A new filtering algorithm for the graph isomorphism problem

Sébastien Sorlin, Christine Solnon

A new filtering algorithm for the graph isomorphism problem

Sébastien Sorlin, Christine Solnon

LIRIS, CNRS UMR 5205, bât. Nautibus, University of Lyon I
43 Bd du 11 novembre, 69622 Villeurbanne cedex, France
{sebastien.sorlin,christine.solnon}@liris.cnrs.fr

Abstract. The graph isomorphism problem consists in deciding if two given graphs have an identical structure. This problem may be modeled as a constraint satisfaction problem in a very straightforward way, so that one can use constraint programming to solve it. However, constraint programming is a generic tool that may be less efficient than dedicated algorithms which take advantage of the global semantic of the original problem to reduce the search space.
Hence, we have introduced in [1] a global constraint dedicated to graph isomorphism problems, and we have defined a partial consistency —the label-consistency— that exploits all edges of the graphs in a global way to narrow variable domains. This filtering algorithm is very powerful in the sense that, for many instances, achieving it allows one to either detect an inconsistency, or reduce variable domains to singletons so that the global consistency can be easily checked. However, achieving the label-consistency implies the computation of the shortest path between every pair of vertices of the graphs, which is rather time consuming.
We propose in this article a new partial consistency for the graph isomorphism problem and an associated filtering algorithm. We experimentally show that this algorithm narrow the variable domains as strongly as our previous label-consistency, but is an order faster, so that it makes constraint programming competitive with *Nauty*, the fastest known algorithm for graph isomorphism problem.

1 Introduction

Graphs provide a rich mean for modeling structured objects and they are widely used in real-life applications to represent, *e.g.*, molecules, images, or networks. In many of these applications, one has to compare graphs to decide if their structures are identical. This problem is known as the Graph Isomorphism Problem (GIP).

More formally, a *graph* is defined by a pair (V, E) such that V is a finite set of vertices and $E \subseteq V \times V$ is a set of edges. In this paper, we consider graphs without self-loops, i.e., $\forall (u, v) \in E,\ u \neq v$. Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there exists a bijective function $f : V \rightarrow V'$ such that $(u, v) \in E$ if and only if $(f(u), f(v)) \in E'$. We shall say that f is

an *isomorphism function*. The GIP consists in deciding if two given graphs are isomorphic.

There exists many dedicated algorithms for solving GIPs such as, e.g., [2–4]. These algorithms are often very efficient (eventhough their worst case complexities are exponential). However, such dedicated algorithms can hardly be used to solve more specific problems, such as isomorphism problems with additional constraints, or larger problems that include GIPs.

An attractive alternative to these dedicated algorithms is to use Constraint Programming (CP), which provides a generic framework for solving any kind of Constraint Satisfaction Problems (CSPs). Indeed, GIPs can be transformed into CSPs in a very straightforward way [5], so that one can use generic constraint solvers to solve them. However, when transforming a GIP into a CSP, the global semantic of the problem is lost and replaced by a set of binary constraints. As a consequence, using CP to solve GIPs may be less efficient than using dedicated algorithms which have a global view of the problem.

In order to allow constraint solvers to handle GIPs in a global way so that they can solve them efficiently without loosing CP's flexibility, we have introduced in [1] a global constraint dedicated to graph isomorphism problems (the *gip* constraint), and we have defined a partial consistency —the label-consistency— and an associated filtering algorithm that exploits all edges of the graphs in a global way to narrow variable domains. This filtering algorithm is very powerful in the sense that, for many instances, achieving it allows one to either detect an inconsistency, or reduce variable domains to singletons so that the global consistency can be easily checked. However, achieving the label-consistency implies the computation of the shortest path between every pair of vertices of the graphs, which is time expensive.

Motivation and outline of the paper. The goal of this paper is to define another partial consistency for the global constraint *gip*: the iterated local label consistency (ILL-consistency). This one is based on an iterated relabelling of the graph vertices and does not need to compute the distance matrix of the graphs. As a consequence, achieving this consistency is less time expensive than achieving the label-consistency.

Section 2 recalls some complexity results for GIPs and an overview of existing approaches for solving these problems. We also recall the definition of the global constraint *gip* and the label-consistency proposed in [1]. In section 3, we introduce the iterated local label consistency (ILL-consistency), a partial consistency for the *gip* constraint based on a relabelling technic of the graph vertices from the direct neighborhood of the vertices. Section 4 experimentally compares label-consistency, ILL-consistency and *Nauty*, the fastest known algorithm for graph isomorphism problem.

2 Approaches for solving graph isomorphism problems

Complexity. The theoretical complexity of the GIP is not exactly stated: the problem is in NP but it is not known to be in P nor to be NP-complete [6] and its own complexity class, *isomorphism-complete*, has been defined. However, some topological restrictions on graphs (e.g., planar graphs [7], trees [8] or bounded valence graphs [9]) make this problem solvable in a polynomial time.

Dedicated algorithms. To solve a GIP, one has to find a one to one mapping between the vertices of the two graphs. The search space composed of all possible mappings can be explored in a "Branch and Cut" way: at each node of the search tree, some graph properties (such as edges distribution, vertices neighbourhood) can be used to prune the search space [4, 2]. This kind of approach is rather efficient and can be used to solve GIPs up to a thousand or so vertices very quickly (in less than one second).

[3] proposes another rather dual approach, which has been originally used to detect graph automorphisms (*i.e.*, non trivial isomorphisms between a graph and itself). The idea is to compute for each vertex v_i a unique label that characterizes the relationships between v_i and the other vertices of the graph, so that two vertices are assigned with a same label if and only if they can be mapped by an isomorphism function. This approach is implemented in the system *Nauty* which is, to our knowledge, the most efficient solver for the graph isomorphism problem. *Nauty* compute a canonical representation of a graph: two graphs have the same representation with Nauty if and only if they are isomorphic. The time needed to solve a GIP with *Nauty* is comparable to "Branch and Cut" methods but *Nauty* is often the quickest for large graphs [10].

Hence dedicated algorithms are very efficient to solve GIPs in practice, even-though their worst case complexities are exponential. However, they are not suited for solving more specific problems, such as GIPs with additional constraints. In particular, vertices and edges of graphs may be associated with labels that characterize them, and one may be interested in looking for isomorphisms that satisfy additional constraints on these labels. This is the case, *e.g.*, in [11] where graphs are used to represent molecules, or in computer aided design (CAD) applications where graphs are used to represent design objects [12].

Constraint Programming. CP is a generic tool for solving constraint satisfaction problems (CSPs), and it can be used to solve GIPs. A *CSP* [13] is defined by a triple (X, D, C) such that :

- X is a finite set of variables,
- D is a function that maps every variable $x_i \in X$ to its domain $D(x_i)$, *i.e.*, the finite set of values that can be assigned to x_i,
- C is a set of constraints, *i.e.*, relations between some variables which restrict the set of values that can be assigned simultaneously to these variables.

Binary CSPs only have binary constraints, *i.e.*, each constraint involves two variables exactly. We shall note $C(x_i, x_j)$ the binary constraint holding between the two variables x_i and x_j, and we shall define this constraint by the set of couples $(v_i, v_j) \in D(x_i) \times D(x_j)$ that satisfy the constraint.

Solving a CSP (X, D, C) involves finding a complete assignment, which assigns a value $v_i \in D(x_i)$ to every variable $x_i \in X$, such that all the constraints in C are satisfied.

CSPs can be solved in a generic way by using constraint programming languages (such as CHOCO [14], Ilog solver [15], or CHIP [16]), *i.e.*, programming languages that integrate algorithms for solving CSPs. These algorithms (called constraint solvers) are often based on a systematic exploration of the search space, until either a solution is found, or the problem is proven to have no solution. In order to reduce the search space, this kind of complete approach is combined with filtering technics that narrow variables domains with respect to some partial consistencies such as Arc-Consistency [13, 17, 18].

Using CP to solve GIPs. Graph isomorphism problems can be formulated as CSPs in a very straightforward way, so that one can use CP languages to solve them [19, 11]. Given two graphs $G = (V, E)$ and $G' = (V', E')$, we define the CSP (X, D, C) such that :

- a variable x_u is associated with each vertex $u \in V$, *i.e.*, $X = \{x_u / u \in V\}$,
- the domain of each variable x_u is the set of vertices of G' that have the same number of adjacent vertices than u, *i.e.*,

$$D(x_u) = \{u' \in V' \ / \ |\{(u, v) \in E\}| = |\{(u', v') \in E'\}|\}$$

- there is a binary constraint between every pair of different variables $(x_u, x_v) \in X^2$, denoted by $C_{edge}(x_u, x_v)$. This constraint expresses the fact that the vertices of G' that are assigned to x_u and x_v must be connected by an edge in G' if and only if the two vertices u and v are connected by an edge in G, *i.e.*,

$$\begin{array}{ll} \text{if } (u, v) \in E, & C_{edge}(x_u, x_v) = E' \\ \text{otherwise} & C_{edge}(x_u, x_v) = \{(u', v') \in V'^2 \mid u' \neq v' \text{ and } (u', v') \notin E'\} \end{array}$$

Once a GIP has been formulated as a CSP, one can use constraint programming to solve it in a generic way, and additional constraints, such as constraints on vertex and edge labels, may be added very easily.

Global constraint and label-consistency. When formulating a GIP into a CSP, the global semantic of the problem is decomposed into a set of binary "edge" constraints, each of them expressing locally the necessity either to maintain or to forbid one edge. As a consequence, using CP to solve GIPs will often be less efficient than using a dedicated algorithm.

To improve the solution process of CSPs associated with GIPs, one can add an *allDiff* global constraint, in order to constrain all variables to be assigned to

different vertices [11]. This constraint is redundant as each binary edge constraint only contains couples of different vertices, so that it will not be possible to assign a same vertex to two different variables. This global constraint allows a constraint solver to prune the search space more efficiently, and therefore to solve GIPs quicker.

However, even with *allDiff* global constraint, CP does not appear to be competitive with dedicated algorithms because most of the global semantic of the problem is still lost. Hence, we have introduced in [1] the global constraint *gip* to define a graph isomorphism problem. We have also defined a partial consistency —the label consistency— that strongly reduces the search space. This partial consistency is based on a labelling of the graph vertices based on the number of vertices at a given distance. We have shown that this partial consistency is very powerful and achieving it generally allows to either detect an inconsistency, or reduce variable domains to singletons so that the global consistency can be easily checked. However, achieving the label-consistency implies the computation of the shortest path between every pair of vertices of the graphs and as a consequence it is time expensive.

3 ILL-consistency

We introduce in this section another filtering algorithm for the graph isomorphism problem global constraint. The main idea of this filtering is to label every vertex with respect to its relationships with the other vertices of the graph. This labelling is "isomorphic-consistent" —in the sense that two vertices that may be associated by an isomorphism function necessary have a same label— so that it can be used to narrow the domains of the variables. These labels are built iteratively: starting from an empty label, each label is extended by considering the labels of its adjacent vertices. This labelling extension is iterated until a fixed point is reached. This fixed point corresponds to a new partial consistency for the graph isomorphism problem.

3.1 Isomorphic-consistent labelling functions

Definition. A *labelling function* is a function denoted by α that, given a graph $G = (V, E)$ and a vertex $v \in V$, returns a label $\alpha_G(v)$. This label does not depend on the names of the vertices but only on the relation defined by E between v and the other vertices of the graph. We shall note $image(\alpha_G)$ the set of labels returned by α for the vertices of a given graph G.

Definition. A labelling function α is *isomorphic-consistent* if for every pair of isomorphic graphs $G = (V, E)$ and $G' = (V', E')$, and for every isomorphism function f between G and G', the vertices that are matched by f have the same labels, *i.e.*, $\forall v \in V, \alpha_G(v) = \alpha_{G'}(f(v))$.

Example. Let us define the labelling function that labels each vertex by its degree, *i.e.*,

$$\forall v \in V, \alpha_{(V,E)}(v) = |\{u \in V, (u,v) \in E\}|$$

This labelling function is isomorphic-consistent as isomorphism functions only match vertices that have a same number of adjacent vertices.

An isomorphic-consistent labelling function can be used to narrow the domains of the variables of a CSP associated with a GIP: the domain of every variable x_u associated with a vertex u can be narrowed to the set of vertices that have the same label than u. We shall say that a labelling function α is stronger than another labelling function α' if it allows a stronger narrowing (or an equivalent narrowing), i.e., if

$$\forall (u,v) \in V^2, \alpha'_G(u) \neq \alpha'_G(v) \Rightarrow \alpha_G(u) \neq \alpha_G(v)$$

3.2 Isomorphic-consistent local relabelling function

We propose to iteratively strengthen an isomorphic-consistent labelling function: at each step, the label of every vertex v is extended with a set of couples (k,l) such that k is the number of vertices that are adjacent to v and that are labelled with l.

Definition. Given a graph $G = (V,E)$ and a labelling function α^i_G for this graph, we define the new labelling function $\alpha^{i+1}_G : V \rightarrow image(\alpha^i_G) \times \wp(\mathbb{N}^* \times image(\alpha^i_G))$ as follows:

$$\forall v \in V, \alpha^{i+1}_G(v) = \alpha^i_G(v) \cdot \{(k,l)\ , k \in \mathbb{N}, l \in image(\alpha^i_G), \\ k = |\{u \in V, (v,u) \in E \wedge \alpha^i_G(u) = l \wedge k > 0\}|\}$$

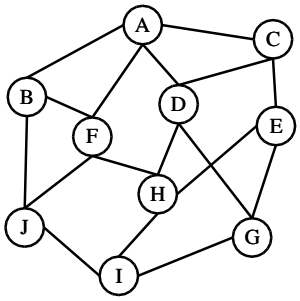

Fig. 1. A graph $G = (V,E)$

Example. For the graph $G = (V, E)$ displayed in figure 1, and the labelling function α^0_G that associates the same empty label $\emptyset$ to every vertex of G, we have $\alpha^1_G(A) = \emptyset \cdot \{(4, \emptyset)\}$ because vertex A is labelled by $\emptyset$ and has four adjacent vertices that are all labelled with $\emptyset$ whereas $\alpha^1_G(B) = \emptyset \cdot \{(3, \emptyset)\}$ because vertex B is labelled by $\emptyset$ and has three adjacent vertices that are all labelled with $\emptyset$.

Theorem 1. *Given an isomorphic-consistent labelling function α^i, the labelling function α^{i+1} is also an isomorphic-consistent labelling function.*

Proof. If α^i is an isomorphic-consistent labelling function then, given the definition of an isomorphic-consistent labelling function, for any pair of isomorphic graphs $G = (V, E)$ and $G' = (V', E')$ and for any isomorphism function f between G and G', $\forall u \in V, \alpha^i_G(u) = \alpha^i_{G'}(f(u))$. Furthermore, as f is an isomorphism function, $\forall (u, v) \in V^2, (u, v) \in E \Leftrightarrow (f(u), f(v)) \in E'$. As a consequence, $\forall u \in V, \forall l \in image(\alpha^i_G), |\{v / (u, v) \in E \wedge \alpha^i_G(v) = l\}| = |\{v' / (f(u), v') \in E' \wedge \alpha^i_{G'}(v') = l\}|$ (because f is a bijective function) and $\forall u \in V, \alpha^{i+1}_G(u) = \alpha^{i+1}_{G'}(f(u))$. The property holds.

A direct consequence of the theorem 1 is that the function α^{i+1} can be used to extend the labels of the vertices of two graphs G and G' without changing the isomorphism properties between G and G'.

Theorem 2. *Given a graph $G = (V, E)$ and a labelling function α^i, the function α^{i+1} is stronger than α^i, i.e.,*

$$\forall (u, v) \in V^2, \alpha^i_G(u) \neq \alpha^i_G(v) \Rightarrow \alpha^{i+1}_G(u) \neq \alpha^{i+1}_G(v)$$

Proof. Straightforward from the fact that each label $\alpha^i(u)$ is a prefix of the label $\alpha^{i+1}(u)$.

A direct consequence of theorem 2 is that, when relabelling the vertices of two graphs G and G' with the function α^{i+1}, the domain of each variable x_v of the CSP corresponding to a GIP between G and G' always has a size inferior or equal than the domain of x_v when the vertices are only labelled by α^i. In other words, the function α^{i+1} can filter the variable domains.

3.3 Iterative local labelling

Relabelling the graph vertices with the function α^{i+1} can introduce more different labels and as a consequence can reduce the variable domains. One can propagate these domain reductions by iterating this relabelling step until a fixed point is reached, *i.e.*, until the number of different labels is not any longer increased.

Starting from an initial isomorphic-consistent labelling α^0, we define a sequence α^1, α^2, α^3, ... of labelling functions such that each step k of this sequence corresponds to a relabelling of the vertices from the labels given at the step $k-1$. Theorem 1 states that each labelling function α^k is isomorphic-consistent, whereas theorem 2 states that each labelling function α^{k+1} is stronger than the previous one α^k. Finally, theorem 3 shows that this sequence reaches a fixed point.

Theorem 3. *Given a graph* $G = (V, E)$*, if* $\exists k \in \mathbb{N}^*$ *such that* $\forall (u, v) \in V^2, \alpha^k_G(u) = \alpha^k_G(v) \Rightarrow \alpha^{k+1}_G(u) = \alpha^{k+1}_G(v)$ *then,* $\forall j \geq k, \forall (u, v) \in V^2, \alpha^k_G(u) = \alpha^k_G(v) \Rightarrow \alpha^j_G(u) = \alpha^j_G(v)$.

Proof. Given its definition, we can see that the function α^{i+1} does not use the labels given by α^i themselves but only an operator of equality between these labels. As a consequence, when a relabelling of the vertices does not change the equality properties between the vertex labels, any further relabelling cannot change these equality properties any more.

Roughly speaking, the theorem 3 shows that, when a step of the sequence α^k does not increase the number of different vertex labels, a fixed point is reached and the relabelling process can be stopped.

Finally, we can trivially show that this fixed point is reached in at most $|V|$ steps.

3.4 ILL-consistency and associated filtering algorithm

We propose to use the sequence of isomorphic-consistent labelling functions defined previously to narrow the domains of the variables of a CSP associated with a GIP. We define the initial labelling function α^0 as the function that associates the same label $\emptyset$ to each vertex. Starting from this initial labelling function, we can then compute a sequence of stronger labelling functions until a fixed point is reached. The last labelling function can be used to define a new partial consistency for a global constraint for the graph isomorphism problem.

Let us recall the syntax proposed in [1] for this global constraint: it is defined by the relation $gip(V, E, V', E', L)$ where

- V and V' are 2 sets of values such that $|V| = |V'|$,
- $E \subseteq V \times V$ is a set of pairs of values from V,
- $E' \subseteq V' \times V'$ is a set of pairs of values from V',
- L is a set of couples which associates one different variable of the CSP to each different value of V, i.e., L is a set of $|V|$ couples of the form (x_u, u) where x_u is a variable of the CSP and u is a value of V, and such that for any pair of different couples (x_u, u) and (x_v, v) of L, both x_u and x_v are different variables and $u \neq v$.

Semantically, the global constraint $gip(V, E, V', E', L)$ is consistent if and only if there exists an isomorphism function $f : V \rightarrow V'$ such that for each couple $(x_u, u) \in L$ there exists a value $u' \in D(x_u)$ so that $u' = f(u)$.

Definition. The global constraint $gip(V, E, V', E', L)$ corresponding to a graph isomorphism problem between $G = (V, E)$ and $G' = (V', E')$ is iterated-local-label consistent (ILL-consistent) if and only if:

$$\forall (x_u, u) \in L,\ \forall u' \in D(x_u), \forall k \in \mathbb{N}\ ,\ \alpha^k_G(u) = \alpha^k_{G'}(u')$$

where α^0 is the labelling function that associates the same label $\emptyset$ to each vertex.

To make the *gip* constraint ILL-consistent, we just have to compute the sequence α^1, α^2, ... of labelling functions for each graph G and G' until a fixed point is reached and to remove from the domain of each variable x_u associated to a vertex $u \in V$ the values $u' \in D(x_u)$ such that $\alpha^k_G(u) \neq \alpha^k_{G'}(u')$.

Note that this process may be stopped before reaching the fixed point. We can use every new labelling function α^i to narrow the domains and, when all the variable domains are reduced to a singleton or when a variable domain becomes empty, the global consistency of the constraint *gip* can be easily checked.

At each step of the sequence, the vertex labels become larger and comparing such labels can be costly in time and in memory. However, one can easily show that these labels can be renamed after each relabelling step, provided that the same name is associated with identical labels in the two graphs. As a consequence, at the end of each relabelling step, labels are renamed with unique integers in order to keep the cost in memory and in time constant at each step of the sequence.

When using appropriate data structures, and provided that labels can be compared in constant time, each relabelling step for a graph $G = (V, E)$ has a time complexity of $\mathcal{O}(|E|)$: for each vertex, one has to look at the labels of the vertices that are adjacent to it. Renaming the labels at each step can be done in a time proportional to the size of the longest label of $image(\alpha)$ (*i.e.*, in the worst case $|E|$). As a consequence, as achieving the ILL-consistency needs at most $|V|$ relabelling steps (in the worst case), the maximum time complexity of our filtering algorithm is $\mathcal{O}(|V| \times |E|)$, the same than for the filtering based on labels [1]. However, we show in section 4 that the average complexity of establishing label-consistency is much more expensive than establishing ILL-consistency.

3.5 Complete example

We propose here a complete example of our relabelling procedure on the graph G of the figure 1. At each step of the sequence, the vertices are renammed by labels l_i. For reason of space, we shall note $\alpha^k_G(S)$ when $\forall (u, v) \in S^2, \alpha^k_G(u) = \alpha^k_G(v)$.

At step 0:

$$\alpha^0_G(\{A, B, C, D, E, F, G, H, I, J\}) = \emptyset$$

The next three relabelling steps successively define α^1, α^2 and α^3 as follows.

$$
\begin{array}{lll}
\alpha^1_G(\{A,D,F,H\}) & = & \emptyset.\{(4,\emptyset)\} \Rightarrow l_{1,1} \\
\alpha^1_G(\{B,C,E,G,I,J\}) & = & \emptyset.\{(3,\emptyset)\} \Rightarrow l_{1,2} \\
\hline
\alpha^2_G(\{A,D,F,H\}) & = & l_{1,1}.\{(2,l_{1,1}),(2,l_{1,2})\} \Rightarrow l_{2,1} \\
\alpha^2_G(\{B,C\}) & = & l_{1,2}.\{(1,l_{1,2}),(2,l_{1,1})\} \Rightarrow l_{2,2} \\
\alpha^2_G(\{E,G,I,J\}) & = & l_{1,2}.\{(1,l_{1,1}),(2,l_{1,2})\} \Rightarrow l_{2,3} \\
\hline
\alpha^3_G(A) & = & l_{2,1}.\{(2,l_{2,1}),(2,l_{2,2})\} \Rightarrow l_{3,1} \\
\alpha^3_G(\{B,C\}) & = & l_{2,2}.\{(2,l_{2,1}),(1,l_{2,3})\} \Rightarrow l_{3,2} \\
\alpha^3_G(\{D,F\}) & = & l_{2,1}.\{(1,l_{2,2}),(1,l_{2,3}),(2,l_{2,1})\} \Rightarrow l_{3,3} \\
\alpha^3_G(\{E,J\}) & = & l_{2,3}.\{(1,l_{2,1}),(1,l_{2,2}),(1,l_{2,3})\} \Rightarrow l_{3,4} \\
\alpha^3_G(\{G,I\}) & = & l_{2,3}.\{(1,l_{2,1}),(2,l_{2,3})\} \Rightarrow l_{3,5} \\
\alpha^3_G(H) & = & l_{2,1}.\{(2,l_{2,1}),(2,l_{2,3})\} \Rightarrow l_{3,6}
\end{array}
$$

We note that, at step 3, two vertices (A and H) have unique labels. As a consequence, we do not need to relabel these vertices during the next steps. The vertex A (resp. H) keeps the label $l_{3,1}$ (resp. $l_{3,6}$).

$$
\begin{array}{lll}
\alpha^4_G(\{B,C\}) & = & l_{3,2}.\{(1,l_{3,1}),(1,l_{3,3}),(1,l_{3,4})\} \Rightarrow l_{4,1} \\
\alpha^4_G(D) & = & l_{3,3}.\{(1,l_{3,1}),(1,l_{3,2}),(1,l_{3,5}),(1,l_{3,6})\} \Rightarrow l_{4,2} \\
\alpha^4_G(E) & = & l_{3,4}.\{(1,l_{3,2}),(1,l_{3,5}),(1,l_{3,6})\} \Rightarrow l_{4,3} \\
\alpha^4_G(F) & = & l_{3,3}.\{(1,l_{3,1}),(1,l_{3,2}),(1,l_{3,4}),(1,l_{3,6})\} \Rightarrow l_{4,4} \\
\alpha^4_G(G) & = & l_{3,5}.\{(1,l_{3,3}),(1,l_{3,4}),(1,l_{3,5})\} \Rightarrow l_{4,5} \\
\alpha^4_G(I) & = & l_{3,5}.\{(1,l_{3,4}),(1,l_{3,5}),(1,l_{3,6})\} \Rightarrow l_{4,6} \\
\alpha^4_G(J) & = & l_{3,4}.\{(1,l_{3,2}),(1,l_{3,3}),(1,l_{3,5})\} \Rightarrow l_{4,7}
\end{array}
$$

At step 4, only two vertices (B and C) share the same label.

$$
\begin{array}{l}
\alpha^5_G(B) = l_{4,1}.\{(1,l_{3,1}),(1,l_{4,4}),(1,l_{4,7})\} \Rightarrow l_{5,1} \\
\alpha^5_G(C) = l_{4,1}.\{(1,l_{3,1}),(1,l_{4,2}),(1,l_{4,3})\} \Rightarrow l_{5,2}
\end{array}
$$

From the step 5, all the vertices have different labels. As a consequence, any graph isomorphism problem involving the graph G of the figure 1 will be solved by our filtering technic.

3.6 Propagation during a search tree process

ILL-filtering does not always reduce every domain to a singleton so that it may be necessary to explore the search space. For example, if all the vertices of the graph have the same degree (*i.e.*, the same number of neighbours), our filtering algorithm is totaly inefficient. Some GIP may have more than one solution (when the graphs are automorph) and as a consequence, some variable domains are not singletons.

When ILL-filtering does not reduce the domain of each variable to a singleton, one has to explore the search space composed of all possible assignments by constructing a search tree. At each node of this search tree, the domain of one variable is splitted into smaller parts, and then filtering technics are applied to

narrow variable domains with respect to some local consistencies. These filtering technics iteratively use constraints to propagate the domain reduction of one variable to other variable domains until either a domain becomes empty (the node can be cut), or a fixed-point is reached (a solution is found or the node must be splitted).

To propagate the domain reductions implied by a search tree assignment, a first possibility is to use the set of C_{edge} constraints as defined in section 2. However, we can still use our filtering method to propagate more strongly the domain reductions. Indeed, assigning a value to a variable corresponds to giving the same unique label to the two corresponding vertices. As a consequence, we can use this new label to restart the relabelling process until it reaches another fixed point.

4 Comparative experimental results

In this section, we compare the efficiency of the label-consistency introduced in [1] and our new ILL-consistency on randomly generated graphs. We also compare these results with the results of *Nauty*, the best known algorithm dedicated to the graph isomorphism problem.

Nauty is a complete algorithm: it always solves a graph isomorphism problem. On the contrary, label-consistency and ILL-consistency are only partial consistencies. However, when labeling vertices by using these consistencies, if there is as many vertex labels than vertices, each variable domain of a GIP involving G becomes a singleton and the global consistency of the CSP is trivially checked.

As a consequence, we choose the following experimental protocol: for each considered graph $G = (V, E)$, we compute the vertex labels with the label-consistency of [1] and the vertex labels with the sequence α until reaching its fixed point. We then count the number of different vertex labels: if there is $|V|$ different labels, any GIP involving G will be perfectly filtered and the problem will be trivially solved.

Note that we only consider non automorphic graphs, *i.e.*, the only existing isomorphism function between G and itself is the identity function. As a consequence, for each considered graph $G = (V, E)$, our algorithm perfectly filters the GIP involving G if and only if the number of different vertex labels is equal to $|V|$.

We consider randomly generated graphs from the Foggia et al.'s benchmark [20]. However, as the number of vertices of the graphs of this benchmark is limited to 1000, we have also generated bigger graphs with a Nauty tool (*genrang*). As a consequence, we consider graphs having between 200 and 10000 vertices and three different edge density: 1%, 5% and 10% (the three densities proposed by [20]). For each size of graphs and each density, the given results are the average results on 100 graphs.

In order to compare the influence of the edge density, we also generate a set of graphs having 1000 vertices and an edge density varying from 1% to 50% (1% by 1%, 100 graphs for each density). We do not test with graphs that have higher

densities because, for the three considered algorithms, it is then more interesting to consider the complementary graph.

4.1 Number of labels

Nauty is a complete algorithm. As a consequence, it always found a perfect filtering (*i.e.*, a different label for each vertex).

On the contrary, label-consistency and ILL-consistency are only partial consistencies and do not find systematicaly a unique label for each vertex. However, except for little graphs with a low density (less than 400 vertices with an edge density of 1%), these two partial consistencies have actually found a perfect labelling of the vertices. As a consequence, label-consistency and ILL-consistency always solve the GIP involving the graphs having more than 400 vertices. Furthermore, the ILL-consistency is obtained at step 2 for all graphs having more than 800 vertices.

$\|V\|$	$\|L_\alpha\|$	T_α	k	$\|L_{label}\|$	T_{label}
200	199,64	0	3,40	191,92	0,01
400	400,00	0	2,88	399,87	0,07
600	600,00	0	2,14	600,00	0,19
800	800,00	0,01	2,01	800,00	0,36

Table 1. Results for the little graphs having a density of 1%. Each line successively displays: the number of vertices $|V|$ of the graphs, the average number of labels $|L_\alpha|$ (resp. $|L_{label}|$) obtained by the ILL-consistency (resp. *label*-consistency), the time needed T_α (resp. T_{label}) in seconds to establish this consistency and k, the average number of steps needed to reach the fix point of the sequence α.

Results for the little graphs having a density of 1% are given into table 1. For some of these graphs, the labeling process does not give an unique label to each vertex. However, the average number of different labels shows an extremly strong reduction of the variable domains. These results also show that ILL-consistency filter the variable domains more strongly than label-consistency and is less expensive to compute.

4.2 Execution time

We compare here the time needed to compute our filtering and to execute Nauty. All tests have be done on a PC at 1,7Ghz and 512MB of RAM running Linux (kernel 2.6).

The first graph of the figure 2 shows that, for the three algorithms, the execution time increases when the density of the graphs increases. The general behavior of the three algorithms is the same whatever the edge density is.

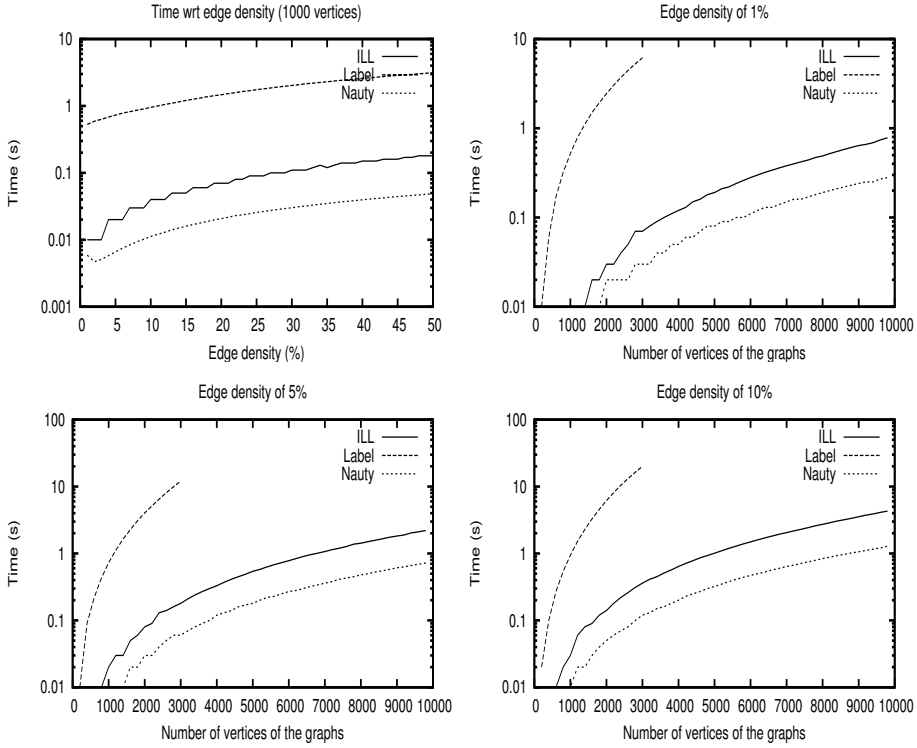

Fig. 2. Execution time w.r.t. edge density for graphs having 1000 vertices and w.r.t. graph size for graph having a density of 1%, 5% or 10%.

The label-consistency is clearly more expensive than the two others filtering technics: we had to interrupt the tests of label-consistency for graphs having up more than 3000 vertices. To establish the label-consistency and the ILL-consistency both have the same worst case complexity. However, as the fix point of the ILL-consistency is reached in only k steps with $k << |V|$, ILL-consistency is generally one order less expensive than label-consistency.

If we compare the ILL-consistency to Nauty, we can show that these two algorithms have a very similar behavior. However, Nauty is generally 3 times as fast as ILL-consistency and is still the best algorithm for graph isomorphism problems. Finally, with 512MB of RAM, our algorithm based on the ILL-consistency begins to swap with graphs having up to 9800 vertices.

5 Conclusion

We have introduced in this paper a new partial-consistency (the ILL-consistency) for the global constraint *gip* defining graph isomorphism problems. This ILL-consistency is based on the computation, for each vertex u, of a sequence of

labels which characterizes the relationship between u and its neighbours that can be viewed as a vertex invariant.

We compare ILL-consistency and label-consistency based on distances between each couple of vertices of the graphs proposed in [1]. These two consistencies are very efficient in the sense where, on randomly generated graphs having up to 400 vertices, achieving them will allow a constraint solver to either detect an inconsistency, or reduce variable domains to singletons so that the global consistency can be easily checked.

These two consistencies have both a theoritical worst case complexity of $\mathcal{O}(|V|\times|E|)$ operations for graphs having $|V|$ vertices and $|E|$ edges. However our experimental results show that ILL-consistency is faster and tigher than label-consistency. Comparing to Nauty, ILL-consistency appear to be competitive. However, Nauty is still 3 times faster than it and is still the fastest algorithm known for graph isomorphism problems.

Our experimentations show that ILL-consistency is strong enough to solve GIP with non automorph randomly generated graphs. Further work will concern the integration of our filtering algorithm into a constraint solver (such as CHOCO [14]), in order to experimentally validate and evaluate it on different kinds of graphs.

References

1. Sorlin, S., Solnon, C.: A global constraint for graph isomorphism problems. In: the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2004), Springer-Verlag (2004) 287–301
2. Ullman, J.: An algorithm for subgraph isomorphism. Journal of the Association of Computing Machinery **23**(1) (1976) 31–42
3. McKay, B.: Practical graph isomorphism. Congressus Numerantium **30** (1981) 45–87
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. McGregor, J.: Relational consistency algorithms and their applications in finding subgraph and graph isomorphisms. Information Science **19** (1979) 229–250
6. Fortin, S.: The graph isomorphism problem. Technical report, Dept of Computing Science, Univ. Alberta, Edmonton, Alberta, Canada (1996)
7. Hopcroft, J., Wong, J.: Linear time algorithm for isomorphism of planar graphs. 6^{th} Annu. ACM Symp. theory of Comput. (1974) 172–184
8. Aho, A., Hopcroft, J., Ullman, J.: The design and analysis of computer algorithms. Addison Wesley (1974)
9. Luks, E.: Isomorphism of graphs of bounded valence can be tested in polynomial time. Journal of Computer System Science (1982) 42–65
10. Foggia, P., Sansone, C., Vento, M.: A performance comparison of five algorithms for graph isomorphism. In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen (2001) 188–199
11. Régin, J.: Développement d'Outils Algorithmiques pour l'Intelligence Artificielle. Application à la Chimie Organique. PhD thesis, Univ. Montpellier II (1995)

12. Champin, P.A., Solnon, C.: Measuring the similarity of labeled graphs. In: 5th International Conference on Case-Based Reasoning (ICCBR 2003). Volume Lecture Notes in Artificial Intelligence Nu. 2689 - Springer-Verlag. (2003) 80–95
13. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press (1993)
14. Laburthe, F., the OCRE project team: CHOCO: implementing a CP kernel. In: Proc. of the CP'2000 workshop on techniques for implementing constraint programming systems, Singapore. (2000)
15. ILOG,S.A.: ILOG Solver 5.0 User's Manual and Reference Manual. (2000)
16. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex and scheduling and placement problems. In: Actes des Journées Francophones de Programmation et Logique, Lille, France. (1992)
17. Mohr, R., Henderson, T.: Arc and path consistency revisited. Artificial Intelligence **28** (1986) 65–74
18. Bessière, C., Régin, J.: Refining the basic constraint propagation algorithm. In Nebel, B., ed.: Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01), San Francisco, CA, Morgan Kaufmann Publishers, Inc. (2001) 309–315
19. Garey, M., Johnson, D.: Computers and Intractability : A Guide to The Theory of NP-Completness. W.H. Freeman, San Francisco (1979)
20. Foggia, P., Sansone, C., Vento, M.: A database of graphs for isomorphism and subgraph isomorphism benchmarking. 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition (2001) 176 –187