Path-Consistency: When Space Misses Time

Assef Chmeiss and Philippe Jégou

LIM - URA CNRS 1787
CMI - Université de Provence
39, rue Joliot Curie - 13453 Marseille Cedex 13 - FRANCE
{jegou,chmeiss}@lim.univ-mrs.fr

Abstract

Within the framework of constraint programming, particulary concerning the Constraint Satisfaction Problems (CSPs), the techniques of preprocessing based on filtering algorithms were shown to be very important for the search phase. In particular, two filtering methods have been studied, these methods exploit two properties of local consistency: arc- and path-consistency. Concerning the arc-consistency methods, there is a linear time algorithm (in the size of the problem) which is efficient in practice (Bessière, Freuder, & Régin 1995). But the limitations of the arc-consistency algorithms requires often filtering methods with higher order like path-consistency filterings. The best path-consistency algorithm proposed is PC-6, a natural generalization of AC-6 (Bessière 1994) to path-consistency (Chmeiss & Jégou 1995)(Chmeiss 1996). Its time complexity is $O(n^3d^3)$ and its space complexity is $O(n^3d^2)$, where n is the number of variables and d is the size of domains. We have remarked that PC-6, though it is widely better than PC-4 (Han & Lee 1988), was not very efficient in practice, specialy for those classes of problems that require an important space to be run. Therefore, we propose here a new path-consistency algorithm called PC-7, its space complexity is $O(n^2d^2)$ but its time complexity is $O(n^3d^4)$ i.e. worse than that of PC-6. However, the simplicity of PC-7 as well as the data structures used for its implementation offer really a higher performance than PC-6. Furthermore, it turns out that when the size of domains is a constant of the problems, the time complexity of PC-7 becomes, like PC-6, optimal i.e. $O(n^3)$.

Keywords: CSPs, Path-Consistency, PC-2, PC-4, PC-6, clp(FD)

Introduction

Within the framework of constraint programming, particulary concerning the Constraint Satisfaction Problems (CSPs), the techniques of preprocessing phase based on filtering algorithms were shown to be very important for the search phase. In particular, two filtering methods have been studied, these methods exploit two properties of local consistency : arc- and path-consistency. Concerning the arc-consistency methods, there are a linear time algorithms (in the size of the CSP) which are efficient in practice (Bessière 1994)(Bessière, Freuder, & Régin 1995). So, arc-consistency can be considered now as a basic tool in the fields of Constraint Programming and Constraint Reasonning. Nevertheless, the filtering corresponding to arc-consistency is limited. For path-consistency, the associated filtering is more powerful than arc-consistency filtering. The best path-consistency algorithm proposed is PC-6, a natural generalization of AC-6 to path-consistency (Chmeiss & Jégou 1995)(Chmeiss 1996). Its time complexity is $O(n^3d^3)$ and its space complexity is $O(n^3d^2)$, where n is the number of variables and d is the size of domains. Unfortunately, we have remarked that PC-6, though it is widely better than PC-4, is not very efficient in practice, specialy for those classes of problems that require an important space to be run. So, it seems that a such filtering cannot be used in practical applications or to be integrated as a basic tool in constraints solvers (Codognet & Nardiello 1994). A possible way to avoid this problem is in defining new properties of consistency between arc- and path-consistency (see (Bennaceur 1994) or (Berlandier 1995)). In this paper, we consider a different way, trying to optimize path-consistency filtering, proposing a new algorithm.

In (Bessière 1994), Bessière has presented a new algorithm for arc-consistency filtering in CSPs. This algorithm, called AC-6, is based on the principle of minimal support. The worst-case time complexity of AC-6 is $O(ed^2)$ where e is the number of constraints, that is the same complexity as AC-4 (Mohr & Hender-

The authors acknowledge support from the project PRC-IA, France.

son 1986). Nevertheless, Bessière showed that AC-6 is better in practice than AC-4. Moreover, the space complexity of AC-6 is $O(ed)$, that is less than AC-4. In (Chmeiss 1996), this approach has been generalized in applying the principle of minimal support to path-consistency; this algorithm is called PC-6. Like PC-4 (Han & Lee 1988), its worst-case time complexity is $O(n^3d^3)$. Nevertheless, experimentations show that PC-6 is significantly more efficient than PC-2 (Mackworth 1977) and PC-4. Moreover, the space complexity of PC-6 which is $O(n^3d^2)$ allows to handle larger CSPs than PC-4 can since space complexity of PC-4 is $O(n^3d^3)$. Nevertheless, because of the size of the required space, the practical interest of PC-6 seems to be limited.

One of the conclusion of (Chmeiss 1996) indicates a way to optimize path-consistency filterings : to make a compromise between time and space in order to find a new algorithm with real practical efficiency. The algorithm we present in this paper, called PC-7, seems to be able to realize this goal. We relaxed the constraint of time complexity to limit the value of space complexity. PC-7 is based on the principle of minimal support as PC-6, but contrary to PC-6, PC-7 does not record supports. While for each pair of compatible values, PC-6 records exactly $n-2$ minimal supports, PC-7 only computes them when it is necessary. So, space complexity of PC-7 is only $O(n^2d^2)$, which is the size of the list of propagations. It is the best space complexity among all known path-consistency algorithms. As a consequence, the time complexity of PC-7 is worse than the one of PC-6 since it is $O(n^3d^4)$. Nevertheless, the simplicity of the algorithm and its data structures induces a really better practical efficiency for PC-7. Moreover, when the size of domains is a constant for the considered application (it is possible for real life problems), PC-7 is optimal in time complexity, $O(n^3)$, and in space, $O(n^2)$.

This paper is organized as follows. Section 2 recalls some definitions and notations on CSPs, and then presents the principles of constraint propagation based on supports for path-consistency, i.e. algorithms PC-4 and PC-6. Section 3 describes PC-7 while section 4 presents experimental results about the comparison between PC-2, PC-4, PC-6 and PC-7 on random CSPs.

A brief state of the art for path-consistency

Definition 1 *A binary CSP is defined by (X, D, C, R), where X is a set of n variables $\{x_1, \ldots x_n\}$ and D is a set of n domains $\{D_1, \ldots D_n\}$ such that D_i is the set of the possible values for variable x_i. C is a set of e binary constraints where each $C_{ij} \in C$ is a constraint between the variables x_i and x_j defined by its associated relation R_{ij}. So, R is a set of e relations, a binary relation R_{ij} between variables x_i and x_j, being a subset of the Cartesian product of their domain that defines the allowed pairs of values for x_i and x_j (i.e $R_{ij} \subset D_i \times D_j$).*

For the networks of interest here, we require that $(b,a) \in R_{ji} \Leftrightarrow (a,b) \in R_{ij}$. The fact that $(a,b) \in R_{ij}$ will be denoted by $R_{ij}(a,b)$ is true. If there is no constraint between the variables x_i and x_j, we consider the universal relation $R_{ij} = D_i \times D_j$. A CSP may be represented by a constraint graph (X, C) in the form of a network in which nodes represent variables and arcs connect variables that appear in the same constraint. An *instantiation* of the variables in X is an n-tuple $(v_1, v_2, \ldots, v_n)$ representing an assignement of $x_i \in X$ to v_i. A *consistent instantiation* of a network is an instantiation of the variables such that the constraints between variables are satisfied, i.e $\forall i, j / 1 \leq i < j \leq n, C_{ij} \in C \Rightarrow R_{ij}(v_i, v_j)$. A consistent instantiation is also called a *solution*. For a given CSP, the problem is either to find all solutions or one solution, or to know if there exists any solution. This decision problem is known to be NP-complete.

Since CSPs are NP-Complete, many techniques have been defined to restrict the size of search space. For example, preprocessing methods based on network consistency algorithms are of great interest in the field of CSPs. These algorithms are based on consistency properties like arc- and path-consistency. We recall below the definition of path-consistency.

Definition 2 *A pair of variables $\{x_i, x_j\}$ is path-consistent iff $\forall (a,b) \in R_{ij}, \forall x_k \in X$, there exists $c \in D_k$ such that $R_{ik}(a,c)$ and $R_{jk}(b,c)$. A CSP is path-consistent iff $\forall x_i, x_j \in X$ the pair $\{x_i, x_j\}$ is path-consistent.*

While filterings based on arc-consistency removed values from domains if they do not satisfy arc-consistency, filterings based on path-consistency removed pairs of values from relations if they do not satisfy path-consistency. So, if a constraint is not defined between a pair of variables, the universal relation is then considered. As a consequence, the constraint graph can be completed in such cases. For arc and path-consistency filterings, the most efficient algorithms are based on the notion of supports.

In (Mohr & Henderson 1986), the notion of support has been introduced to path-consistency preprocessing method producing the algorithm PC-3, and later PC-4 (Han & Lee 1988). A support is a value that allows a pair of values to be compatible: a pair of values $(a,b) \in R_{ij}$ is supported by the value $c \in D_k$ if

the relations $R_{ik}(a,c)$ and $R_{jk}(b,c)$ hold. So, for path-consistency, it is the deletion of a pair of values that will be propagated. For example, the deletion of (a,c) or (b,c) will be propagated to (a,b). PC-4 needs to represent sets of supports and counters for each pair of values. Moreover, a list of removed pairs of values is maintained to record pairs of value already removed but not yet propagated; this list is denoted *List* in the algorithm PC that implements PC-4.

Algorithm PC;
begin
 Initialization;
 while $List \neq \emptyset$ **do begin** { propagation}
  choose (i,a,k,c) in $List$;
  $List \leftarrow List - (i,a,k,c)$;
  Propagate(i,k,a,c);
  Propagate(k,i,c,a);
 end
end;

The procedure *Initialization* assigns counters, builds sets of supports, and initializes the list *List*. The procedure *Propagate* propagates the deletion of pairs of values $R_{ik}(a,c)$ analysing the neighbourhood of variables x_i and x_k in the constraint graph in order to verify if there is no variable x_j such that the value $a \in D_i$ (respectively $c \in D_k$) is a support for a pair $(b,c) \in R_{jk}$ (respectively for a pair $(b,a) \in R_{ji}$). If such a support does not exist, the new inconsistent pairs of values will be deleted, and then will be inserted in the list *List* in order to be propagated later. This algorithm stops when *List* is empty, but we can optimize it in stoping it as soon as a relation becomes empty (because the CSP cannot verify path-consistency). The worst-case time and space complexity of PC-4 is $O(n^3d^3)$.

Using the principle of minimal suppports introduced by Bessière (Bessière 1994) for arc-consistency, PC-4 has been improved resulting the algorithm PC-6 (Chmeiss 1996). Its time complexity is optimal $O(n^3d^3)$ as PC-4, but it has a better space complexity $O(n^3d^2)$.

If the space complexity of PC-6 is bounded by $O(n^3d^2)$, it is still an important space complexity. For example if $n = 128$ and $d = 8$, required space to run PC-6 will be $2^{27} > 10^8$ space units. Assuming that space needed by PC-6 to run for practical applications is too large, we can try to use PC-2 (Mackworth 1977) that needs "only" $O(n^3+n^2d^2)$ space to be run. Unfortunately, running PC-2 is not obvious. On one hand, its time complexity is $O(n^3d^5)$, and on the other hand, space complexity is still cubic in n. It seems that there are two possible ways to optimize path-complexity filterings: either trying to maintain time optimality of PC-6 with a smaller space complexity, or relax time complexity (but with a real practical efficiency) looking for an optimal space complexity algorithm. In the next section, we propose an algorithm related to the second possibility.

A compromise between time and space

The Algorithm

Our algorithm, PC-7, is based on supports without recording any of them. When a propagation of a removed pair of values is to be processed, instead of looking for the next support from the current one as PC-6 does, PC-7 has to start again the search from the first value of the domains. So, the time complexity of PC-7 will be increased of a factor of d, but this allows us to minimize the required space to the size of the list *List*. The scheme of PC-7 is the same as PC-6 one. The data structures used for PC-7 is just the list of deleted pairs of values and not propagated yet. The inititalization phase consists on checking if there exists at least one support per pair of values (a,b). So, any pair with no support must be deleted and added to the list. Indeed, this initialization phase of PC-7 is exactly the same as for PC-6, except for the construction of the support's sets. Concerning the propagation phase, PC-7 restarts looking for a new support from the first value of the domains which is not the case for PC-6. Three constant time functions are used to handle ordered domains D_i : $First(D_i)$ returns the smallest value in the domain D_i, $Last(D_i)$ returns the last value in the domain D_i and $Next\_value(a, D_i)$ which returns the successor of a in D_i. These functions are used in the following $Withoutsupport$ function which checks if $(a,b) \in R_{ij}$ has a support in D_k :

Function Withoutsupport
 (i,j,k: **integer**; a,b : **values**) : **boolean**;
{This function returns $False$ **if** in D_k there is}
{a support c of $(a,b) \in R_{ij}$ **else** it returns $True$}
begin
 $c \leftarrow$ First(D_k);
 while $c < Last(D_k)$
  and not ($R_{ik}(a,c)$ **and** $R_{jk}(b,c)$) **do**
  $c \leftarrow$ Next_value(c, D_k);
 $Withoutsupport \leftarrow$ **not** ($R_{ik}(a,c)$ **and** $R_{jk}(b,c)$)
end; { Withoutsupport }

The difference between PC-4 or PC-6 and PC-7 can be found in the initialization and propagation procedures. *Propagate* has three parameters versus 4 in PC-6. So, in PC we have to call the *Propagate* procedure as follows : $Propagate(i,k,a)$ and $Propagate(k,i,c)$.

```
Procedure Initialization ;
begin
  List ← ∅;
  for i, j, k = 1 to n : i < j, k ≠ i, k ≠ j do
   for (a, b) ∈ R_ij do begin
     if Withoutsupport(i, j, k, a, b) then begin
       R_ij(a, b) ← 0; R_ji(b, a) ← 0;
       Append(List, (i, a, j, b));
     end
   end;
end; { Initialization }
```

```
Procedure Propagate(in i, k:integer;in a: values);
begin
   for j = 1 to n, j ≠ i, j ≠ k do
     for b ∈ D_j do
       if R_ij(a, b) = 1 then begin
         if Withoutsupport(i, j, k, a, b) then begin
           R_ij(a, b) ← 0; R_ji(b, a) ← 0;
           Append(List, (i, a, j, b));
         end
       end
end; { Propagate }
```

Correctness of PC-7

We do not give here a complete proof of the correcteness of PC-7. The key steps given here are similar to the proof proposed in (Bessière 1994) for AC-6. Assume that $MaxR_{ij}$ denotes relations R_{ij} resulting after a path-consistency filtering. It is clear that during propagation, each relation R_{ij} satisfies $MaxR_{ij} \subseteq R_{ij}$ because a values pair $(a, b) \in R_{ij}$ is deleted iff there is no support in a domain D_k. So, $MaxR_{ij} \subseteq R_{ij}$ is an invariant property of PC-7. Moreover, a second invariant property of PC-7 is: for all i, j, for all $(a, b) \in R_{ij}$, and for all k, there is necessary one value $c \in D_k$ such that $(a, c) \in R_{ik} \cup List$ and $(b, c) \in R_{jk} \cup List$. In $List$, the pair (a, c) (respectively (b, c)) corresponds to (i, a, k, c) (respectively (j, b, k, c)). Consequently, at the end of PC-7, when $List = \emptyset$, we have exactly the same property and therefore all values pairs are consistent, that is satisfy path-consistency. Since for all $i, j, MaxR_{ij} \subseteq R_{ij}$ holds, PC-7 obtains the expected result, that is $MaxR = R$.

Space and time complexity

The space complexity of PC-7 is bounded by the maximum size of the list $List$, i.e. $O(n^2d^2)$ since there is at most n^2d^2 pairs of values. It is clear that the time complexity of the initialization step is $O(n^3d^3)$. Concerning the propagation step, since we have at most n^2d^2 elements in the list, then there is at most $2 \times n^2 \times d^2$ calls to the procedure *Propagate* in which the number of iterations is bounded by nd, i.e. the number of variables multiplied by the number of values in the domains. Finally, the cost of the procedure *Withoutsupport* is bounded by the domain's size d. Consequently, the time complexity of PC-7 is bounded by $O(n^2d^2 \times nd \times d) = O(n^3d^4)$.

We can remark that if the size of the domains d is a constant of the problem (this is possible for some applications), the time complexity of PC-7 becomes $O(n^3)$ i.e the same as for PC-6, so PC-7 complexity is then optimal in space and time.

Experiments

In this section we compare PC-7 with PC-2, PC-4 and PC-6. We have chosen these algorithms on account of their time and space complexity (see Table 1). The choice of PC-2 is motivated by the fact that it has the best space complexity with respect to the other known algorithms. Moreover, its time complexity is also $\Omega(n^3d^3)$. We have conserved PC-4 to show that its efficiency is really lower than that of PC-6.

Algorithm	Time	Space
PC-2	$O(n^3d^5)$	$O(n^3 + n^2d^2)$
PC-4	$O(n^3d^3)$	$O(n^3d^3)$
PC-6	$O(n^3d^3)$	$O(n^3d^2)$
PC-7	$O(n^3d^4)$	$O(n^2d^2)$

Table 1: Complexity of path-consistency algorithms

Our experiments were performed over randomly generated CSPs using the random model proposed in (Hubbe & Freuder 1992). The generator considers four parameters: the number of variables n, the domain size d, the tightness of the constraints t, and the constraint graph density cd. The *constraint tightness* t is the fraction of the combinatorially possible pairs that are not allowed by the constraints between two variables: $t = 1 - \frac{|R_{ij}|}{d^2}$. The *constraint graph density* is a value cd varying between 0 and 1 indicating the fraction of the possible constraints beyond the minimum $n - 1$ (for a connected acyclic graph). Note that if $cd = 1$, the number of constraints is $(n^2 - n)/2$, which corresponds to a complete constraint graph. For each 4-tuples (n, d, t, cd), 20 randomly CSPs were generated. Results reported so far represent the average over the 20 problems for each of the algoritms.

For PC-4 and PC-6, we limited the experiments to CSPs with $n = 16$ and $d = 8$, but for PC-2 and PC-7 experiments on problems with $n = 32$ and $d = 8$ were performed. In fact, we have remarked that it was impossible for PC-4 to be run over problems with

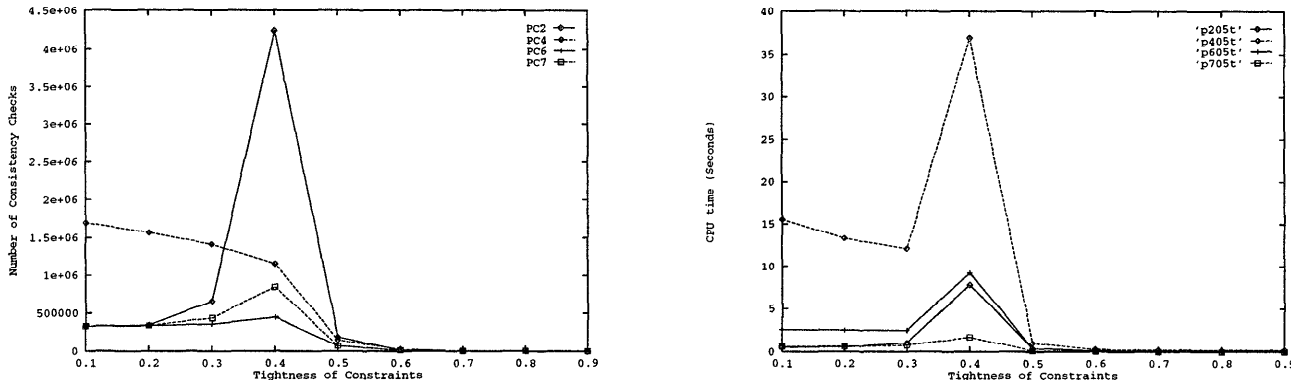


Figure 1: Consistency checks and CPU time for $n = 16, d = 8$ and $cd = 0.5$

$n = 32$ and $d = 8$ on our computer because of the required memory space. Moreover, for PC-6 the real time execution was widely superior than the CPU time, some hours in place of some minutes for some classes of problems. The last point can be explained by the fact that the management of the required memory space leads to an excessive usage of the secondary memory.

We have chosen two measures of comparison, on the one hand the number of consistency checks, on the other hand the CPU execution time. Figures provide results in terms of the number of consistency checks as well as the CPU time. In each case, the *x-axis* represents the constraint tightness; it varies from 0.1 to 0.9 with a step of 0.1.

Figure 1 presents comparisons between PC-2, PC-4, PC-6 and PC-7 for a CSP's with $n = 16$, $d = 8$ and $cd = 0.5$. Concerning these classes of CSPs, mentionned above, it is clair that PC-6 realizes the smallest number of consistency checks. By contrast, PC-7 is the more efficient algorithm for CPU time as a measure of performance. For the number of consistency checks, we remark that PC-7 realizes as much as PC-6, this is due to the fact that PC-7 always restarts from the first value of domains (during the search for a support).

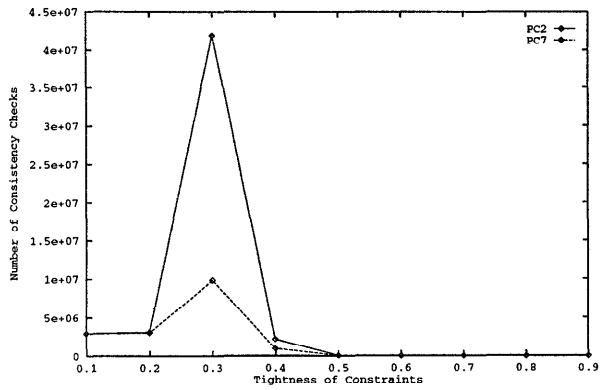
Figure 2 compares PC-2 with PC-7 for a CSPs with $n = 32$ and $d = 8$. Here, results taken on account concern CSP's the graph density of which $cd = 0.5$. Figure concerning CPU time shows that PC-7 outperforms PC-2, except for $t = 0.1$ both algorithms have almost the same performances. Also, PC-7 outperforms PC-2 for the number of consistency checks as a measure of efficiency. We note that for some problems in this classes, PC-6 needs 5 to 6 minutes CPU time (some hours real time).

PC-7 was also tested for CSPs with $n = 64$ and $d = 8$ for which CPU time was between 1 and 2 minutes, then for CSPs with $n = 128$ and $d = 8$ for which CPU time varied from 5 to 20 minutes.

Observing these experiments, we can conclude that PC-7 is the best path-consistency filtering algorithm. Specially it allows us to handle CSPs of important size while other algoritms failed. The fact that PC-7 outperforms PC-2 can be explained naturally by considerations lied to the theoretical complexity in time and space. Contrary to the theoretical evaluation of time complexity, the surprise comes from the efficiency of PC-7 versus PC-6 which could be explained by the time lost by PC-6 in treating the used data structures which require handling the lists S_{iakc} containing pairs (j, b) which are represented by doubly linked lists. Moreover, these elements (j, b) must be linked by pointers to $(i, a) \in S_{jbkc}$. If a such data structures leads to an optimal theoretical time complexity, they increase the CPU time because of the required number of operations for each propagation step, which is widely more important than the one of PC-7. The multiplicative hidden constant of PC-6 seems to be widely greater than PC-7 one. *A contrario*, PC-7 depends only on the list of deleted pairs of values, already used by PC-6, and another point is that the implementation of PC-7 is very simple. These different points enable PC-7 to has a remarkable efficiency in time and a weak memory usage, finally it is very easy to implement.

Conclusion

In this paper, we presented a new algorithm to achieve path-consistency in constraints networks; this algorithm is called PC-7. Space complexity of PC-7 is $O(n^2d^2)$, that is currently the best space complexity for algorithms to achieving path-consistency. Time complexity of PC-7 is $O(n^3d^4)$, that is slightly more than the one of PC-6. Nevertheless, the simplicity of



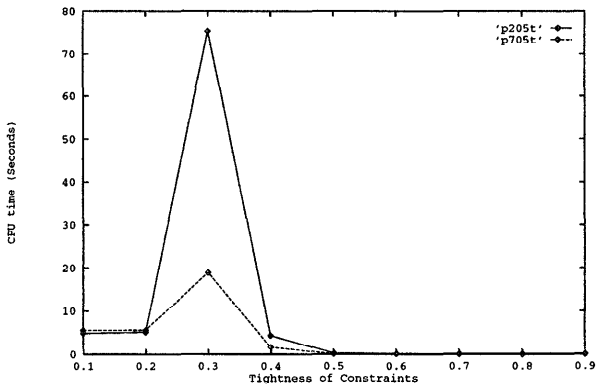


Figure 2: Consistency checks and CPU time for $n = 32, d = 8$ and $cd = 0.5$

the algorithm and the simplicity of its data structure allow PC-7 to be really efficient in practice, i.e. the CPU time required to run is really less than for PC-6. Moreover, for cases such that the size of domains is a constant parameter of problems (this fact frequently appears in real life applications), PC-7 becomes theoreticaly optimal for time complexity, that is $O(n^3)$ like PC-6. Finally, it may be interesting to see if this approach can be applied to the temporal reasoning as described in (Allen 1983).

Refernces

Allen, J.F. 1983. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832-843.

Bennaceur, H. 1994. Partial Consistency for Constraint Satisfaction Problems. In Proceedings of ECAI'94, Amsterdam, The Netherelands, 120-124.

Berlandier, P. 1995. Filtrage de problèmes par consistance de chemin restreinte. In *Revue d'Intelligence Artificielle* 9(1):225-238.

Bessière, C. 1994. Arc-Consistency and Arc-Consistency Again. In *Artificial Intelligence* 65:179-190.

Bessière, C.; Freuder, E.C.; and Régin, J.C. 1995. Using Inference to Reduce Arc Consistency Computation. In Proceedings of the IJCAI'95, 592-598.

Chmeiss, A. 1996. Sur la consistance de chemin et ses formes partielles. In the Actes du Congrès AFCET-RFIA'96, France, 212-219.

Chmeiss, A., and Jégou, Ph. 1995. Partial and Global Path Consistency Revisited, Technical Report, 120.95, Laboratoire d'Informatique de Marseille, France.

Codognet, P., and Nardiello, G. 1994. Path Consistency in clp(FD), *Proceedings 1st International Conference on Constraints in Computational Logics, Munchen, Germany, Lecture Notes in Computer Science*, vol. 845 (1994) 201-216.

Han, C.C., and Lee, C.H. 1988. Comment on Mohr and Henderson's Path Consistency Algorithm. In *Artificial Intelligence* 36:125-130.

Hubbe, P., and et Freuder, E.C. 1992. An Efficient Cross-Product Representation of the Constraint Satisfaction Problem Search Space. In Proceedings of AAAI'92, 421-427.

Mackworth, A.K. 1977. Consistency in networks of relations. In *Artificial Intelligence* 8:99-118.

Mackworth, A.K., and Freuder, E.C. 1985. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. In *Artificial Intelligence* 25:65-74.

Mohr, R., and Henderson, T.C. 1986. Arc and Path Consistency Revisited. In *Artificial Intelligence* 28:225-233.