A global constraint for graph isomorphism problems

Sébastien Sorlin, Christine Solnon

LIRIS, CNRS FRE 2672, 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 can 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 can take advantage of the global semantic of the original problem.
Hence, we introduce in this paper a new global constraint dedicated to graph isomorphism problems, and we define an associated filtering algorithm that exploits all edges of the graphs in a global way to narrow variable domains. We then show how this global constraint can be decomposed into a set of "distance" constraints which propagate more domain reductions than "edge" constraints that are usually generated for this 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 structure is 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 shall restrict our attention to 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 [21, 17, 7]. These algorithms are often very efficient (eventhough their worst case complexities are exponential). However, such dedicated algorithms can hardly be used to solve more general 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 [16], 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 isomorphism problems may be less efficient than using dedicated algorithms which have a global view of the problem.

Outline of the paper. The goal of this paper is to allow constraint solvers to handle GIPs in a global way so that they can solve them efficiently without loosing CP's flexibility. To this aim, we introduce a new global constraint for modeling GIPs, and we show how one can take benefit of this globality to solve more efficiently GIPs.

Section 2 gives some complexity results for GIPs and an overview of existing approaches for solving these problems. Section 3 presents some properties of the GIP which are used to define our filtering algorithm. In section 4, we introduce a new global constraint for modeling GIPs on non directed graphs, and we define filtering technics for this global constraint. In section 5, we discuss the extension of our work to directed graphs and to the subgraph isomorphism problem.

2 Solving graph isomorphism problems

Complexity. The theoretical complexity of the GIP is not exactly stated: the problem is in NP but it is not know to be in P or to be NP-complete [10] and its own complexity class, *isomorphism-complete*, has been defined. However, some topological restrictions on graphs (e.g., planar graphs [12], trees [2] or bounded valence graphs [15]) 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 [7, 21]. This kind of approach is rather efficient and can be used to solve GIPs up to 1000 vertices very quickly (less than 1 second).

[17] 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. 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 [9].

Hence dedicated algorithms are very efficient to solve GIPs in practice, eventhough their worst case complexities are exponential. However, they are not suited for solving more general 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 finding isomorphisms that satisfy particular constraints on these labels. This is the case, e.g., in [19] where graphs are used to represent molecules, or in computer aided design (CAD) applications where graphs are used to represent design objects [6].

Constraint Programming. CP is a generic tool for solving constraint satisfaction problems (CSPs), and it can be used to solve GIPs. A *CSP* [20] 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 assignement, which assigns one 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 [13], or CHIP [1]), *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 techniques that narrow variables domains with respect to some partial consistencies such as Arc-Consistency [20, 18, 5].

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 [11, 19]. 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 entering and leaving edges than u, *i.e.*,

$$\begin{aligned} D(x_u) = \{u' \in V' \; / \; & |\{(u,v) \in E\}| = |\{(u',v') \in E'\}| \text{ and} \\ & |\{(v,u) \in E\}| = |\{(v',u') \in E'\}|\} \end{aligned}$$

- there is one binary constraint between every pair of different variables. The constraint holding between two different variables $(x_u, x_v) \in X^2$ is denoted by $C_{edge}(x_u, x_v)$ and 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, can be added very easily.

Discussion. 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 [19]. 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. However, adding this global constraint allows a constraint solver to prune the search space more efficiently, and therefore to solve GIPs quicker. Hence, with respect to the definition of globality introduced in [4], this *allDiff* constraint is not semantically global, as it can be decomposed into a semantically equivalent set of binary constraints, but it is AC-operationally global, as an AC-filtering on the global constraint is stronger than an AC-filtering on the equivalent set of binary constraints.

In this paper, we introduce a new global constraint to define GIPs. This global constraint is not semantically global, as it can be decomposed into a set of binary edge constraints as described above. However, by considering all edges of the graphs in a global way, we can prune more efficiently the search space. Note that this GIP global constraint can be combined with an *allDiff* constraint to filter even more values.

3 Some properties of the GIP

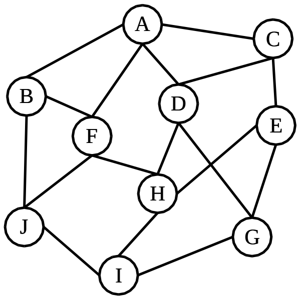
When looking for an isomorphism function between two given graphs, one can use vertex properties to reduce the search space. For example, one can compute the degree of each vertex, or the number of adjacent triangles to each vertex, and use these "vertex invariants" to prune every mapping which violates them. More generally, a vertex invariant is a label $l(v)$ assigned to each vertex v such that if there exists an isomorphism function which links v to v' then $l(v) = l(v')$ (but the converse is not necessary true). The most famous exemple of vertex invariants is the *degree* of a vertex (*i.e.*, the number of incoming and outgoing edges) : if f is an isomorphism function between $G=(V,E)$ and $G'=(V',E')$, then for each vertices $v \in V$, the vertices v and $f(v)$ have the same degree.

We introduce in this section some definitions and theorems that will be used to define a new vertex invariant based on distances. We shall restrict our attention to *undirected graphs*, *i.e.*, graphs with undirected edges so that (u,v) and (v,u) are considered to be the same edge. The extension of our work to directed graphs is discussed in section 5.1. We shall assume that graphs are connected, so that every vertex is reachable from any other vertex.

3.1 Definitions and theorems

Definition 1. Given a graph $G=(V,E)$, a *path* between two vertices u and v is a sequence $<v_0, v_1, v_2, ..., v_k>$ of vertices such that $v_0=u$, $v_k=v$ and for all $i \in [1,k]$, $(v_{i-1}, v_i) \in E$. The *length* of a path π, noted $|\pi|$, is the number of its edges.

Definition 2. Given a graph $G=(V,E)$, a *shortest path* between two vertices u and v is a path between u and v the length of which is minimal. The *length of the shortest path* between u and v is noted $\delta_G(u,v)$. We shall say that $\delta_G(u,v)$ is the *distance* between u and v.



$\delta_G(u,v)$	A	B	C	D	E	F	G	H	I	J
A	0	1	1	1	2	1	2	2	3	2
B	1	0	2	2	3	1	3	2	2	1
C	1	2	0	1	1	2	2	2	3	3
D	1	2	1	0	2	2	1	1	2	3
E	2	3	1	2	0	2	1	1	2	3
F	1	1	2	2	2	0	3	1	2	1
G	2	3	2	1	1	3	0	2	1	2
H	2	2	2	1	1	1	2	0	1	2
I	3	2	3	2	2	2	1	1	0	1
J	2	1	3	3	3	1	2	2	1	0

Fig. 1. A graph $G=(V,E)$ and distances between any pair of its vertices.

Theorem 1. *Given two graphs $G=(V,E)$ and $G'=(V',E')$ such that $|V|=|V'|$, and a bijective function $f : V \rightarrow V'$, the two following properties are equivalent*

$$f \text{ is an isomorphism function, i.e., } (u,v) \in E \Leftrightarrow (f(u), f(v)) \in E' \quad (1)$$

$$\forall (u,v) \in V^2, \delta_G(u,v) = \delta_{G'}(f(u), f(v)) \quad (2)$$

Proof. (1) $\Rightarrow$ (2)*: if f is an isomorphism function, then (u, v) is an edge of G iff $(f(u), f(v))$ is an edge of G' so that $< v_1, v_2, ..., v_n >$ is a path in G iff $< f(v_1), f(v_2), ..., f(v_n) >$ is a path in G', and therefore $< v_1, v_2, ..., v_n >$ is a shortest path in G_1 iff $< f(v_1), f(v_2), ..., f(v_n) >$ is a shortest path in G_2, and property* (2) *holds.*
(2) $\Rightarrow$ (1)*: For any pair of vertices $(u, v) \in V \times V$, if (u, v) is an edge of G, then $< u, v >$ is the shortest path between u and v so that $\delta_G(u, v) = 1$, and therefore $\delta_{G'}(f(u), f(v)) = 1$, so that $(f(u), f(v))$ is an edge of G' (and vice versa).*

Theorem 1 will be used to define "distance" constraints for propagating domain reductions when solving GIPs. We now introduce some more definitions that will be used to define a partial consistency and a filtering algorithm for GIPs.

Definition 3. Given a graph $G = (V, E)$, a vertex $u \in V$, and a distance $i \in [0, |V| - 1]$, we note $\Delta_G(u, i)$ the set of vertices that are at a distance of i from u, and $\#\Delta_G(u, i)$ the number of vertices that are at a distance of i from u, *i.e.*,

$$\Delta_G(u, i) = \{v \in V / \delta_G(u, v) = i\} \quad \text{and} \quad \#\Delta_G(u, i) = |\Delta_G(u, i)|$$

For example, for the graph G of Fig. 1, we compute:

$$\begin{array}{ll} \Delta_G(A, 0) = \{A\} & \#\Delta_G(A, 0) = 1 \\ \Delta_G(A, 1) = \{B, C, D, F\} & \#\Delta_G(A, 1) = 4 \\ \Delta_G(A, 2) = \{E, G, H, J\} & \#\Delta_G(A, 2) = 4 \\ \Delta_G(A, 3) = \{I\} & \#\Delta_G(A, 3) = 1 \\ \Delta_G(A, i) = \emptyset & \#\Delta_G(A, i) = 0, \ \forall i \geq 4 \end{array}$$

Definition 4. Given a graph $G = (V, E)$ and a vertex $u \in V$, we note $\#\Delta_G(u)$ the sequence composed of $|V|$ numbers respectively corresponding to the number of vertices that are at a distance of 0, 1, ... $|V| - 1$ from u, *i.e.*,

$$\#\Delta_G(u) = \ < \#\Delta_G(u, 0), \#\Delta_G(u, 1), ..., \#\Delta_G(u, |V| - 1) >$$

We shall omit zeros at the end of sequences.
For example, the sequences of the vertices of the graph G of Fig. 1 are

$$\begin{array}{l} \#\Delta_G(A) = \#\Delta_G(D) = \#\Delta_G(F) = \ < 1, 4, 4, 1 > \\ \#\Delta_G(B) = \#\Delta_G(C) = \#\Delta_G(E) = \#\Delta_G(G) = \#\Delta_G(I) = \ < 1, 3, 4, 2 > \\ \#\Delta_G(H) = \ < 1, 4, 5, 0 > \\ \#\Delta_G(J) = \ < 1, 3, 3, 3 > \end{array}$$

Each sequence $\#\Delta_G(u)$ characterizes the relationships of the vertex u with the other vertices of G by means of distances. Hence, when looking for a graph isomorphism, one can use these sequences as a vertex invariant to reduce the search space by pruning all mappings that associate two vertices with different sequences. However, many different vertices within a same graph may have a same sequence so that this criterion will not narrow much the search space. For example, on the graph example of Fig. 1, there are five different vertices the sequence of which is $< 1, 3, 4, 2 >$. Definition 6 will go one step further in order to characterize more precisely the relationships of a vertex u with the other vertices of the graph.

Definition 5. Given a graph $G = (V, E)$, we note $\#\Delta_G$ the set of all different sequences associated with the vertices of G, *i.e.*,

$$\#\Delta_G = \{s | \exists u \in V, s = \#\Delta_G(u)\}$$

For example, the set of all different sequences for the graph G of fig 1 is

$$\#\Delta_G = \{< 1, 3, 3, 3 >, < 1, 3, 4, 2 >, < 1, 4, 4, 1 >, < 1, 4, 5, 0 >\}$$

Definition 6. Given a vertex $u \in V$, we note $label_G(u)$ the set of all tuples (i, s, k) such that i is a distance, s is a sequence, and k is the number of vertices that are at a distance of i from u and the sequence of which is s, *i.e.*,

$$\begin{aligned} label_G(u) = \{(i, s, k) \;/\; & i \in [0, |V| - 1], \\ & s \in \#\Delta_G, \text{ and} \\ & k = |\{v \in \Delta_G(u, i) / \#\Delta_G(v) = s\}|\} \end{aligned}$$

We shall omit the tuples (i, s, k) such that $k = 0$.
For example, for the graph G of Fig. 1, we compute

$$\begin{aligned} label_G(A) = \{ & (0, < 1, 4, 4, 1 >, 1), \\ & (1, < 1, 3, 4, 2 >, 2), (1, < 1, 4, 4, 1 >, 2), \\ & (2, < 1, 3, 3, 3 >, 1), (2, < 1, 3, 4, 2 >, 2), (2, < 1, 4, 5, 0 >, 1), \\ & (3, < 1, 3, 4, 2 >, 1)\} \end{aligned}$$

as there is one vertex (A) that is at a distance of 0 from A and which sequence is $< 1, 4, 4, 1 >$, two vertices (B and C) that are at a distance of 1 from A and which sequence is $< 1, 3, 4, 2 >$, two vertices (D and F) that are at a distance of 1 from A and which sequence is $< 1, 4, 4, 1 >$, etc...

Theorem 2. *Given two graphs $G = (V, E)$ and $G' = (V', E')$, if there exists an isomorphism function $f : V \rightarrow V'$ that matches the two graphs then, for each vertex $u \in V$, $label_G(u) = label_{G'}(f(u))$.*

Proof. *f is a bijection, and the distance between two vertices u and v in G is equal to the distance between their associated vertices $f(u)$ and $f(v)$ in G' (see theorem 1). Therefore, the number of vertices of G that are at a distance of i from u is equal to the number of vertices of G' that are a distance of i from $f(u)$, so that $\#\Delta_G(u) = \#\Delta_{G'}(f(u))$. As a consequence, the set of sequences of the two graphs are equals, i.e., $\#\Delta_G = \#\Delta_G$. Then, for each sequence $s \in \#\Delta_G$, and for each vertex $u \in V$, the number of vertices that are at a distance of i from u and which sequence is s is equal to the number of vertices that are at a distance of i from $f(u)$ and which sequence is also s, and therefore $label_G(u) = label_{G'}(f(u))$.*

3.2 Algorithms and complexities

We discuss in this section time and space complexities required to compute the different values introduced in 3.1. These complexities are given for a non directed connected graph $G = (V, E)$ such that $|V| = n$ and $|E| = p$ with $n - 1 \leq p < n^2$.

$\boldsymbol{\delta_G(u, v)}$**.** All definitions introduced in section 3.1 are based on distances between couples of vertices. A Breadth First Search (BFS) [8] from each vertex of G is needed to compute them all: n BFS are needed, each of them performing $\mathcal{O}(p)$ operations, so that the time complexity is in $\mathcal{O}(np)$. The space needed to store these informations is in $\mathcal{O}(n^2)$.

$\boldsymbol{\Delta_G(u, i)}$**,** $\boldsymbol{\#\Delta_G(u, i)}$ **and** $\boldsymbol{\#\Delta_G(u)}$**.** All these values can be computed in an incremental way while computing shortest paths: each time a distance $\delta_G(u, v)$ is computed, the vertex u (resp. v) is added to the set $\Delta_G(v, \delta_G(u, v))$ (resp. $\Delta_G(u, \delta_G(u, v))$), both $\#\Delta_G(u, \delta_G(u, v))$ and $\#\Delta_G(v, \delta_G(u, v))$ are incremented, and the sequences $\#\Delta_G(u)$ and $\#\Delta_G(v)$ are updated by incrementing their $\delta_G(u, v)^{\text{th}}$ component. All these operations can be done in constant time. The space needed to store these informations is in $\mathcal{O}(n^2)$.

$\boldsymbol{label_G(u)}$**.** To compute and compare labels efficiently, we first sort the set of all sequences, so that a unique integer is associated with each different sequence. This is done in $\mathcal{O}(n^2.log(n))$ operations as there is at most n different sequences, and the comparison of two sequences is in $\mathcal{O}(n)$. Then, the computation of all labels can be done in $\mathcal{O}(n^2)$ operations (as there are n labels to compute, each of them containing at most n different triples), and requires $\mathcal{O}(n^2)$ space. Finally, the comparison of two labels can be done in $\mathcal{O}(n)$ operations, provided that the set of triples (i, s, k) contained in each label is sorted.

As a consequence, the computation of all values introduced in section 3.1 for a graph $G = (V, E)$ requires $\mathcal{O}(|V|.|E| + |V|^2.log(|V|))$ operations and $\mathcal{O}(|V|^2)$ space.

4 A global constraint for GIPs

We now introduce a new global constraint for tackling GIPs efficiently. Syntactically, this constraint 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 $u' = f(u)$.

This global constraint is not semantically global as it can be represented by a semantically equivalent set of binary constraints as described in section 2. However, the *gip* constraint allows us to exploit the global semantic of GIPs to solve them more efficiently. We now define a partial consistency, and an associated filtering algorithm (section 4.1); we shall then describe how to propagate constraints (section 4.2).

4.1 Label-consistency and label-filtering for *gip* constraints

Theorem 2 establishes that an isomorphism function always maps vertices that have identical labels. Hence, we can define a partial consistency for the *gip* constraint, called *label-consistency*, that ensures that for each couple $(x_i, v) \in L$, each value u in the domain of x_i has the same label than v.

Definition 7. The global constraint $gip(V, E, V', E', L)$ is label-consistent iff

$$\forall (x_u, u) \in L,\ \forall u' \in D(x_u),\ label_{(V,E)}(u) = label_{(V',E')}(u')$$

To achieve label-consistency, one just has to compute the label of each vertex of the two graphs, as described in section 3.2, and remove from the domain of each variable x_u associated with a vertex $u \in V$ every value $u' \in D(x_u)$ such that $label_{(V,E)}(u) \neq label_{(V',E')}(u')$.

This label-filtering often drastically reduces variable domains. Let us consider for example the graph G of Fig. 1. The first three triples (sorted by increasing distance, and then by increasing sequence number) of the label of each vertex are:

$$\begin{aligned}
label_G(A) &= \{(0, <1,4,4,1>, 1), (1, <1,3,4,2>, 2), (1, <1,4,4,1>, 2), ...\}\\
label_G(B) &= \{(0, <1,3,4,2>, 1), (1, <1,3,3,3>, 1), (1, <1,4,4,1>, 2), ...\}\\
label_G(C) &= \{(0, <1,3,4,2>, 1), (1, <1,3,4,2>, 1), (1, <1,4,4,1>, 2), ...\}\\
label_G(D) &= \{(0, <1,4,4,1>, 1), (1, <1,3,4,2>, 2), (1, <1,4,4,1>, 1), ...\}\\
label_G(E) &= \{(0, <1,3,4,2>, 1), (1, <1,3,4,2>, 2), (1, <1,4,5,0>, 1), ...\}\\
label_G(F) &= \{(0, <1,4,4,1>, 1), (1, <1,3,3,3>, 1), (1, <1,3,4,2>, 1), ...\}\\
label_G(G) &= \{(0, <1,3,4,2>, 1), (1, <1,3,4,2>, 2), (1, <1,4,4,1>, 1), ...\}\\
label_G(H) &= \{(0, <1,4,5,0>, 1), (1, <1,3,4,2>, 2), (1, <1,4,4,1>, 2), ...\}\\
label_G(I) &= \{(0, <1,3,4,2>, 1), (1, <1,3,3,3>, 1), (1, <1,3,4,2>, 1), ...\}\\
label_G(J) &= \{(0, <1,3,3,3>, 1), (1, <1,3,4,2>, 2), (1, <1,4,4,1>, 1), ...\}
\end{aligned}$$

Actually, all vertices of G have different labels. As a consequence, for any *gip* constraint between G and another graph G', label-filtering will allow one either to detect an inconsistency (if some label of G is not in G'), or to reduce the domain of each variable to a singleton so that global consistency can be easily checked.

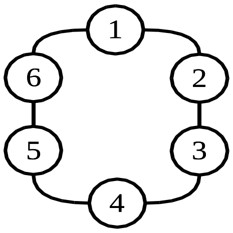
On this example, we can compare label-consistency of a *gip* constraint with arc consistency of the CSP defined in section 2. Let us define another graph $G' = (V', E')$ that is isomorphic to the graph $G = (V, E)$ of Fig. 1, and such that each vertex $u \in V$ is renamed into u' in G'. Let us consider the CSP which modelizes the problem of finding an isomorphism between these two graphs, as defined in section 2. For this CSP, the domain $D(x_u)$ of each variable x_u contains every vertex $u' \in V'$ such that u and u' have a same number of incident edges, so that :

$$D(x_A) = D(x_D) = D(x_F) = D(x_H) = \{A', D', F', G'\}$$
$$D(x_B) = D(x_C) = D(x_E) = D(x_G) = D(x_I) = D(x_J) = \{B', C', E', G', I', J'\}$$

This CSP already is arc consistent so that an AC-filtering will not reduce any domain. Note also that, on this example, adding an *allDiff* constraint does not allow to filter more domains.

4.2 Propagating constraints

Label-filtering does not always reduce every domain to a singleton so that it may be necessary to explore the search space. Let us consider for example the graph displayed in Fig. 2. This graph has many symetries (it is isomorphic to any graph obtained by a circular permutation of its vertices), so that all vertices are associated with a same sequence and a same label. In this case, label-filtering does not narrow any domain.



For each vertex $u \in V$,
$\#\Delta_G(u) = \ < 1, 2, 2, 1 >$ and
$label_G(u) = \{\ (0, < 1, 2, 2, 1 >, 1),$
$(1, < 1, 2, 2, 1 >, 2),$
$(2, < 1, 2, 2, 1 >, 2),$
$(3, < 1, 2, 2, 1 >, 1)\}$

Fig. 2. A circular graph $G = (V, E)$ and vertex sequences and labels

When label-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 *gip* constraint, a first possibility is to use the set of C_{edge} constraints as defined in section 2. However, we can take advantage of results obtained while achieving label-consistency to define "tighter" constraints. By tighter, we mean that each constraint is defined by a smaller (or equal) number of allowed couples of values so that propagating them may narrow domains more strongly.

The idea is to constrain each pair of variables (x_u, x_v) associated with a pair of vertices (u, v) of the first graph to take their values within the set of pairs of vertices (u', v') of the second graph such that the distance between u and v is equal to the distance between u' and v'. Indeed, Theorem 1 proves that a bijective function between two graphs is an isomorphism function if and only if this function preserves the distances between every pair of vertices in each graph. Therefore, the global constraint $gip(V, E, V', E', L)$ is semantically equivalent to a set of "distance" constraints defined as follows: for all $((x_u, u), (x_v, v)) \in L \times L$ such that $u \neq v$,

$$C_{distance}(x_u, x_v) = \{(u', v') \in V' \times V' \mid \delta_{(V,E)}(u, v) = \delta_{(V',E')}(u', v')\}$$

One can easily show that each binary constraint $C_{distance}(x_u, x_v)$ is tighter than (or equal to) the corresponding binary constraint $C_{edge}(x_u, x_v)$ defined in section 2:

- if the vertices of G associated with the variables x_u and x_v are connected by an edge in G, then $C_{distance}(x_u, x_v) = C_{edge}(x_u, x_v) = E'$,
- otherwise, $C_{distance}(x_u, x_v) \subseteq C_{edge}(x_u, x_v)$ as $C_{edge}(x_u, x_v)$ contains all pairs of vertices of G' that are not connected by an edge whereas $C_{distance}$-(x_u, x_v) only contains the pairs of vertices of G' such that the distance between them is equal to the distance between the vertices of G associated with x_u and x_v.

As a consequence, propaging a $C_{distance}$ constraint will always remove at least as many values as propaging the corresponding C_{edge} constraint, and in some cases it will remove more values.

For example, let us consider the graph G of Fig. 2 and let us define another graph $G' = (V', E')$ that is isomorphic to G and such that each vertex $u \in V$ is renamed into u' in G'. We note x_u the variable associated with each vertex $u \in V$. The edge constraint between x_1 and x_4 contains every pair of vertices of G' that are not connected by an edge, *i.e.*,

$$\begin{aligned} C_{edge}(x_1, x_4) = \{ & (1', 3'), (1', 4'), (1', 5'), (2', 4'), (2', 5'), (2', 6'), \\ & (3', 5'), (3', 6'), (3', 1'), (4', 6'), (4', 1'), (4', 2'), \\ & (5', 1'), (5', 2'), (5', 3'), (6', 2'), (6', 3'), (6', 4')\} \end{aligned}$$

whereas the distance constraint between x_1 and x_4 only contains pairs of vertices of G' that are at a distance of 3 one from each other as the distance between 1 and 4 is 3, *i.e.*,

$$C_{distance}(x_1, x_4) = \{(1', 4'), (2', 5'), (3', 6'), (4', 1'), (5', 2'), (6', 3')\}$$

As the distance constraint between x_1 and x_4 is tighter than the corresponding edge constraint, it can propagate more domain reductions. For example, if x_1 is assigned to $1'$, a forward-checking propagation of $C_{distance}(x_1, x_4)$ reduces the domain of x_4 to the singleton $\{4'\}$, whereas a forward-checking propagation of $C_{edge}(x_1, x_4)$ only reduces the domain of x_4 to $\{3', 4', 5'\}$, that is, the set of vertices that are not connected to $1'$ by an edge.

Also, after the suppression of value $1'$ from the domain of x_1, an AC propagation of $C_{distance}(x_1, x_4)$ will remove the value $4'$ from the domain of x_4, whereas an AC propagation of $C_{edge}(x_1, x_4)$ will not remove any value.

5 Extensions to directed graphs and subgraph isomorphism problems

5.1 Directed graphs.

All definitions and theorems introduced in section 3 actually hold for directed graphs, provided that paths respect edge directions. However, in this case, there may exist many couples of vertices which are not connected by a path respecting edge directions, so that the sequences describing vertices may be very short. In this case, label-filtering may not reduce much the search space.

Another way to extend our work to directed graphs is to first consider the corresponding non-directed graph (by ignoring edge directions), and to compute sequences and labels on this non-directed graph. Then, constraints can be added to express edge directions.

Finally, a last way to extend our work to directed graphs is to consider together several kinds of paths, each one having a corresponding kind of distance, e.g. directed paths, which respect edge directions, non directed paths, which ignore edge directions... We can then use these different kinds of paths to compute, for each vertex as many labels as defined distances. Two vertices can then be linked together if and only if, for each defined distance, the two vertices have the same label.

Obviously, the third possibility should allow one to narrow more tightly domains. However, it is also more expensive to achieve. Hence, we shall experimentally compare these three different possibilities.

5.2 Subgraph isomorphism problems

A graph $G = (V, E)$ is a *subgraph* of another graph $G' = (V', E')$, denoted by $G \subseteq G'$, if $V \subseteq V'$ and $E = E' \cap (V \times V)$. A graph $G = (V, E)$ is an *isomorphic subgraph* of another graph $G' = (V', E')$ if there exists a subgraph $G'' \subseteq G'$ that is isomorphic to G. The Subgraph Isomorphism Problem (SGIP) consists in deciding if a graph $G = (V, E)$ is an isomorphic subgraph of another graph $G' = (V', E')$.

If the theoretical complexity of the GIP is not yet completely stated, the SGIP clearly is an NP-complete problem [11]. Actually, the SGIP is a more challenging problem for which rather small instances still cannot be solved within a reasonable amount of time.

One can modelize a SGIPs as CSPs, in a very similar way than for GIPs. However, like for GIPs, one could take benefit of the global semantic of the problem to define more powerful filtering algorithms. However, a subgraph isomorphism function $f : V \rightarrow V'$ does not preserve distances between vertices like a graph isomorphism function, as stated in Theorem 1: for every path $< v_1, v_2, ..., v_n >$ in G, there exists a path $< f(v_1), f(v_2), ..., f(v_n) >$ in G' but the opposite is not always true (f is not a bijection and it may exist some vertices of V' which are not linked to a vertex of V). As a consequence, the distance $\delta_G(u, v)$ between two vertices u and v of G may be greater than the distance $\delta_{G'}(f(u), f(v))$ between $f(u)$ and $f(v)$.

Hence, filtering technics for handling efficiently SGIPs, could be based on the following property: given two undirected graphs $G = (V, E)$ and $G' = (V', E')$, if f is an isomorphism function between G and a subgraph of G', then

$$\forall (u, v) \in V \times V,\ \delta_G(u, v) \geq \delta_{G'}(f(u), f(v))$$

This property could be used to define a partial consistency, and an associated filtering algorithm. The idea would be to check that, for every vertex u of G, the domain of the variable x_u associated with u only contains vertices u' such that, for every distance $k \in 1..|V_1| - 1$, the number of vertices $v \in G$ for which $\delta_G(u, v) \leq k$ is lower or equal to the number of vertices $v' \in G'$ for which $\delta_{G'}(u', v') \leq k$.

We could also use this property to define distance constraints for propagating domain reductions while exploring a search tree.

6 Conclusion

We have introduced in this paper a new global constraint for defining graph isomorphism problems. To tackle efficiently this global constraint, we have first defined a partial consistency, called label-consistency, and an associated filtering algorithm, that can be used to narrow variable domains before solving the CSP. This label-consistency is based on the computation, for each vertex u, of a label which characterizes the global relationship between u and the other vertices of the graph by means of shortest paths and which can be viewed as a vertex invariant. In many cases, achieving label-consistency 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.

Then, for cases such that label-consistency does not allow to solve the graph isomorphism problem, we have defined a set of distance constraints that is semantically equivalent to the global GIP constraint and that can be used to propagate domain reductions. We have shown that these distance constraints are tighter than edge constraints, that simply check that edges are preserved by the mapping, so that propagating distance constraints remove more (or as many) values than propagating edge constraints. Note that this set of distance constraints can be combined with a global *allDiff* constraint to propagate even more domain reductions.

Both label-filtering and the generation of distance constraints can be done in $\mathcal{O}(np + n^2log(n))$ operations for graphs having n vertices and p edges (such that $n - 1 \leq p \leq n^2$. As a comparison, achieving arc consistency with AC2001 on a CSP describing an isomorphism problem with edge constraints will require $\mathcal{O}(ed^2)$ operations [3] where e is the number of constraints, *i.e.*, $e = n(n-1)/2$, and d is the size of the largest domain, *i.e.*, $d = n$. Hence, the complexity of achieving label-consistency on the global constraint is an order lower than the complexity of achieving AC-consistency on a semantically equivalent set of edge constraints. However, one should note that these two consistencies are not comparable: for some graphs, such as the graph of Fig. 1, label-consistency is stronger and actually solves the problem, whereas AC-consistency on edge constraints does not reduce any domain; for some other graphs, such as the graph of Fig. 2, label-consistency does not reduce any domain, whereas AC-consistency on edge constraints can reduce some variable domains as soon as one variable is assigned to a value.

Further work will first concern the integration of our filtering algorithm into a constraint solver (such as CHOCO [14]), in order to experimentally validate and evaluate it. A distance constraint can be seen like an invariant of a couple of vertices : it is a label $l(u, v)$ assigned to a couple (u, v) of vertices such that, if there exists an isomorphism function which links vertices (u, v) to vertices (u', v') then $l(u, v) = l(u', v')$. One could choose stronger invariants *i.e.*, defining more "tighter" constraints than the one based on the distance between the two vertices. For example, one could choose a label $l(u, v)$ of a couple (u, v) of vertices of a graph G which describes the distances between u, v and all vertices of the graphs more than only the distance between u and v. The label $l(u, v)$ should be then a set of triples (d_u, d_v, n), each one expressing the fact that there is n vertices of G which are respectively at distance of d_u from vertex u and at distance of d_v from vertex v. Using stronger invariants can prune more efficiently the search space but can also be more expensive to compute and to compare. One has to find the best compromise between the time needed to compute it and the efficiency of the filtering, and experiments should be performed to determine this. Finally, we shall clarify relationships between different levels of partial consistencies on $C_{distance}$ and C_{edge} constraints. In particular, for all examples we have experimented, we have noticed that after the assignment of a variable, a forward checking propagation of $C_{distance}$ constraints always reduces domains as much as an AC propagation of C_{edge} constraints. Hence, we shall try to prove this property, or find a counter example to it.

References

1. Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex and scheduling and placement problems. In *Actes des Journées Francophones de Programmation et Logique, Lille, France*, 1992.
2. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms.* Addison Wesley, 1974.
3. Christian Bessière and Marie-Odile Cordier. Arc-consistency and arc-consistency again. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 108–113, Menlo Park, CA, USA, July 1993. AAAI Press.
4. Christian Bessière and Pascal Van Hentenryck. To be or not to be... a global constraint. *CP'03, Kinsale, Ireland*, pages 789–794, 2003.
5. Christian Bessière and Jean-Charles Régin. Refining the basic constraint propagation algorithm. In Bernhard Nebel, editor, *Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, pages 309–315, San Francisco, CA, August 4–10 2001. Morgan Kaufmann Publishers, Inc.
6. Pierre-Antoine Champin and Christine Solnon. Measuring the similarity of labeled graphs. *5th International Conference on Case-Based Reasoning (ICCBR 2003)*, Lecture Notes in Artificial Intelligence Nř2689 - Springer-Verlag:80–95, 2003.
7. Luigi Pietro Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159. Cuen, 2001.
8. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.
9. Pasquale Foggia, Carlo Sansone, and Mario Vento. A performance comparison of five algorithms for graph isomorphism. In *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 188–199. Cuen, 2001.
10. Scott Fortin. The graph isomorphism problem. Technical report, Dept of Computing Science, Univ. Alberta, Edmonton, Alberta, Canada, 1996.
11. Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to The Theory of NP-Completness.* W.H. Freeman, San Francisco, 1979.
12. John E. Hopcroft and Jin-Kue Wong. Linear time algorithm for isomorphism of planar graphs. 6^{th} *Annu. ACM Symp. theory of Comput.*, pages 172–184, 1974.
13. ILOG,S.A. *ILOG Solver 5.0 User's Manual and Reference Manual.* 2000.
14. François Laburthe and 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. Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer System Science*, pages 42–65, 1982.
16. James J. McGregor. Relational consistency algorithms and their applications in finding subgraph and graph isomorphisms. *Information Science*, 19:229–250, 1979.
17. Brendan D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.
18. Roger Mohr and Thomas C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:65–74, 1986.
19. Jean-Charles Régin. *Développement d'Outils Algorithmiques pour l'Intelligence Artificielle. Application à la Chimie Organique.* PhD thesis, Univ. Montpellier II, 1995.
20. Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.
21. Jeffrey D. Ullman. An algorithm for subgraph isomorphism. *Journal of the Association of Computing Machinery*, 23(1):31–42, 1976.