# Speeding up constrained path solvers with a reachability propagator

Student: **Luis Quesada** Supervisors: **Peter Van Roy and Yves Deville**

Université catholique de Louvain
Place Sainte Barbe, 2, B-1348 Louvain-la-Neuve, Belgium
{luque, pvr, yde}@info.ucl.ac.be

## 1 Introduction

We present a propagator which we call *Reachability* that implements a generalized reachability constraint on a directed graph $g$. Given a source node $source$ in $g$, we can identify three parts in the *Reachability* constraint: (1) the relation between each node of $g$ and the set of nodes that it reaches, (2) the association of each pair of nodes $\langle source, i\rangle$ with its set of cut nodes (i.e., the nodes that are included in all paths going from $source$ to $i$), and (3) the association of each pair of nodes $\langle source, i\rangle$ with its set of bridges (i.e., the edges that are included in all paths going from $source$ to $i$).

Our contribution is a propagator that is suitable for solving the Simple Path problem with mandatory nodes [Sel02,CB04]. This problem consists in finding a simple path in a directed graph containing a set of mandatory nodes. A simple path is a path where each node is visited once. Certainly, this problem can be trivially solved if the graph has no cycles since in that case there is only one order in which we can visit the mandatory nodes [Sel02]. However, if the graph has cycles the problem is NP complete since we can easily reduce the Hamiltonian Path problem [GJ79,CLR90] to this problem.

Notice however that we can not trivially reduce Simple Path with mandatory nodes to Hamiltonian path. One could think that optional nodes (i.e. nodes that are not mandatory) can be eliminated in favor of new edges as a preprocessing step, which finds a path between each pair of mandatory nodes. However, the problem is that the paths that are precomputed may share nodes. This may lead to violations of the requirement that a node should be visited only once.

In figure 1, we illustrate this situation. Mandatory nodes are in solid lines. In the second graph we have eliminated the optional nodes by connecting each pair of mandatory nodes depending on whether there is a path between them. However, we observe that the second graph has a simple path going from node 1 to node 4 while the first one does not. Indeed, the simple path in the second graph is not a valid solution to the original problem since it implies that node 3 is visited twice.

The other reason that makes the elimination of optional nodes difficult is that finding $k$ pairwise disjoint paths between $k$ pairs of nodes $\langle s_1, d_1\rangle$,$\langle s_2, d_2\rangle$,...,$\langle s_k, d_k\rangle$ is NP complete [SP78].

In general, we can say that the set of optional nodes that can be used when going from a mandatory node $a$ to a mandatory node $b$ depends on the path that has been traversed before reaching $a$. This is because the optional nodes used in the path going from the source to $a$ can not be used in the path going from $a$ to $b$.

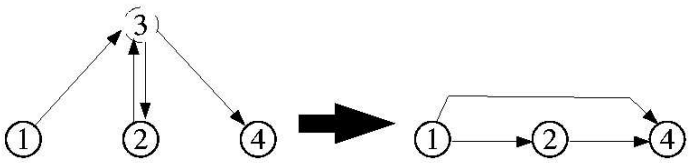


**Fig. 1.** Relaxing Simple Path with mandatory nodes by eliminating the optional nodes

## 2 The reachability propagator

### 2.1 Reachability constraint

The Reachability Constraint is defined as follows:

$$Reachability(g, source, rn, cn, be) \equiv \forall_{i \in N}. \begin{array}{l} rn(i) = Reach(g, i) \wedge \\ cn(i) = CutNodes(g, source, i) \wedge \\ be(i) = Bridges(g, source, i) \end{array} \quad (1)$$

where $g$ is a graph whose set of nodes is a subset of $N$, $source$ is a node of g, $rn(i)$ is the set of nodes that $i$ reaches in $g$ (defined by $Reach(g, i)$), $cn(i)$ is the set of nodes appearing in all paths from source to $i$ in $g$ (defined by $CutNodes(g, source, i)$), and $be(i)$ is the set of edges appearing in all paths from $source$ to $i$ in $g$ (defined by $Bridges(g, source, i)$) [1].

The above definition of *Reachability* implies the following properties which are crucial for the pruning that *Reachability* performs. These properties define relations between the functions $rn$, $cn$, $be$, nodes and edges. These relations can then be used for pruning, as we show in section 2.2.

1. If $\langle i, j \rangle$ is an edge of $g$, then $i$ reaches $j$.

$$\forall_{\langle i,j \rangle \in edges(g)}.j \in rn(i) \quad (2)$$

2. If $i$ reaches $j$, then $i$ reaches all the nodes that $j$ reaches.

$$\forall_{i,j,k \in N}.j \in rn(i) \wedge k \in rn(j) \rightarrow k \in rn(i) \quad (3)$$

3. If $source$ reaches $i$ and $j$ is a cut node between $source$ and $i$ in $g$, then $j$ is reached from $source$ and $j$ reaches $i$:

$$\forall_{i,j \in N}.i \in rn(source) \wedge j \in cn(i) \rightarrow j \in rn(source) \wedge i \in rn(j) \quad (4)$$

[1] Any node in $N$ is a cut node between $i$ and $j$ if there is no path going from $i$ to $j$. Similarly, any edge in $N \times N$ is a bridge between $i$ and $j$ if there is no path going from $i$ to $j$.

4. Reached nodes, cut nodes and bridges are nodes and edges of g:

$$\forall_{i \in N}.rn(i) \subseteq nodes(g) \quad (5) \qquad \forall_{i \in N}.cn(i) \subseteq nodes(g) \quad (6) \qquad \forall_{i \in N}.be(i) \subseteq edges(g) \quad (7)$$

### 2.2 Pruning rules

We implement the constraint in Equation 1 with the propagator

$$Reachability(G, Source, RN, CN, BE) \tag{8}$$

In this propagator we have that:

- $G$ is a graph variable (i.e., a variable whose domain is a set of graphs [DDD05]). The upper bound of $G$ ($max(G)$) is the greatest graph to which $G$ can be instantiated, and its lower bound ($min(G)$) is the smallest graph to which $G$ can be instantiated. So, $i \in nodes(G)$ means $i \in nodes(min(G))$ and $i \notin nodes(G)$ means $i \notin nodes(max(G))$ (the same applies for edges). In what follows, $\{\langle N_1, E_1\rangle \# \langle N_2, E_2\rangle\}$ will denote a graph variable whose lower bound is $\langle N_1, E_1\rangle$ and upper bound is $\langle N_2, E_2\rangle$. I.e., if $g = \langle n, e\rangle$ is the graph that $G$ approximates, then $N_1 \subseteq n \subseteq N_2$ and $E_1 \subseteq e \subseteq E_2$.
- $Source$ is an integer representing the source in the graph.
- $RN(i)$ is a Finite Integer Set (FS) [Ger97] variable associated with the set of nodes that can be reached from node $i$. The upper bound of this variable ($max(RN(i))$) is the set of nodes that could be reached from node $i$ (i.e., nodes that are not in the upper bound are nodes that are known to be unreachable from $i$). The lower bound ($min(RN(i))$) is the set of nodes that are known to be reachable from node $i$. In what follows $\{S_1 \# S_2\}$ will denote a FS variable whose lower bound is the set $S_1$ and upper bound is the set $S_2$.
- $CN(i)$ is a FS variable associated with the set of nodes that are included in every path going from $Source$ to $i$.
- $BE(i)$ is a FS variable associated with the set of edges that are included in every path going from $Source$ to $i$.

The definition of *Reachability* and its derived properties give place to a set of propagation rules. We show here the most representative ones. The others are given in [QVD05b]. A propagation rule is defined as $\frac{C}{A}$ where $C$ is a condition and $A$ is an action. If $C$ is true, the pruning defined by $A$ can be performed.

- From (2) $\forall_{\langle i,j\rangle \in edges(g)}.j \in rn(i)$ we obtain:

$$\frac{\langle i, j\rangle \in edges(min(G))}{j \in min(RN(i))} \tag{9}$$

- From (3) $\forall_{i,j,k \in N}.j \in rn(i) \wedge k \in rn(j) \rightarrow k \in rn(i)$ we obtain:

$$\frac{j \in min(RN(i)) \wedge k \in min(RN(j))}{k \in min(RN(i))} \tag{10}$$

– From (4)$\forall_{i,j \in N}.i \in rn(source) \wedge j \in cn(i) \rightarrow j \in rn(source) \wedge i \in rn(j)$ we obtain:

$$\frac{i \in min(RN(Source)) \wedge j \in min(CN(i))}{j \in min(RN(Source))} \quad (11)$$

$$\frac{i \in min(RN(Source)) \wedge j \in min(CN(i))}{i \in min(RN(j))} \quad (12)$$

– From (1)$\forall_{i \in N}.rn(i) = Reach(g, i)$ we obtain:

$$\frac{j \notin Reach(max(G), i)}{j \notin max(RN(i))} \quad (13)$$

– From (1)$\forall_{i \in rn(source)}.cn(i) = CutNodes(g, source, i)$ we obtain:

$$\frac{j \in CutNodes(max(G), Source, i)}{j \in min(CN(i))} \quad (14)$$

– From (1)$\forall_{i \in rn(source)}.be(i) = Bridges(g, source, i)$ we obtain:

$$\frac{e \in Bridges(max(G), Source, i)}{e \in min(BE(i))} \quad (15)$$

– From (5) $\forall_{i \in N}.rn(i) \subseteq nodes(g)$, (6)$\forall_{i \in N}.cn(i) \subseteq nodes(g)$ and (7)$\forall_{i \in N}.be(i) \subseteq edges(g)$ we obtain:

$$\frac{k \in min(RN(i))}{k \in nodes(min(G))} \quad (16) \qquad \frac{k \in min(CN(i))}{k \in nodes(min(G))} \quad (17) \qquad \frac{e \in min(BE(i))}{e \in edges(min(G))} \quad (18)$$

*Reachability* has been implemented using a message passing approach on top of the multi-paradigm programming language Oz [Moz04]. In [QVD05a], we discuss the implementation of *Reachability* in detail.

In [QVD05b], we show the effectiveness of our *Reachability* propagator by applying it to the Simple Path problem with mandatory nodes. We do an experimental evaluation of *Reachability* that shows that it provides strong pruning, obtaining solutions with very little search. Furthermore, we show that *Reachability* is also useful for defining a good labeling strategy and dealing with ordering constraints among mandatory nodes. These experimental results give evidence that *Reachability* is a useful primitive for solving constrained path problems over graphs.

## 3 Related work

– The cycle constraint of CHIP [BC94,Bou99] $cycle(N, [S_1, ..., S_n])$ models the problem of finding $N$ distinct circuits in a directed graph in such a way that each node is visited exactly once. Certainly, Hamiltonian Path can be implemented using this constraint. In fact, [Bou99] shows how this constraint can be used to deal with the Euler knight problem (which is an application of Hamiltonian Path). However, this constraint only covers the case where we are to visit all the nodes of the graph, which is a specific case of Simple Path with mandatory nodes.

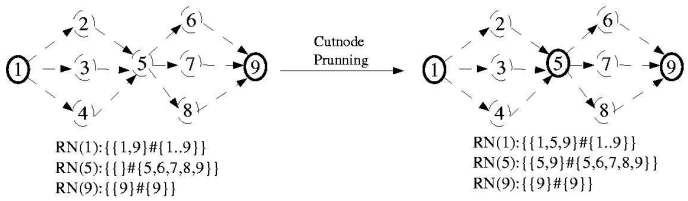


**Fig. 2.** Discovering cut nodes

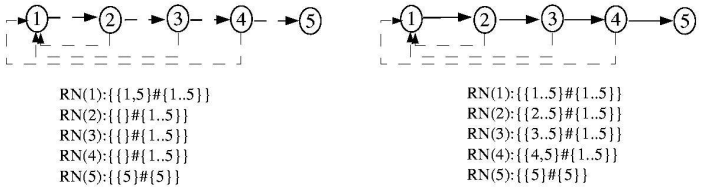


**Fig. 3.** Discovering bridges

- [Sel02] suggests some algorithms for discovering mandatory nodes and non-viable edges in directed acyclic graphs. These algorithms are extended by [CB04] in order to address directed graphs in general with the notion of strongly connected components and condensed graphs. Nevertheless, examples like [tes] represent tough scenarios for this approach since almost all the nodes are in the same strongly connected component.
- CP(Graph) introduces a new computation domain focussed on graphs including a new type of variable, graph domain variables, as well as constraints over these variables and their propagators [DDD05]. CP(Graph) also introduces node variables and edge variables, and is integrated with the finite domain and finite set computation domain. Consistency techniques have been developed, graph constraints have been built over the kernel constraints and global constraints have been proposed. Certainly, Simple Path with mandatory nodes can be implemented in terms of the global constraints of this framework. However, we still have to compare the performance of *Reachability* with respect to this approach.

## 4 Conclusion and future work

We presented *Reachability*: a constrained path propagator that can be used for speeding up constrained path solver. After introducing its semantics and pruning rules, we showed how our approach differ from related approaches.

It is important to remark that both the computation of cut nodes and the computation of bridges play an essential role in the performance of *Reachability*. The reason is that each one is able to prune when the other can not. Notice that Figure 2 is a context where the computation of bridges cannot infer anything since there is no bridge. Similarly, Figure 3 represents a context where the computation of bridges discovers more information than the computation of cut nodes.

A drawback of our approach is that each time we compute cut nodes and bridges from scratch, so one of our next tasks is to overcome this limitation. I.e., given a graph $g$, how can we use the fact that the set of cut nodes between $i$ and $j$ is $s$ for recomputing the set of cut nodes between $i$ and $j$ after the removal of some edges? We believe that a dynamic algorithm for computing cut nodes and bridges will improve our performance in a radical way.

As mentioned before, the implementation of *Reachability* was suggested by a practical problem regarding mission planning in the context of an industrial project. Our future work will concentrate on making propagators like *Reachability* suitable for non-monotonic environments (i.e., environments where constraints can be removed). Instead of starting from scratch when such changes take place, the pruning previously performed can be used to repair the current pruning.

## References

[BC94] N. Beldiceanu and E. Contejean. Introducing global constraints in chip, 1994.

[Bou99] Eric Bourreau. *Traitement de contraintes sur les graphes en programmation par contraintes*. Doctoral dissertation, Université Paris, Paris, France, 1999.

[CB04] Hadrien Cambazard and Eric Bourreau. Conception d'une contrainte globale de chemin. In *10e Journées nationales sur la résolution pratique de problèmes NP-complets (JNPC'04)*, pages 107–121, Angers, France, June 2004.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[DDD05] G. Dooms, Y. Deville, and P. Dupont. CP(Graph):introducing a graph computation domain in constraint programming. In *CP2005 Proceedings*, 2005.

[Ger97] C. Gervet. Interval propagation to reason about sets: Defi nition and implementation of a practical language. *CONSTRAINTS journal*, 1(3):191–244, 1997.

[GJ79] Michael Garey and David Johnson. *Computers and Intractability: A Guide to the The Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[Moz04] Mozart Consortium. The Mozart Programming System, version 1.3.0, 2004. Available at *http://www.mozart-oz.org/*.

[QVD05a] Luis Quesada, Peter Van Roy, and Yves Deville. Reachability: a constrained path propagator implemented as a multi-agent system. In *CLEI2005 Proceedings*, 2005.

[QVD05b] Luis Quesada, Peter Van Roy, and Yves Deville. The reachability propagator. Research Report INFO-2005-07, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 2005. Available at *http://www.info.ucl.ac.be/˜luque/SPMN/paper.pdf*.

[Sel02] Meinolf Sellmann. *Reduction Techniques in Constraint Programming and Combinatorial Optimization*. Doctoral dissertation, University of Paderborn, Paderborn, Germany, 2002.

[SP78] Y. Shiloach and Y. Perl. Finding two disjoint paths between two pairs of vertices in a graph. *Journal of the ACM*, 1978.

[tes] Spmn_52a. Available at *http://www.info.ucl.ac.be/˜luque/CICLOPS2005/test_52.ps*.