A note on arithmetic constraint propagation

Arnaud Malapert, Jean-Charles Régin

CNRS Internal Report I3S/RR-2012-03-FR

A note on arithmetic constraint propagation

Arnaud Malapert and Jean-Charles Régin
I3S CNRS – Université Nice-Sophia Antipolis
{arnaud.malapert,jean-charles.regin}@unice.fr

Résumé

Nous nous intéressons à la résolution de problèmes de grandes tailles contenant principalement des contraintes arithmétiques comme des problèmes de plus court chemin. Nous montrons qu'un simple modèle PPC n'est pas compétitif avec des algorithmes ou contraintes spécialisés. Ce phénomène est causé par le mécanisme de propagation de contraintes qui détermine l'ordre dans lequel les contraintes sont révisées. Fort de cette observation, nous proposons de modifier la propagation pour intégrer certaines idées cruciales, mais simples, des algorithmes spécialisés. Nous montrons l'intérêt de cette idée sur des problèmes de plus court chemin et nous questionnons sa généralisation à travers des expériences sur d'autres problèmes. Nous analysons aussi la dynamique du phénomène de propagation et constatons que le nombre de variables traitées plusieurs fois dans une même phase de propagation est faible.

Abstract

We consider the resolution by constraint programming of large problems, *i.e.* involving millions of constraints, which mainly imply arithmetic constraints, like shortest path problems or other related problems. We show that a simple constraint programming model is not competitive with dedicated algorithms (or dedicated constraints). This mainly comes from the propagation mechanism, *i.e.* the ordering along which the constraints are revised. Thus, we propose a modification of this propagation mechanism integrating the main ideas of the dedicated algorithms. We give some experiments for the shortest path problem and more general problems which confirms the robustness of our approach. Last, we give some results showing that only a few variables are considered more than once during a propagation step.

1 Introduction

The shortest path problem (SPP) is a component of more general problems, like resource constrained shortest path or scheduling (PERT/CPM) problems. It consists in finding a path between a source node s to each node v of a directed graph which minimizes the sum of the lengths of its constituent arcs. The SPP is efficiently solved by the labeling method combined to one of the two well-known strategies for selecting the next variable to scan proposed by Bellman-Ford-Moore [2] and Dijkstra [6]. In order to solve general problems, we first compare the performance of constraint programming (CP) to dedicated algorithms. In fact, we will show that the design of a simple model competitive with the dedicated algorithms is not obvious. This lack of efficiency caused by the propagation of constraints will lead us to modify the propagation mechanism. Then, we will study the effects of our modification on radio link frequency assignment problems. Last, we will report results on the number of times a variable is considered within a single propagation step.

2 Labeling method and CP

The labeling method and constraint programming have some strong similarities for solving the SPP. Let $G = (X, A)$ be a directed graph where each arc $(u, v) \in A$ has a length $l(u, v)$, and let s be a node of X. The problem is to compute for every node $v \in X$ the shortest path distance from the source node s to nodes v.

Labeling method The labeling method for solving the SPP is defined as follows [5]. For every node v, the method maintains its distance label $d(v)$, its parent $p(v)$ and its status $S(v) \in \{unreached, labeled, scanned\}$. The initialization consists in fixing these values for every node as follows: v, $d(v) = \infty$, $p(v) = null$, and $S(v) = unreached$. The first step consists in setting the source as follows: $d(s) = 0$ and $S(s) = labeled$. Then, the SCAN opera-

tion is applied in order to label nodes until all nodes have a scanned status. After a SCAN operation, some unreached and scanned nodes may become labeled. If there is no negative cycle, the labeling method terminates and the parents define a correct shortest path tree and $d(v)$ is the shortest path distance from s to v for each node v.

Function SCAN(v)

for *each* $(v, w) \in A$ **do**
  if $d(v) + l(v, w) < d(w)$ **then**
    $d(w) \leftarrow d(v) + l(v, w)$;
    $S(w) \leftarrow labeled$;
    $p(w) \leftarrow v$;
$S(v) \leftarrow scanned$;

The Bellman-Ford algorithm maintains the set of labeled nodes as a FIFO queue: the next node to scan is removed from the head of the queue whereas labeled nodes are added to the tail of the queue. The Dijkstra algorithm selects the labeled node v with the minimum value $d(v)$, but is restricted to positive lengths.

Constraint propagation On the other hand, constraint programming associates a filtering algorithm FILTER(C) to each constraint C, which aims at reducing the domain of its variables by removing inconsistent values, *i.e.* values which can not belong to a solution of the constraint. FILTER(C) returns the set of variables that have been modified by the filter. After each modification of the domain of a variable, all constraints involving this variable need to be reconsidered, because new domain reductions are possible. This process is repeated until no modification occurs which necessarily happens because domains are finite. The function FILTER(Q) applies this mechanism called *constraint propagation*.

Function FILTER(Q)

while $Q \neq \emptyset$ **do**
  `/* pick y in Q and remove y from Q */`
  for *each constraint* C *involving* y **do**
    $M \leftarrow$ FILTER(C) ;
    if $\exists x \in M / D(x) = \emptyset$ **then return** false;
    $Q \leftarrow Q \cup M$;
return true;

The conclusion of studies [8, 3, 1] about the variable/constraint revision ordering for arc consistency algorithms is that: the *"variable oriented propagation"* is preferable to the "constraint oriented propagation" ; the best strategy is to *select a variable with the minimum domain size*. The variable oriented propagation selects successively variables whose domain have been modified and applies the filtering algorithms of all the constraints involving this variable. The constraint oriented propagation considers the constraints in turn independently of the variables, like in the classical AC-3 algorithm. Note that a similar study on labeling methods lead to the same conclusion [7].

3 Shortest path model

In this section, we describe a simple constraint programming model for the shortest path problem. For each node $v \in X$, let $d(v)$ be a positive unbounded integer variable with the exception of $d(s)$ which is equal to 0. For each arc $(u, v) \in A$, we state the constraint $d(v) \leq d(u) + l(u, v)$ which applies the *bound consistency* rather than arc consistency. Therefore, the maximum value of the domain of $d(v)$ after the initial propagation is the shortest path distance from s (no search algorithm is needed).
At first glance, we can simulate Bellman-Ford algorithm by using a FIFO Q and Dijkstra algorithm by selecting a variable of Q with the minimum domain size (because minimum values of the domains are 0 and maximum values represent the distance from s). In fact, this assumption is false for almost all existing solvers because of the initialization of the constraints, *i.e.* the first call to their filtering algorithms. Most solvers add a constraint at the end of the propagation of the current subproblems as explained below.

4 Modifying the constraint propagation

Constraint initialization Most solvers handle only modifications between two successive calls of the same filtering algorithm. This information (called delta domains or waiting list) can be used to speed up the filtering algorithm. However, its management is complex because other variables/constraints can be propagated between the modification of the domain of a variable and the propagation of the variable. For instance, FILTER(C) usually considers the current domains when the constraint C is initialized, but the constraint C can be propagated more than once for the same modifications if some variables involved in C are also in Q. Thus, the solver can apply FILTER(C) for modifications that are prior to the initialization of the constraint and this is not reasonable. So, most of the solvers postpone the initialization of a constraint until the propagation of the current subproblems terminates.
This method has some consequences that can be emphasized for the shortest path problem in Figure 1. A CP solver will successively add a constraint and propagate the already defined problem when a modification occurs. The constraints will be added or filtered in

that order: $C_1, C_2, C_3, C_4, C_5, C_6$ (v_1 is reduced), C_2 (v_2 is reduced), C_4, C_7, C_8, C_9 (v_1 is reduced), C_2 (v_2 is reduced), C_4, C_{10}, C_{11}, C_{12} (v_1 is reduced), C_2, and C_4. The labeling method will insert s in the queue. Constraints $C_1, C_3, C_5, C_8, C_{11}$ will be considered from s. They add nodes v_1, v_2, v_3, v_4, v_5 to the queue. Then, v_1 is extracted; C_2 is filtered; v_2 is extracted; C_4 is filtered; v_3 is extracted; C_6 is filtered (adding v_1); C_7, C_9, C_{10}, C_{12} are filtered; v_1 is extracted, C_2 is filtered (adding v_2); v_2 is extracted and C_4 is filtered. We can note that less constraints are reconsidered by the labeling method. More precisely, constraints C_2 and C_4 are considered twice by the labeling method and four time by the CP solver.

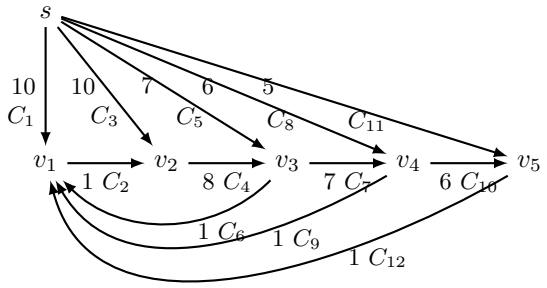

Figure 1: Example of shortest path problem.

In order to prevent this issue, we implement a mechanism which does not need to wait until the end of the propagation for adding a new constraint. In this case, the initialization is separated from the propagation. On the contrary, the immediate propagation calls FILTER(Q) as soon as a modification occurs during the initialization.

Parent checking heuristic The Bellman-Ford algorithm can be improved by using the parent checking heuristic which consists in scanning a node v only if its parent $p(v)$ is not in the queue Q. For shortest path problems, the presence of the parent of a node v in the queue Q will necessarily lead to a new modification of $d(v)$. Even though it is not always true for constraint propagation, we will evaluate a variant of the constraint propagation mechanism where the propagation of a variable is postponed if its parent belong to the queue Q.

5 Experimental results

We report here experimental results for several variants of the constraint propagation mechanism of the Choco solver (`http://choco.mines-nantes.fr`) which is an open source java library. Let P** and _** denote the variants with and without the parent checking heuristic respectively. Let *F* and *D* denote the variants where the next variable is selected according to FIFO or min-domain strategies. Let **_ and **S denote the variants with immediate propagation and separated initialization. Note that the naive implementation of min-domain and parent checking heuristics is linear in the number of variables in Q.

Shortest path problem We discuss here the results for SPP reported in Table (a) on four categories of random graphs with 2000 nodes and approximately 2 millions of constraints. The categories D and SD define complete and semi-complete directed acyclic graph (DAG) with positive lengths and 20% of additional arcs. In a semi-complete DAG, indegrees and outdegrees of half of the nodes are equal to one. Each additional arc creates a cycle with a positive length. The categories DN and SDN differ from D and SD by the presence of negative lengths (but no negative cycle). The rows $\#v$ and t correspond to the number of propagated variables and the propagation time given in seconds respectively.
The min-domain strategy is more efficient than the FIFO, and the separation of the initialization and the propagation is worthwhile. Combining min-domain with the separation of the initialization and the propagation perfectly simulate Dijkstra algorithm even in presence of negative lengths. Therefore, the solver propagates each variable only once which is the minimal possible number of propagated variables (because all variables are modified at least once). However, this combination is often a little bit more time consuming than with the FIFO strategy because of the size of Q with the exception of the category DN.
The parent checking heuristic provides a small improvement for *F_ and postpones approximately 1% of the propagated variables. However, it significantly slows down the propagation for *FS, because of the size of Q leads to postpone approximately 135% of the propagated variables (some variables are postponed more than once). We have also evaluated other variants of the parent checking heuristic (an ancestor is in the queue, several parents ...), but the results were quite similar. Finally, these results clearly show that the parent checking heuristic is almost useless.

Radio link frequency assignment problem The RLFAP model contains variables (6000 in average) with enumerated domains and constraints which enforce either the arc consistency either the bound consistency. the top Table (b) gives the results obtained during the initial propagation. Variants with parent checking does not show any improvement even if 13% and 89% of the propagated variables were respectively postponed when using PF_ and PFS. The rows $\%v$, $\%r$ and t respectively correspond to the percentage of

		_F_	_FS	_D_	_DS	PF_	PFS	PD_	PDS
D	$\#v$	10516	7551	9164	1999	10343	7480	9164	1999
	t	16.4	3.2	14.7	4.8	16.3	12.2	3.4	5.6
SD	$\#v$	10231	7888	9164	1999	10091	7772	9164	1999
	t	6.9	1.8	6.4	2.7	6.8	5.5	1.9	3.3
DN	$\#v$	13916	7177	12506	1999	13732	7146	12506	1999
	t	19.1	12.0	17.1	6.0	18.8	13.5	16.1	6.7
SDN	$\#v$	13021	7328	11885	1999	12867	7291	11885	1999
	t	7.6	1.7	7.1	2.7	7.6	5.2	2.1	3.4

(a) Results for shortest path problems

	_F_	_FS	_D_	_DS
$\%v$	101	92	99	84
$\%r$	5	2	4	1
t	0.67	0.78	0.70	3.30

	_F_		_D_	
	$\%_{00}v$	$\%_{00}r$	$\%_{00}v$	$\%_{00}r$
SAT	30.1	1.5	28.6	0.1
UNSAT	4.6	0.5	1.9	0.1

(b) RLFAP: initial propagation and shaving

propagated variables relatively to the number of variables, the percentage of reentrant variables relatively to the number of propagated variables, and the propagation time given in seconds. **S reduces the number of propagated and reentrant variables but it does not pass on computation time, especially for _DS. The ratio of reentrant variables represents the maximum improvement obtained by another variable ordering strategies. Indeed, the propagation mechanism can be improved if we reduce the number of time a variable is propagated during the same step. Each variable which is modified at least once during a propagation must be extracted from Q, because it is necessary to study the consequences of its modification. Therefore, a "perfect" propagation would consider, in the best case, only once each modified variable. This preliminary results should still be confirmed on other problems.

The bottom Table (b) gives the results of the shaving restricted to the bounds of the domains.Note that **S is useless during the shaving process because constraints are already initialized. The rows $\%_{00}v$ and $\%_{00}r$ respectively correspond to the perthousand of propagated and reentrant variables. Computation times are not mentioned because the solver was heavily instrumented. The results are grouped by answer (550000 SAT, 130000 UNSAT), but SAT answers with only one single propagated variable were ignored. The number of propagated variables is slightly reduced by using _D_, especially for the UNSAT answers. However, the number of reentrant variables is really low which corresponds to a very small possible improvement using another variable ordering. The experiment confirms the observations from [4]: less variables are propagated when the answer is UNSAT than when it is SAT.

6 Conclusion

We have showed that CP solvers must be modified in order to be competitive with the best algorithms for solving shortest path problems. We have identified the modifications of propagation that are efficient for simple but large problems like SPP or RLFAP. Unfortunately, we have also showed that only a few variables are considered twice during a propagation which limits further improvements for the considered problems.

References

[1] T. Balafoutis and K. Stergiou. Exploiting constraint weights for revision ordering in AC algorithms. In *Proc. of ECAI-2008-W31*, 2008.

[2] R. Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[3] F. Boussemart, F. Hemery, and C. Lecoutre. Revision ordering heuristics for the CSP. In *Proc. of CPAI 2004*, pages 29–43, 2004.

[4] F. Boussemart, F. Hemery, C. Lecoutre, and M. Samy-modeliar. Contrôle statistique du processus de propagation de contraintes. In *Proc. of JFPC 2011*, pages 65–74, 2011.

[5] B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Math. Program.*, 73:129–174, 1996.

[6] E.W. Dijkstra. A note on two problems in connection with graphs. *Num. Math.*, 1:269–271, 1959.

[7] Y. Dinitz and R. Itzhak. Hybrid bellman-ford-dijkstra algorithm. Technical Report CS-10-04, Ben-Gurion University of the Negev, 2010.

[8] R.J. Wallace and E.C. Freuder. Ordering heuristics for arc consistency algorithms. In *Proc. of Canadian AI 1992*, pages 163–169, 1992.