Some Practicable Filtering Techniques for the Constraint Satisfaction Problem

Romuald Debruyne and **Christian Bessière**
LIRMM (UMR 5506 CNRS)
161 rue Ada
34392 Montpellier Cedex 5 - France
Email: {debruyne, bessiere}@lirmm.fr

Abstract

Filtering techniques are essential in order to efficiently solve constraint satisfaction problems (CSPs). A blind search often leads to a combinatorial explosion, the algorithm repeatedly finding the same local inconsistencies. Maintaining a local consistency can strongly reduce the search effort especially on hard and large problems. A good illustration are the good time performances on such problems of maintaining arc consistency during search compared to forward checking which maintains a lower level of local consistency. On the one hand, arc consistency (2-consistency) is the most used filtering technique because it cheaply removes some values that cannot belong to any solution. On the other hand, other k-consistencies (k≥3) have important space and time requirements because they can change the set of constraints. They can only be used on very small CSPs. Thus, in this paper, we study and compare the filtering techniques that are more pruningful than arc consistency while leaving unchanged the set of constraints. As arc consistency, they only remove inconsistent values in the domains, and so, can deal with large CSPs.

1 Introduction

Constraint satisfaction problems (CSPs) are used more and more to solve combinatorial problems arising in artificial intelligence. They involve finding an assignment of values to variables subject to constraints. This is a NP-hard task. So, the time needed by a primitive backtrack search to find such an assignment will probably be strongly affected by thrashing because it will fall many times into the same local inconsistencies. Thus filtering techniques are used to remove some local inconsistencies before or during the search. To find the suitable level of consistency to achieve we have to both consider the pruning efficiency of the filterings involved, and their time and space complexities. Obviously, the overhead caused by removing local inconsistencies has to be outweighted by its gain, which can change depending on the problem to solve. On very easy and very small CSPs, filtering techniques are useless because we can quickly find a solution or prove the inconsistency. But it can be advantageous to use an algorithm that wastes a few seconds on easy CSPs if it can save many minutes (hours ?) on hard problems. A good illustration is the efficiency of maintaining arc consistency during search (MAC) on hard CSPs [Bessière and Régin, 1996]. On these problems, MAC can outperform forward checking (FC) which maintains a lower level of local consistency. The harder a CSP is, the more useful filtering techniques are. So, maintaining a local consistency during search is essential in order to deal with large and hard problems. Obviously, whatever the level of local consistency maintained may be, some pathological CSPs which are hard to solve can be found. But the more pruningful the filtering technique is, the more seldom these CSPs are.

Arc consistency (AC) filtering is widely used on binary CSPs because it removes some values that cannot belong to any solution with a low space and time complexity. On the other hand, strong path consistency can remove more inconsistencies but is never used because of its huge complexity and because it changes the structure of the problem by adding constraints. Indeed, path consistency algorithms remove compatible pairs of values that cannot be extended to a consistent instantiation including any third variable. If a path inconsistent pair of values between two independent variables is found, it is removed from the corresponding universal relation and so the structure of the constraint graph is changed. This is a real drawback, especially when we deal with sparse problems because the resulting constraint graph is often complete. The second important drawback is the lost of semantics. In real applications, a constraint is rarely represented by a boolean matrix or a set of compatible pairs of values. It is represented by a predicate which has a particular semantics ($\neq$, $\leq$, ...), and this semantics is lost when some allowed pairs of values are removed. This leads to the generation of the boolean matrix, and so to space intractability.

The aim of this paper is to study the filtering techniques that can be used in practice. So, we only care about those that do not change the constraints and that do not require to have the constraints in extension, i.e. those that only remove values in the variable domains.

2 Definitions and notations

A *binary CSP* $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a set $\mathcal{X} = \{ i, j, \ldots\}$ of *n variables*, each taking value in its respective finite *domain* D_i, D_j, ... elements of $\mathcal{D}$, and a set $\mathcal{C}$ of e binary constraints. d is the size of the largest domain.

A *binary constraint* C_{ij} is a subset of the cartesian product $D_i \times D_j$ that denotes the compatible pairs of values for i and j. We note $C_{ij}(a, b) = true$ to specify that $((i, a), (j, b)) \in C_{ij}$. We then say that (j, b) is a *support* of (i, a) on C_{ij}. With each CSP we associate a *constraint graph* in which nodes represent variables and arcs connect pairs of variables which are constrained explicitly. The *neighborhood* of i is the set of variables linked to i in the constraint graph. A domain $\mathcal{D}' = \{D'_i, D'_j, \ldots\}$ is a *sub-domain* of $\mathcal{D} = \{D_i, D_j, \ldots\}$ if $\forall i, D'_i \subseteq D_i$. An *instantiation* of a set of variables S is an indexed set of values $\{I_j\}_{j\in S}$ s.t. $\forall j \in S \quad I_j \in D_j$. An instantiation I of S satisfies a constraint C_{ij} if $\{i, j\} \not\subseteq S$ or $C_{ij}(I_i, I_j)$ is true. An instantiation is *consistent* if it satisfies all the constraints. A *solution* of $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a consistent instantiation of $\mathcal{X}$. A value (i, a) is *consistent* if there is a solution I such that $I_i = a$, and a CSP is consistent if it has at least one solution. In the following we denote by $P\mid_{D_i=\{a\}}$ the CSP obtained by restricting D_i to $\{a\}$ in P.

3 Overview on filtering techniques

Filtering techniques may detect inconsistency or remove some local inconsistencies in order to make the search easier. They look for partial instantiations that cannot be extended to a solution and remove them.

Fig. 1 presents the properties corresponding to the filtering techniques studied. The best AC algorithms are AC-6 [Bessière, 1994] with the $O(ed^2)$ optimal worst-case time complexity and $O(ed)$ worst-case space complexity, and AC-7 [Bessière *et al.*, 1995], which deals with the bidirectionality of constraints to improve AC-6.

k-consistency, i.e. $(k-1, 1)$-consistency [Freuder, 1985], removes the consistent instantiations of length $k-1$ that cannot be extended to a consistent instantiation including any additional k^{th} variable. The time and space complexity of k-consistency algorithms is polynomial in k. Furthermore, if $k \geq 3$, k-consistency changes the structure of the problem. So, only 2-consistency (i.e. arc consistency) can be used in practice. However, path consistency (3-consistency) being widely studied, we will compare strong path consistency (enforcing both arc and path consistency) with the other filtering techniques removing values. The best PC algorithms are PC-5 [Singh, 1995] with $O(n^3d^3)$ worst-case time complexity and $O(n^3d^2)$ worst-case space complexity, and PC-8 [Chmeiss and Jégou, 1996] which requires smaller space ($O(n^2d)$ [1]) but with $O(n^3d^4)$ worst-case time complexity.

Arc consistency being a low level of consistency and path consistency leading to important drawbacks, Berlandier proposed the restricted path consistency (RPC) in [Berlandier, 1995]. This filtering technique removes the arc inconsistent values but in addition it checks the path consistency of the pairs of values involving a weakly supported value. The idea is that if a value (i, a) has an unique support (j, b) on C_{ij}, it may be advantageous to check the path consistency of $((i, a), (j, b))$ because the deletion of this pair leads to the arc inconsistency of (i, a). So, RPC uses path inconsistency to remove values but does not delete any pair of values. This allows to remove more values than AC while avoiding the drawbacks of PC. The RPC algorithm in [Berlandier, 1995] has $O(end^3)$ worst-case time complexity and $O(end + ed^2)$ worst-case space complexity. But this algorithm is not optimal in time.

- A binary CSP is **(i, j)-consistent** iff $\forall k \in \mathcal{X}$, $D_k \neq \emptyset$, and any consistent instantiation of i variables can be extended to a consistent instantiation including any j additional variables.
- A domain D_i is **arc consistent** iff, $\forall a \in D_i$, $\forall j \in \mathcal{X}$ s.t. $C_{ij} \in \mathcal{C}$, there exists $b \in D_j$ s.t. $C_{ij}(a, b)$. A CSP is **arc consistent** ((1, 1)-consistent) iff $\forall D_i \in D$, $D_i \neq \emptyset$ and D_i is arc consistent.
- A pair of variables (i, j) is path consistent iff $\forall (a, b) \in C_{ij}$, $\forall k \in \mathcal{X}$, there exists $c \in D_k$ s.t. $C_{ik}(a, c)$ and $C_{jk}(b, c)$. A CSP is **path consistent** ((2, 1)-consistent) iff $\forall i, j \in \mathcal{X}$, (i, j) is path consistent.
- A binary CSP is **strongly path consistent** iff it is node-consistent, arc consistent and path consistent.
- A binary CSP is **path inverse consistent** iff it is (1, 2)-consistent i.e. $\forall (i, a) \in D$ $\forall j, k \in \mathcal{X}$ s.t. $j \neq i \neq k \neq j$, $\exists (j, b) \in D$ and $(k, c) \in D$ s.t. $C_{ij}(a, b) \wedge C_{ik}(a, c) \wedge C_{jk}(b, c)$
- A binary CSP is **neighborhood inverse consistent** iff $\forall (i, a) \in D$, (i, a) can be extended to a consistent instantiation including the neighborhood of i.
- A binary CSP is **restricted path consistent** iff $\forall i \in \mathcal{X}$, D_i is a non empty arc consistent domain and, $\forall (i, a) \in D$, for all $j \in \mathcal{X}$ s.t. (i, a) has an unique support b in D_j, for all $k \in \mathcal{X}$ linked to both i and j, $\exists c \in D_k$ s.t. $C_{ik}(a, c) \wedge C_{jk}(b, c)$.
- A binary CSP P is **singleton arc consistent** iff $\forall i \in \mathcal{X}$, $D_i \neq \emptyset$ and $\forall (i, a) \in D$, $P\mid_{D_i=\{a\}}$ has an arc consistent sub-domain.
- A binary CSP P is **singleton restricted path consistent** iff $\forall i \in \mathcal{X}$, $D_i \neq \emptyset$ and $\forall (i, a) \in D$, $P\mid_{D_i=\{a\}}$ has a restricted path consistent sub-domain.

Figure 1: The local consistencies studied.

[1] Remark that we still need a $O(n^2d^2)$ data structure for the constraints representation.

Another way to avoid the drawbacks of k-consistency is to consider the "opposite" property. k-inverse consistency (i.e. $(1, k-1)$-consistency [Freuder, 1985]) removes the values that cannot be extended to a consistent instantiation including any $k-1$ additional variables. The constraint graph is unchanged by this filtering technique and the space required to enforce it is linear. However, as for k-consistency, the worst-case time complexity is polynomial in k, and so, inverse consistency quickly becomes prohibitive when k grows. The first level removing more values than arc consistency is path inverse consistency ($k = 3$). Obviously, the more k is important, the more pruningful k-inverse consistency is, but we cannot reasonably remove the n-inverse inconsistent values. A good compromise is to make sure that each value can be extended to a consistent instantiation including its neighborhood. This filtering technique called neighborhood inverse consistency (NIC)[Freuder and Elfe, 1996] adapts the level of local inverse consistency to the number of constraints involving the variable. This is an efficient method on sparse problems but its exponential worst-case time complexity cannot guarantee a reasonable cpu time.

Singleton consistency is a new class of filtering techniques. It is based on the fact that if a value $(i,\ a)$ is consistent, then the CSP obtained by restricting the domain of i to the singleton $\{a\}$ is consistent. So, if $P\ |_{D_i=\{a\}}$ is inconsistent, we can delete $(i,\ a)$. Obviously finding if $P\ |_{D_i=\{a\}}$ is consistent may be costly. Therefore, we only check if a local consistency holds in $P\ |_{D_i=\{a\}}$. For example, a CSP is singleton arc consistent if it does not have any empty domain, and $\forall(i,\ a) \in \mathcal{D}$, $P\ |_{D_i=\{a\}}$ has a non empty arc consistent sub-domain. Any AC algorithm can be used to determine whether achieving arc consistency in $P\ |_{D_i=\{a\}}$ can yield a domain wipe out, but a lazy approach (such as LAC7 [Schiex *et al.*, 1996]) is sufficient. The singleton consistency can be used with any algorithm removing inconsistent values as those studied in this paper. If the local consistency can be enforced in a polynomial time in $P\ |_{D_i=\{a\}}$, the corresponding singleton consistency has a polynomial worst-case time complexity too.

Since AC algorithms can incrementally propagate the deletion of a value, we propose the algorithm of Fig. 2. It can easily be adapted to enforce a singleton consistency with a stronger pruning efficiency, such as SRPC or SPIC.

```
Procedure SingletonAC( P )
  P ← AC( P );
  Repeat
    Changed ← false;
    for all (i, a) ∈ D do
      if the propagation of the deletion of {b ∈ D_i s.t. b ≠ a}
      in P leads to a wipe-out(*) then
        D_i ← D_i\{a};
        Propagate the deletion of (i, a) in P to achieve AC;
        Changed ← true;
  until Changed=false;

(*) This test only determines if achieving AC in P |_{D_i={a}} leads to a
domain wipe out.
```

Figure 2: A SAC algorithm.

4 Pruning efficiency

4.1 Qualitative study

In order to compare the pruning efficiency of the local consistencies we study, we introduce the following transitive relation. We say that a local consistency LC is *stronger* than another local consistency LC' if in any CSP in which LC holds, LC' holds too. For example and by definition, RPC is stronger than AC. Therefore, a RPC algorithm removes at least all the arc inconsistent values. Consequently, the domain of RPC(P) is a sub-domain of the domain of AC(P) whatever P may be. A local consistency LC is *strictly stronger* than another local consistency LC' if LC is stronger than LC' and there is at least one CSP in which LC' holds and LC does not hold.

Theorem 4.1 *If* $|\ \mathcal{X}\ |\geq 3$*, path inverse consistency is stronger than restricted path consistency.*

Proof. First, let us show that PIC is stronger than arc consistency if $|\ \mathcal{X}\ |\geq 3$. For all $(i,\ a) \in D$ and $C_{ij} \in \mathcal{C}$, let k be any third variable. PIC implies the existence of $(j,\ b) \in D$ and $(k,\ c) \in D$ such that $C_{ij}(a,\ b)$, $C_{ik}(a,\ c)$ and $C_{jk}(b,\ c)$. So, for all $(i,\ a) \in D$ and $C_{ij} \in \mathcal{C}$, there exists a support $(j,\ b) \in D$. The second property of RPC is a trivial consequence of PIC. If PIC holds, any value $(i,\ a) \in D$ can be extended to a consistent instantiation including any two other variables. So, if $(i,\ a)$ has an unique support $(j,\ b)$ on C_{ij} we are sure that whatever k is, $(i,\ a)$ can be extended to a consistent instantiation I including j and k s.t. $I_j = b$. ■

Theorem 4.2 *If* $|\ \mathcal{X}\ |\geq 3$ *and the constraint graph is connected, neighborhood inverse consistency is stronger than path inverse consistency.*

Proof. see [Debruyne and Bessière, 1997] ■

Theorem 4.3 *If* $|\ \mathcal{X}\ |\geq 3$*, singleton arc consistency is stronger than path inverse consistency.*

Proof. Suppose that a singleton arc consistent value $(i,\ a)$ is not path inverse consistent. Let j and k be two variables s.t. $(i,\ a)$ cannot be extended to a consistent instantiation including j and k. $(i,\ a)$ being singleton arc consistent, let D' be an arc consistent sub domain of $P\ |_{D_i=\{a\}}$ and b a value in D'_j. There exists a value $c \in D'_k$ supporting $(j,\ b)$ on C_{jk}. Since b and c are in an arc consistent sub-domain of $P\ |_{D_i=\{a\}}$, $C_{ij}(a,\ b)$ and $C_{ik}(a,\ c)$. We have therefore a contradiction since $(a,\ b,\ c)$ is a consistent instantiation of $\{i,\ j,\ k\}$. ■

Theorem 4.4 *Singleton restricted path consistency is stronger than singleton arc consistency.*

Proof. Trivial since restricted path consistency is stronger than arc consistency. ∎

Theorem 4.5 *Strong path consistency is stronger than singleton arc consistency.*

Proof. We have to prove that if a value $(i,\ a)$ is singleton arc inconsistent, a strong path consistency algorithm removes it. First, let us show that if a value $(j,\ b)$ is arc inconsistent in $P\ |_{D_i=\{a\}}$ and $(a,\ b) \in C_{ij}$ then $(a,\ b)$ is path inconsistent w.r.t. P (by induction on the number of arc inconsistent values deleted before $(j,\ b)$). If $(j,\ b)$ is the first value deleted by an AC algorithm because it has no support in D_k, $(a,\ b)$ cannot be extended to a consistent instantiation of $\{i,\ j,\ k\}$. If after the deletion of m arc inconsistent values, $(j,\ b)$ has to be removed because it no longer has a support in D_k, all the supports of $(j,\ b)$ on C_{jk} have already been deleted in $P\ |_{D_i=\{a\}}$. Thus, by induction hypothesis $(a,\ c)$ is path inconsistent in P for all $(k,\ c)$ compatible with $(j,\ b)$. So, $(a,\ b)$ is path inconsistent in P.

According to this lemma, if a value $(i,\ a)$ is singleton arc inconsistent because AC clears the domain of j in $P\ |_{D_i=\{a\}}$, path consistency removes $(a,\ b)$ from C_{ij} for all $b \in D_j$ and AC deletes $(i,\ a)$. ∎

SAC and strong PC are strongly related. In [Mc Gregor, 1979], Mc Gregor proposed a strong path consistency algorithm that can be modified to achieve SAC. This algorithm repeatedly enforces arc consistency on $P\ |_{D_i=\{a\}}$ for all $(i,\ a) \in D$. If a wipe out occurs, $(i,\ a)$ is removed, otherwise $(a,\ c)$ is deleted from C_{ik} for all $(k,\ c)$ removed by arc consistency in $P\ |_{D_i=\{a\}}$. The algorithm stops when it can no longer detect inconsistent values or path inconsistent pairs of values. A SAC algorithm is obtained by omitting deletions of pairs of values in this algorithm. So, strong PC removes more values only when the propagation of the deletions of pairs of values leads to some additional value deletions.

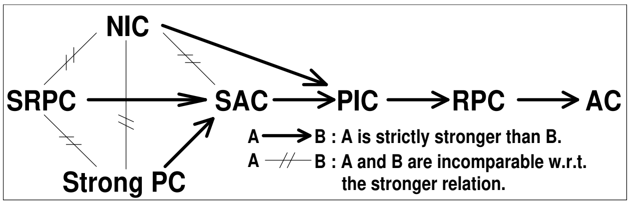

Figure 3: Relations between the local consistencies.

Fig. 3 shows the relations between the filtering techniques studied. A continuous arrow from A to B means that the local consistency A is strictly stronger than B. There is a crossed line between A and B if A and B are incomparable w.r.t. the "stronger" relation. If A is not stronger than B (B is strictly stronger than A or A and B are incomparable), a CSP in which A holds and B does not hold can be found in [Debruyne and Bessière, 1997]. The "stronger" relation does not induce a total ordering. For instance, NIC and SRPC are incomparable.

4.2 Experimental evaluation

The study in 4.1 gives no quantitative information. A local consistency can be stronger than another while performing few additional value deletions. Furthermore, a local consistency LC can remove more values than another local consistency LC' on most of the CSPs while being incomparable with LC' because of some particular CSPs.

Fig. 4 shows on which random CSPs the local consistencies are useful to detect inconsistency. The CSP generator involves four parameters : n the number of variables, d the common size of all the initial domains, p_1 the proportion of constraints in the network (p_1=1 corresponds to the complete graph) and p_2 the proportion of forbidden pairs of values in a constraint (the tightness)[2]. The problems of Fig. 4 have 20 variables, 10 values in each domain, and for each possible pair $(p_1,\ p_2)$, 500 problems were generated. The continuous line shows for each density the minimal tightness for which the 500 random CSPs are all inconsistent. For each filtering technique, Fig. 4 presents the pairs (p_1, p_2) for which the given local inconsistency is detected on all the generated problems. For example, for AC the limit is 0.69 at density 0.5. This means that for a smaller tightness, there is at least one of the 500 problems on which AC does not remove all the values.

Restricted path consistency is very close to PIC although it requires much less computational effort. To achieve RPC, we have to try to extend a value $(i,\ a)$ to a consistent 3-tuple involving $\{i, j, k\}$ only if $(i,\ a)$ has an unique support in D_j. It is sufficient to remove most of the values deleted by PIC. As said in section 4.1, SAC is close to strong path consistency which removes more inconsistent values only when there is an important propagation of pairs of values removed. Although SRPC and strong PC are incomparable, SRPC can detect more inconsistent values on most of the CSPs. All these polynomial filterings have a similar behavior. They show difficulties in inconsistency detection on dense problems with loose constraints. It was predicted in [van Beek, 1994]. NIC has a very different behavior. On very dense CSPs, checking if a value can be extended to a consistent instantiation including its neighborhood is close to checking the consistency of the value. However, on the generated problems with a density between 0.2 and

[2]See [Frost *et al.*, 1996] for a complete description of the uniform random CSP generator we used.

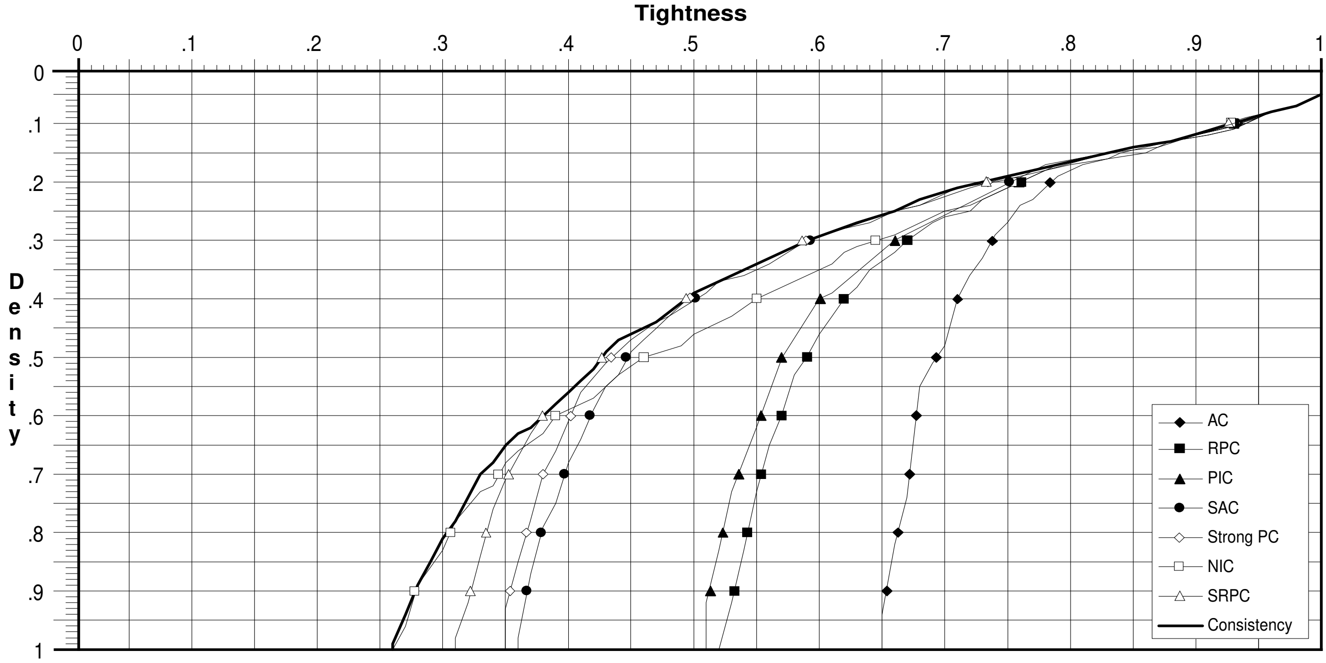

Figure 4: Evaluation of inconsistency detection on random problems with n=20 and d=10.

0.55, NIC prunes fewer values than SAC, strong PC, and SRPC.

5 Time efficiency

The same generator is used to evaluate the cpu time required to enforce the local consistencies. All the generated problems have 100 variables and 10 values in each initial domain. Fig. 5 shows the results for both CSPs having relatively few constraints, and dense CSPs. The algorithms used are AC-6, the RPC algorithm in [Berlandier, 1995], the PIC and NIC algorithms in [Freuder and Elfe, 1996] (NIC using forward checking with minimal domain heuristic), a strong path consistency algorithm based on PC-8 and AC-6, the singleton arc consistency algorithm of Fig. 2 based on AC-6, and a SRPC algorithm based on the RPC algorithm in [Berlandier, 1995]. All these algorithms have been modified to stop as soon as a wipe out occurs. For each tightness, 15 instances were generated and Fig. 5 presents mean values. For some problems at density 0.25, NIC is very time consuming. Therefore, we set a 2 hours limit. No result on NIC is given if one of the 15 instances required more than 2 hours.

Although NIC and SRPC remove more values, strong path consistency is the most expensive filtering technique on most of the generated problems. The consequences of the exponential worst case time complexity of NIC can be observed at density 0.25. For a tightness between 0.18 and 0.30, some problems were not finished after 2 hours. NIC cannot avoid the combinatorial explosion, and so, is unusable on these problems. In addition, NIC is the less efficient (if we exclude strong PC) on CSPs involving many constraints or having tight constraints. The most promising filtering techniques are SAC and RPC. SAC removes most of the strong path inconsistent values while requiring less cpu time than PIC. The RPC algorithm used is non optimal and does not propagate the value deletions as soon as possible. This explains the relatively bad time performances for tightness between 0.5 and 0.8 at density 0.25. However, RPC may remove much more values than AC.

6 Conclusion

In this paper we studied the filtering techniques that are practicable on large CSPs, i.e. those that do not change the constraints. When the aim is detecting inconsistency, SRPC and NIC are the most powerful local consistencies. However, they are expensive in time and the exponential worst case time complexity of NIC makes it unusable on some dense CSPs. Two promising local consistencies are SAC and RPC, SAC having a good cpu time to number of deleted values ratio, and RPC requiring little additional cpu time compared to AC while removing most of the path inverse inconsistent values.

References

[Berlandier, 1995] P. Berlandier. Improving Domain Filtering using Restricted Path Consistency. In *Proceedings of the IEEE CAIA-95*, Los Angeles CA, 1995.

[Bessière and Régin, 1996] C. Bessière and J.C. Régin. MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ ?) on Hard Problems. In *Proceedings of CP-96*, pages 61–75, Cambridge, MA, USA, 1996.

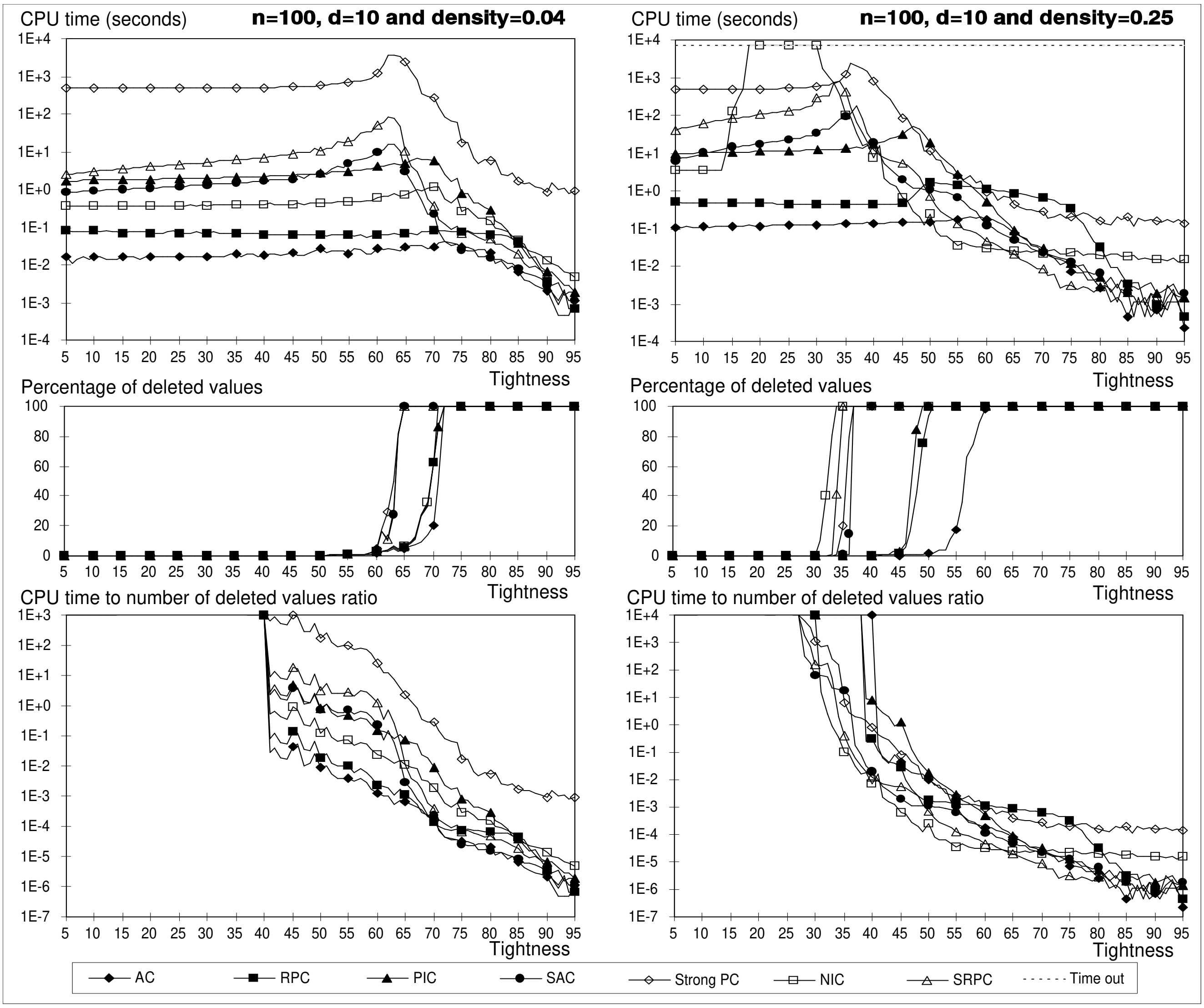

Figure 5: Comparison of the filtering techniques on random CSPs with n=100 and d=10.

[Bessière *et al.*, 1995] C. Bessière, E.C. Freuder, and J.C. Régin. Using inference to reduce arc-consistency computation. In *Proceedings of the 14th IJCAI*, Montréal, Canada, 1995.

[Bessière, 1994] C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence 65*, pages 179–190, 1994.

[Chmeiss and Jégou, 1996] A. Chmeiss and P. Jégou. Two New Constraint Propagation Algorithms Requiring Small Space Complexity. In *Proceedings of the 8th IEEE ICTAI*, pages 286–289, Toulouse, France, 1996.

[Debruyne and Bessière, 1997] R. Debruyne and C. Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. Technical Report 97035, LIRMM, Montpellier, France, 1997.

[Freuder and Elfe, 1996] E. Freuder and C.D. Elfe. Neighborood Inverse Consistency Preprocessing. In *Proceedings of AAAI-96*, pages 202–208, Portland, Oregon, 1996.

[Freuder, 1985] E. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4):755–761, 1985.

[Frost *et al.*, 1996] D. Frost, C. Bessière, R. Dechter, and J.C. Régin. Random Uniform CSP Generators. http://www.ics.uci.edu/~dfrost/csp/generator.html, 1996.

[Mc Gregor, 1979] J.J. Mc Gregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences 19*, pages 229–250, 1979.

[Schiex *et al.*, 1996] T. Schiex, J.C. Régin, C. Gaspin, and G. Verfaillie. Lazy arc consistency. In *Proceedings of AAAI-96*, pages 216–221, Portland, Oregon, 1996.

[Singh, 1995] M. Singh. Path Consistency Revisited. In *Proceedings of the 7th IEEE ICTAI*, Washington D.C., 1995.

[van Beek, 1994] P. van Beek. On the inherent level of local consistency in constraint networks. In *Proceedings of AAAI-94*, pages 368–373, Seattle WA, 1994.