactly corresponds to $l_{deg}$ as the label $\alpha^1(u)$ of every node $u$ is $\emptyset.m_u$ with $m_u(\emptyset) = deg(u)$. Hence, ILF$(sip, D, l_{deg}, k)$ achieves the same level of consistency as ILF$(sip, D, l_\emptyset, k+1)$.

One may define stronger initial labelings by integrating information brought by the domains. Let us define for example the labeling $l_{dom} = (\mathbb{L}_{dom}, \alpha_{dom}, \preceq_{dom})$ such that

- $\mathbb{L}_{dom} = \mathbb{N}$,
- $\alpha_{dom}$ assigns a different unique label to every node, and
- $\forall (u, v) \in N_p \times N_t, \alpha_{dom}(u) \preceq_{dom} \alpha_{dom}(v)$ iff $v \in D(x_u)$ and $deg(u) \leq deg(v)$.

When every variable domain contains all the target nodes, $l_{dom}$ is equivalent to $l_{deg}$. However, if some target nodes have been removed from some variable domains, then $l_{dom}$ is strictly stronger than $l_{deg}$ so that ILF$(sip, D, l_{dom}, k)$ will achieve a stronger consistency than ILF$(sip, D, l_{deg}, k)$.

Note however that $l_{dom}$ introduces many different labels and that $\alpha_{dom}$ is not a total order so that the neighborhood extension procedure may become expensive as most of the time is spent for comparing multisets and the complexity of this step depends on the number of different labels (see 4.2.4).

A good compromise to limit the number of different initial labels while strengthening the initial labeling with respect to information brought by the domains may be obtained with the labeling $l_{domCC} = (\mathbb{L}_{domCC}, \alpha_{domCC}, \preceq_{domCC})$ such that

- $\mathbb{L}_{domCC} = \mathbb{N}.\mathbb{N}$,
- $\forall u \in Nodes, \alpha_{domCC}(u) = deg(u).id(u)$ where $id(u)$ is defined as follows. Let $G = ((N_p, N_t), E)$ be the bipartite graph such that $E = \{(u, v) \in N_p \times N_t \mid v \in D(x_u)\}$. We compute the different connected components of $G$, and we associate a different id with every different connected component of $G$. We note $id(u)$ the id of the connected component which contains $u$.
- $\preceq_{domCC}$ is the set of couples $(deg_1.id_1, deg_2.id_2)$ such that $deg_1 \leq deg_2$ and $id_1 = id_2$.

$l_{domCC}$ is weaker than $l_{dom}$, but it introduces fewer labels. When the bipartite graph defined by domains is connected, $l_{domCC}$ is equivalent to $l_{deg}$. However, if it is not connected, then $l_{domCC}$ is strictly stronger than $l_{deg}$ so that ILF$(sip, D, l_{domCC}, k)$ will achieve a stronger consistency than ILF$(sip, D, l_{deg}, k)$.

### *4.3.2 Comparison with forward checking on c2 and* AllDiff

Our filtering procedure ILF achieves a level of consistency at least as strong as forward checking (FC) on the standard SIP model which combines a global *AllDiff* constraint with $c2$ constraints, provided that the number of strengthening iterations (defined by the parameter $k$) is greater or equal to 1, and whatever the initial labeling $l^0$ is. In this case, our filtering procedure can be used as a unique filter for SIP, thus offering a global constraint for SIP.

*Property 3* ILF(_,_,_,1) is as strong as $FC(AllDiff + c2)$.

*Proof* Each time a variable $x_u$ associated with a pattern node $u$ is assigned to a target node $a$, FC filters the domain of every variable $x_v$ with respect to $c2$ and *AllDiff* constraints as follows:

- For every pattern node $v \in adj(u)$, FC of $c2(x_u, x_v)$ removes any value $b \in D(x_v)$ such that $(a, b)$ is not an edge.
  ILF(_,_,_,1) also removes these values. Indeed, during the first iteration, the same unique label $l_{ua}$ is assigned to nodes $u$ and $a$ (line 6) before computing the extended labeling $l^1$. Hence, after the first iteration, we have $\alpha^1(v) \not\preceq^1 \alpha^1(b)$ for every node $b$ such that $(a, b)$ is not an edge. Indeed, $\alpha^1(v) = \alpha^0(v).m_v$ with $l_{ua} \in m_v$ (as $u \in adj(v)$) whereas $\alpha^1(b) = \alpha^0(b).m_b$ with $l_{ua} \notin m_b$ (as $a \notin adj(b)$) so that $m_v \not\preceq^0 m_b$ and therefore $\alpha^1(v) \not\preceq^1 \alpha^1(b)$.
- FC of *AllDiff* removes $a$ from the domains of every variable associated with a pattern node $v \neq u$.
  ILF(_,_,_,1) also removes these values. Indeed, because of the unique label $l_{ua}$ assigned to nodes $u$ and $a$, the label of every pattern node $v \neq u$ is not compatible with the label of $a$ so that $a$ is removed from $D(x_v)$ by ILF during the first iteration. ■

*4.3.3 Comparison with arc consistency on c2 and* AllDiff

Our filtering procedure ILF is not comparable with arc consistency (AC) on the standard SIP model which combines a global *AllDiff* constraint with $c2$ constraints.

Indeed, ILF is able to prove the inconsistency of the SIP instance of Figure 2, as shown in Ex. 7. This is still true even if the initial labeling is $l_\emptyset$. As a comparison, $AC(AllDiff + c2)$ is not able to prove the inconsistency and does not reduce any domain.

However, $AC(AllDiff + c2)$ may be stronger than ILF in some cases. Let us consider for example the case where both the pattern and the target graphs are regular graphs such that every pattern (resp. target) node has the same degree $d_p$ (resp. $d_t$), with $d_p \leq d_t$. If the initial labeling is $l_{deg}$, all pattern (resp. target) nodes have the same label $d_p$ (resp. $d_t$) and the fixpoint is reached at the first iteration without reducing any domain as $d_p \leq d_t$. However, $AC$ may be able to reduce some domains in some cases as shown in Ex. 8 for $AC(c2)$ and Ex. 9 for $AC(AllDiff)$.

*Example 8* Let us consider two pattern nodes $u$ and $v$ that are linked by an edge, and let us consider the case where the domains of the two variables $x_u$ and $x_v$ respectively associated with $u$ and $v$ is such that none of the target nodes of $D(x_u)$ and $D(x_v)$ are linked by an edge, i.e., $(D(x_u) \times D(x_v)) \cap E_t = \emptyset$. In this case, the propagation of $AC(c2)$ removes all values from $D(x_u)$ and $D(x_v)$ since each value of $D(x_u)$ (resp. $D(x_v)$) has no support in $D(x_v)$ (resp. $D(x_u)$).

Note that ILF may also be able to reduce the domains of $x_u$ and $x_v$ to empty sets if the initial labeling $l^0$ takes into account information brought by domains. This is the case, e.g., if the initial labeling is the $l_{dom}$ labeling introduced in 4.3.1. Indeed, in this case, a different label $l_u$ is associated with every different node $u$, and we have $l_u \preceq^0 l_a$ iff $a \in D(x_u)$. Hence, in our example, $l_u$ will be compatible only with the set of labels $\{l_a \mid a \in D(x_u)\}$ whereas $l_v$ will be compatible only with the set of labels $\{l_b \mid b \in D(x_v)\}$. After the first labeling extension, we will have

- $\alpha^1(u) = l_u.m_u$ such that $l_v \in m_u$ as $v \in adj(u)$, and
- for every target node $a \in D(x_u)$, $\alpha^1(a) = l_a.m_a$ such that $\forall b \in D(x_v), l_b \notin m_a$ as $b \notin adj(a)$.

Hence, $\forall a \in D(x_u), m_u \not\preceq^0 m_a$ as $m_a$ only contains labels which are not compatible with the label $l_v$ which belongs to $m_u$. Therefore, $\forall a \in D(x_u), \alpha^1(u) \not\preceq^1 \alpha^1(a)$, so that the domain of $x_u$ will be reduced to the empty set after one iteration of ILF.

*Example 9* Let us consider the case where a subset of $n$ variables associated with pattern nodes all have the same domain $D$ such that $\#D < n$. In this case, the instance has no solution as the subgraph isomorphism function must be injective. This will be detected by $AC(AllDiff)$ as it trivially detects inconsistency when the number of values is lower than the number of variables.

Note that this kind of inconsistency cannot be detected by ILF, whatever the initial labeling is.

*4.3.4 Comparison with nRF+*

When the initial labeling is $l_{deg}$, our filtering procedure ILF is not comparable with the filtering procedure nRF+ proposed by Larrosa and Valiente in [17]. Indeed, ILF is

able to prove the inconsistency of the SIP instance of Fig. 2, as shown in Ex. 7, whereas nRF+ combined with $AC(AllDiff + c2)$ is not able to prove the inconsistency: it is only able to reduce the domains of the variables associated with nodes 2 and 4 to $\{A, B, D\}$ whereas the domains of the other variables are not reduced. However, ILF is not able to reduce any domain for the SIP instance of Fig. 1 whereas nRF+ is able to reduce some domains as shown in Example 1.

When the initial labeling is $l_{dom}$, our filtering procedure ILF is at least as strong as nRF+, provided that the number of iterations $k$ is greater or equal to 1.

*Property 4* ILF(_,_,$l_{dom}$,1) is as strong as nRF+.

*Proof.* nRF+ removes from the domain of a variable associated with a pattern node $i$ every target node $a$ such that $\#\mathcal{F}(i, a) < \#adj(i)$. This value is also removed after the first iteration of ILF. Indeed, let us consider the multisets $m_i$ and $m_a$ which contain the labels of the neighbours of $i$ and $a$. When the initial labeling is $l_{dom}$, the label in $m_i$ of a node $j$ adjacent to $i$ is compatible with the label in $m_a$ of a node $b$ adjacent to $a$ only if $b \in D(x_j)$. Therefore, if $\#\mathcal{F}(i, a) < \#adj(i)$, $m_i$ will not be compatible with $m_a$ and $a$ will be removed from $D(x_i)$. ■

## 5 Weaker filtering algorithm ILF*

At each iteration of ILF, the most expensive step is the computation of the partial order $\preceq^{i+1}$ during the neighborhood extension procedure, which is done in $\mathcal{O}(n_p \cdot n_t \cdot d^{5/2})$. This comes from the fact that each call to the *test* function is in $\mathcal{O}(d^{5/2})$.

However, if $\preceq^i$ is a total order (instead of a partial order), the function $test(m_u, m_v, \preceq^i)$ may be implemented much 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 $\mathcal{O}(d \cdot \log d)$, and the complexity of the neighborhood extension becomes $\mathcal{O}(n_p \cdot n_t \cdot d)$ as the sorting of multisets may be done once.

When $\preceq^i$ is not a total order, one may extend it into a total order $\leq$. This total order may 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 $test(m_u, m_v, \preceq^i)$ returns false. As a consequence, using this approximated order may induce a weaker filtering.

In this section, we first introduce in 5.1 the theoretical framework that defines a new neighborhood labeling extension based on a total order and proves its validity; then we show in 5.2 how to compute the total order; finally, we define in 5.3 the new filtering algorithm called ILF$^*$ and we discuss its complexity and the level of consistency it achieves.

### 5.1 Labeling strengthening based on total orders

We define a new neighborhood extension procedure where old labels are extended by multisets of neighbor labels like in Definition 7, but where the partial order over the new extended labels is defined with respect to a total order over the old labels, thus reducing the complexity of its computation.

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 if $test(m_u, m_v, \preceq^i)$ returns $true$, then $test(m_u, m_v, \leq)$ also returns $true$.

**Definition 11** 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 12** Let $l = (\mathbb{L}, \preceq, \alpha)$ be a labeling, and let $\leq$ be a consistent total order for $l$. The *neighborhood extension of* $l$ *based on* $\leq$ is the labeling $l' = (\mathbb{L}', \preceq', \alpha')$ where $\mathbb{L}'$ and $\alpha'$ are defined like in Definition 7, and the order relation $\preceq' \subseteq \mathbb{L}' \times \mathbb{L}'$ is defined by

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

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

**Theorem 3** *Let* $l = (\mathbb{L}, \preceq, \alpha)$*,* $l' = (\mathbb{L}', \preceq', \alpha')$*, and* $l'' = (\mathbb{L}', \preceq'', \alpha')$*, be three labelings such that* $l'$ *is the neighborhood extension of* $l$ *and* $l''$ *is the neighborhood extension of* $l$ *based on a total consistent order.*

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

*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 an extension of $\preceq$, we have $m \preceq m' \Rightarrow m \leq m'$. Hence, $CC_{l'} \subseteq CC_{l''} \subseteq CC_l$. (i) is a direct consequence of (iii), as $l'$ is SIC (Theorem 1). □

### 5.2 Computation of a total order

Given a partial order $\preceq \subseteq \mathbb{L} \times \mathbb{L}$, one may easily compute a consistent total order by performing a topological sort of the graph $G = (\mathbb{L}, \preceq)$ [7]. Topological sorts are not unique so that different consistent total orders may be derived from a given partial order, and these different total orders may induce prunings of different strength: the less new couples of compatible nodes are introduced by the total order, the better the filtering. However, we conjecture that finding the best consistent total order is NP-hard (see appendix A).

We propose a heuristic algorithm that aims at computing a total order that introduces few new compatible couples without guarantee of optimality. This algorithm is based on the following greedy principle: starting from an empty sequence, one iteratively adds some labels at the end of the sequence, until all labels have been sequenced.

**Algorithm 2**: Computation of a consistent total order

**Input**:
the set $L_p$ of labels of pattern nodes
the set $L_t$ of labels of target nodes
a partial order $\preceq \subseteq L_p \times L_t$
**Output**: an increasing sequence $S$ defining a consistent total order for $l$

1 **for** *every label* $e \in L_p \cap L_t$ **do**
2  replace $e$ by $e'$ in $L_t$ (where $e'$ is a new label such that $e' \notin L_p \cup L_t$)
3  **for** *every label* $e'' \in L_p$ *such that* $e'' \preceq e$ **do**
4  add $(e'', e')$ to $\preceq$
5 $S \leftarrow <>$
6 **while** $L_t \neq \emptyset$ **do**
7  $Cand \leftarrow \{e_t \in L_t \mid \#\{e_p \in L_p, e_p \preceq e_t\}$ is minimal$\}$
8  $Cand' \leftarrow \{e_t \in Cand \mid \sum_{e_p \in L_p \wedge e_p \preceq e_t} \#\{e \in L_t, e_p \preceq e\}$ is maximal$\}$
9  randomly choose $e_t$ in $Cand'$
10  **for** *every label* $e_p \in L_p$ *such that* $e_p \preceq e_t$ **do**
11  add $e_p$ at the end of $S$ and remove $e_p$ from $L_p$
12  add $e_t$ at the end of $S$ and remove $e_t$ from $L_t$
13 add the remaining elements of $L_p$ at the end of $S$
14 **return** $S$

The total order is derived from the sequence by stating that a label is smaller than every label occuring after it in the sequence.

This algorithm is described in algorithm 2. It first renames the labels of the target graph that are also used in the pattern graph (lines 1–4). This does not modify the set of compatible couples of nodes, induced by the labeling. This renaming step is done in $\mathcal{O}(n_p \cdot n_t)$.

Then, at each iteration of our greedy algorithm, we choose a label $e_t$ of the target graph which has not yet been sequenced in $S$ (lines 7–9) and then we introduce in the sequence the selected label $e_t$, preceded by every label $e_p \in L_p$ such that $e_p \preceq e_t$ (lines 10–12) in order to preserve the compatibility relation.

Our goal is to sequence the labels of $L_p$ as late as possible (in order to minimize the number of target nodes which are compatible with pattern nodes). To this aim, we first compute (line 7) 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 select (line 8) the labels which are already compatible with the maximum number of labels in $L_t$.

The time complexity of this heuristic algorithm is in $\mathcal{O}(n_t \cdot \log n_t \cdot d_p \cdot d_t)$, which for simplicity will be approximated by $\tilde{O}(n_p \cdot n_t \cdot d)$.

### 5.3 Complexity of ILF*

We denote by ILF* the filtering procedure obtained when calling algorithm 2 before computing the new partial ordering $\preceq^{i+1}$, i.e., before line 12 of algorithm 1, and replacing the current partial order by the computed total order in the call to the *test* function.

*Property 5* The ILF* procedure has an overall time complexity of

$$\tilde{O}(min(k, n_p \cdot n_t) \cdot n_p \cdot n_t \cdot d)$$

| Class | Target graph $\#N_t$ | $d_{min}$ | $d_{max}$ | Pattern graph $p_n$ | $p_e$ | directed graphs ? | feasible instances ? |
|---|---|---|---|---|---|---|---|
| A | 200 | 5 | 8 | 90% | 90% | no | yes |
| B | 600 | 5 | 8 | 90% | 90% | no | yes |
| C | 1000 | 5 | 8 | 90% | 90% | no | yes |
| D | 600 | 5 | 8 | 90% | 90% | yes | yes |
| E | 300 | 20 | 300 | 90% | 90% | no | yes |
| F | 300 | 20 | 300 | 90% | 90% + 10% | no | no |

**Table 1** Description of the considered classes of instances

## 6 Experimental Results

### 6.1 Considered instances

We evaluate our approach on scale-free networks and on graphs of the GraphBase database used in [17].

*Scale-free networks (classes A to F)* Scale-free networks model a wide range of real networks, such as social, Internet, or neural networks [1]. We have randomly generated graphs using a power law distribution of degrees $P(d = k) = k^{-\lambda}$. 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. These classes are described in Table 1. 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). We have performed experiments with different percentages $p_n$ and $p_e$ and noticed that when $p_n$ and $p_e$ are smaller than 0.7 the number of solutions increases so much that instances become much easier to solve. Hence, we only report results obtained on harder instances with $p_n = 0.9$.

All instances of classes A to D are sparse feasible instances such that node degrees range between 5 and 8. Target graphs in classes A to C are undirected and their number of nodes is increased from 200 to 1000 to study scale-up properties. Target graphs in class D are directed, such that their edges have been randomly directed.

All instances of classes E and F are denser undirected instances that have been generated with $d_{min} = 20$, $d_{max} = 300$. 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 experiments reported below, each run searches for all solutions of an instance.

*GraphBase database (classes G and H)* This database contains graphs with different properties, i.e., simple, acyclic, connected, biconnected, triconnected, bipartite and planar (see [17]). It contains 113 undirected graphs and 59 directed graphs. We have considered the first 30 directed graphs. This set contains graphs ranging from 10 nodes to 462 nodes. Using these graphs, we have generated 406 directed instances (class

| | Solved instances (%) | | | | | | Average time | | | | | | Average failed nodes | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ILF($k$) | | | ILF*($k$) | | | ILF($k$) | | | ILF*($k$) | | | ILF($k$) | | | ILF*($k$) | | |
| | 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 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 0.9 | 0.4 | 57.7 | 0.3 | 0.4 | 0.5 | 19.4 | 2.0 | 0.0 | 0.5 | 0.1 | 0.0 |
| B | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 19.6 | 3.5 | 420.4 | 3.5 | 4.2 | 5.6 | 54.4 | 0.6 | 0.0 | 0.1 | 0.1 | 0.0 |
| C | 100.0 | 100.0 | 5.0 | 100.0 | 100.0 | 100.0 | 91.8 | 11.2 | 562.2 | 11.2 | 13.3 | 18.2 | 97.0 | 0.6 | 1.0 | 0.1 | 0.1 | 0.1 |
| D | 100.0 | 100.0 | 95.0 | 100.0 | 100.0 | 100.0 | 1.6 | 1.6 | 209.4 | 1.6 | 1.6 | 1.6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| E | 100.0 | 100.0 | 0.0 | 100.0 | 100.0 | 100.0 | 22.9 | 14.8 | - | 16.7 | 18.6 | 29.5 | 144.3 | 42.0 | - | 29.1 | 19.9 | 17.6 |
| F | 95.0 | 95.0 | 15.0 | 95.0 | 95.0 | 95.0 | 38.8 | 11.8 | 3.0 | 7.9 | 6.9 | 8.1 | 147.4 | 17.7 | 1.0 | 6.3 | 3.6 | 2.7 |
| G | 83.3 | 83.3 | 76.1 | 82.9 | 81.7 | 81.2 | 2.6 | 4.4 | 23.3 | 4.9 | 3.9 | 2.3 | 470.1 | 378.9 | 21.5 | 275.4 | 151.6 | 149.6 |
| H | 58.0 | 64.8 | 55.9 | 64.3 | 63.9 | 63.9 | 18.4 | 6.4 | 14.4 | 6.2 | 7.5 | 7.3 | 35865.4 | 658.7 | 50.5 | 396.5 | 344.5 | 331.1 |

**Table 2** Comparison of different variants of ILF($k$) and ILF*($k$) where $k$ 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 2Gb 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.

G) and 1276 undirected instances (class H) of the subgraph isomorphism problem by considering all couples of graphs $(G_p, G_t)$ such that $N_p < N_t$.

### 6.2 Comparison of ILF($k$) and ILF*($k$)

Algorithm 1 has been implemented in Gecode (http://www.gecode.org), using CP(Graph) and CP(Map) [10,9] which provide graph and function domain variables. These new structured domains offer abstractions that simplified the implementation of the filtering algorithms. In all experiments, the filtering algorithms ILF and ILF* have been combined with AC(c2) and AC(*AllDiff*).

Table 2 compares ILF($k$) and ILF*($k$), respectively based on an exact partial order or an approximated total order (as described in 4.2 and 5), for different limits $k$ on the number of iterations. In all variants, the initial labeling $l^0$ is the labeling $l_{deg}$ defined in 2.3. 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 ILF and ILF*. The number of failed nodes with ILF*(2) is greater than with ILF(2), but it is smaller than with ILF(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 ILF*(2) is significantly quicker than ILF(2).

Table 2 also shows that the best performing variant differs when considering different classes of instances. The directed instances of classes D and G are best solved when $k = 0$, i.e., with the simple $l_{deg}$ labeling. Instances of classes A, B, C, E, F and H are undirected; they are best solved when $k = 1$ than when $k = 0$, i.e., after one iteration of the labeling extension. Instances of class F, which have significantly higher node degrees and are infeasible, are very difficult ones. For these instances, iterative labeling extensions actually improve the solution process and the best results are obtained with $ILF*(4)$.

As a conclusion, these experiments 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. Good average results are obtained with ILF*(2).

| | Solved instances (%) | | | | | | Average time | | | | | | Average failed nodes | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | vflib | CP1 | CP2 | CP3 | CP4 | ILF*(2) | vflib | CP1 | CP2 | CP3 | CP4 | ILF*(2) | CP1 | CP2 | CP3 | CP4 | ILF*(2) |
| A | 65.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 79.6 | 0.2 | 0.9 | 2.7 | 1.3 | 0.3 | 440.1 | 19.4 | 99.9 | 13.9 | 0.5 |
| B | 0.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | - | 6.7 | 21.1 | 113.6 | 16.2 | 3.5 | 1314.1 | 54.4 | 475.6 | 18.9 | 0.1 |
| C | 0.0 | 100.0 | 100.0 | 90.0 | 100.0 | 100.0 | - | 81.3 | 105.6 | 150.7 | 42.5 | 11.2 | 6207.1 | 97.0 | 218.3 | 13.6 | 0.1 |
| D | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 0.2 | 0.1 | 1.8 | 2.2 | 3.8 | 1.6 | 2.1 | 0.0 | 0.0 | 0.0 | 0.0 |
| E | 0.0 | 84.2 | 100.0 | 63.2 | 100.0 | 100.0 | - | 53.4 | 7.7 | 88.6 | 12.0 | 16.7 | 19149.5 | 32.4 | 1518.2 | 31.5 | 29.1 |
| F | 0.0 | 40.0 | 90.0 | 35.0 | 90.0 | 95.0 | - | 48.1 | 39.2 | 35.1 | 43.2 | 7.9 | 9448.4 | 60.8 | 533.7 | 57.6 | 6.3 |
| G | 63.9 | 72.2 | 74.7 | 68.6 | 74.0 | 82.9 | 7.9 | 3.3 | 6.0 | 11.8 | 6.1 | 4.9 | 25773.6 | 680.7 | 3820.6 | 717.3 | 275.4 |
| H | 46.5 | 61.5 | 65.3 | 55.3 | 63.8 | 64.3 | 19.4 | 12.7 | 11.2 | 19.1 | 10.5 | 6.2 | 251208.9 | 2482.7 | 5704.4 | 1044.6 | 396.5 |

**Table 3** 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.

### 6.3 Comparison with state-of-the-art approaches

We now compare results obtained with our filtering procedure ILF*(2), which offers a good compromise between search space reduction and time, with other CP models. We consider four different CP models which combine different kinds of constraints and different levels of consistency as introduced in Section 2.3:

- CP1 performs forward checking on c2 and *AllDiff* constraints;
- CP2 maintains arc consistency on c2 and *AllDiff* constraints;
- CP3 combines forward checking on c2 and *AllDiff* constraints with the filtering procedure nRF+ of [17];
- CP4 combines arc consistency on c2 and *AllDiff* constraints with the filtering procedure nRF+ of [17].

All CP models have been implemented in Gecode using CP(Graph) and CP(Map).

We compare these different CP models with a state-of-the-art algorithm coming from a C++ library `vflib`.

Table 3 compares all these approaches and shows us that CP approaches clearly outperform `vflib`, except for directed instances of class D. When comparing the different CP approaches, we note that the number of failed nodes is reduced when considering higher consistencies, and ILF*(2) always explores much less nodes than other CP models. This allows ILF*(2) to solve more instances for all classes, except for instances of class H which are best solved by CP2.

## 7 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 [23] where a total order defined on some graph property labeling is strengthened until a fixpoint is reached. The extension to subgraph isomorphism has been theoretically funded. The order is partial and can also be iterated until a fixpoint is reached. However, using such a partial order is very time consuming. Instead, one can map this partial order to a total order. Performing such a mapping is hard, and can be efficiently approxi-

mated through a heuristic algorithm. Experimental results show that our propagators are efficient against state-of-the-art propagators and algorithms.

One of the advantages of our CP approach is to easily accomodate extra constraints. Further work includes the comparisons of the different CP approaches, including our ILF and ILF* algorithms on different classes of SI problems with side constraints. We will also extend our filters to better take into account the domain of the variables. Although Gecode has a copy strategy, it is worth introducing incrementality in our filtering algorithms, reducing computation between different executions of the propagator. Future work will also include the development of dynamic termination criteria for the iterative labeling, the experimental study of other degree distributions, and the analysis of alternative initial labelings.

## Acknowledgments

The authors want to thank the anonymous reviewers for their helpful comments. Christine Solnon acknowledges an ANR grant BLANC 07-1_184534: this work was done in the context of project SATTIC. This research is also partially supported by the Interuniversity Attraction Poles Programme (Belgian State, Belgian Science Policy).

## References

1. A.-L. Barabasi, Linked: How Everything Is Connected to Everything Else and What It Means, Plume, 2003.
2. C. Bessière, P. Van Hentenryck, To be or not to be ... a global constraint, in: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP), vol. LNCS 2833, Springer-Verlag, 2003, pp. 789–794.
3. D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition., IJPRAI 18 (3) (2004) 265–298.
4. L. Cordella, P. Foggia, C. Sansone, F.Tortella, M. Vento, Graph matching: A fast algorithm and its evaluation, in: ICPR '98: Proceedings of the 14th International Conference on Pattern Recognition-Volume 2, IEEE Computer Society, Washington, DC, USA, 1998, p. 1582.
5. L. Cordella, P. Foggia, C. Sansone, M. Vento, An improved algorithm for matching large graphs, in: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen, 2001, pp. 149–159.
6. L. P. Cordella, P. Foggia, C. Sansone, M. Vento, Performance evaluation of the vf graph matching algorithm, in: ICIAP '99: Proceedings of the 10th International Conference on Image Analysis and Processing, IEEE Computer Society, Washington, DC, USA, 1999, p. 1172.
7. T. H. Cormen, C. Stein, R. L. Rivest, C. E. Leiserson, Introduction to Algorithms, McGraw-Hill Higher Education, 2001.
8. P. T. Darga, M. H. Liffiton, K. A. Sakallah, I. L. Markov, Exploiting structure in symmetry detection for cnf, in: Proc. Design Automation Conference (DAC), IEEE/ACM, 2004, pp. 530–534.
9. Y. Deville, G. Dooms, S. Zampelli, P. Dupont, Cp(graph+map) for approximate graph matching, 1st International Workshop on Constraint Programming Beyond Finite Integer Domains, CP2005 (2005) 33–48.
10. G. Dooms, Y. Deville, P. Dupont, Cp(graph): Introducing a graph computation domain in constraint programming, in: Principles and Practice of Constraint Programming, vol. 3709 of Lecture Notes in Computer Science, 2005, pp. 211–225.
11. P. Foggia, C. Sansone, M. Vento, A database of graphs for isomorphism and sub-graph isomorphism benchmarking, CoRR cs.PL/0105015.
12. G. Fowler, R. Haralick, F. G. Gray, C. Feustel, C. Grinstead, Efficient graph automorphism by vertex partitioning, Artificial Intelligence 21 (1983) 245–269.

13. M. Garey, D. Johnson, Computers and Intractability, Freeman and Co., New York, 1979.
14. J. A. Grochow, M. Kellis, Network motif discovery using subgraph enumeration and symmetry-breaking, in: T. P. Speed, H. Huang (eds.), RECOMB, vol. 4453 of Lecture Notes in Computer Science, Springer, 2007, pp. 92–106.
15. J. Guo, F. Hueffner, H. Moser, Feedback arc set in bipartite tournaments is np-complete, Information Processing Letters 102 (2-3) (2007) 62–65.
16. J. E. Hopcroft, R. M. Karp, An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs., SIAM J. Comput. 2 (4) (1973) 225–231.
17. J. Larrosa, G. Valiente, Constraint satisfaction algorithms for graph pattern matching, Mathematical. Structures in Comp. Sci. 12 (4) (2002) 403–422.
18. B. D. McKay, Practical graph isomorphism., Congressus Numerantium 30 (1981) 45–87.
19. J. Régin, Développement d'outils algorithmiques pour l'intelligence artificielle. application à la chimie organique, Ph.D. thesis (1995).
20. J.-C. Regin, A filtering algorithm for constraints of difference in CSPs, in: Proc. 12th Conf. American Assoc. Artificial Intelligence, vol. 1, Amer. Assoc. Artificial Intelligence, 1994, pp. 362–367.
21. M. Rudolf, Utilizing constraint satisfaction techniques for efficient graph pattern matching, in: Theory and Application of Graph Transformations, No. 1764 in Lecture Notes in Computer Science, Springer, 1998, pp. 238–252.
22. S. Sorlin, C. Solnon, A global constraint for graph isomorphism problems, in: 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR 2004), vol. 3011 of LNCS, Springer-Verlag, 2004, pp. 287–301.
23. S. Sorlin, C. Solnon, A new filtering algorithm for the graph isomorphism problem, 3rd International Workshop on Constraint Propagation and Implementation, CP2006.
24. S. Sorlin, C. Solnon, A parametric filtering algorithm for the graph isomorphism problem, Constraints 13(4).
URL `http://liris.cnrs.fr/publis/?id=3364`
25. J. R. Ullmann, An algorithm for subgraph isomorphism, J. ACM 23 (1) (1976) 31–42.
26. G. Valiente, Algorithms on Trees and Graphs, Springer-Verlag, Berlin, 2002.
27. S. Zampelli, A constraint programming approcah to subgraph isomorphism, Ph.D. thesis, UCLouvain, Department of Computing Science & Engineering (June 2008).
28. S. Zampelli, Y. Deville, P. Dupont, Approximate constrained subgraph matching, in: Principles and Practice of Constraint Programming, vol. 3709 of Lecture Notes in Computer Science, 2005, pp. 832–836.
29. S. Zampelli, Y. Deville, C. Solnon, S. Sorlin, P. Dupont, Filtering for subgraph isomorphism, in: Proc. 13th Conf. of Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, Springer, 2007, pp. 728–742.

## Appendix A: Complexity of the best total order problem

Given a partial order $\preceq \subseteq N_p \times N_t$, our best total order problem involves finding the consistent total order $\leq$ which introduces the smallest number of new compatibility relationships from $N_p$ to $N_t$, i.e., such that $\#\{(u, v) \in N_p \times N_t, u \leq v\}$ is minimal.

Our problem may be formalized as follows. Let us define the directed bipartite graph between $N_p$ and $N_t$ which contains an arc from $u \in N_p$ to $v \in N_t$ if $\alpha(u) \preceq \alpha(v)$, and an arc from $v \in N_t$ to $u \in N_p$ if $\neg(\alpha(u) \preceq \alpha(v))$. The underlying undirected bipartite graph is complete so that it corresponds a total relation. However, it may not correspond to a total order as it may contain cycles (so that transitivity is not verified). The best total order problem is equivalent to finding the smallest number of arcs from $N_t$ to $N_p$ that should be reversed so that the graph becomes acyclic.

We conjecture that this problem is NP-hard. Indeed, the feedback arc set problem, which involves finding the smallest number of arcs that should be deleted to make a directed graph acyclic has been shown to be NP-hard, even in the case of complete bipartite graphs [15]. The only difference between the feedback arc set problem and our best total order problem is that we must not reverse (or delete) arcs from $N_p$ to $N_t$ (otherwise the total order would not be consistent).