A Simple Way to Improve Path Consistency Processing in Interval Algebra Networks

Christian Bessière
LIRMM (UMR 5506 CNRS),
161 rue Ada,
34392 Montpellier cedex 5, France
Email: bessiere@lirmm.fr

Abstract

Reasoning about qualitative temporal information is essential in many artificial intelligence problems. In particular, many tasks can be solved using the interval-based temporal algebra introduced by Allen (All83). In this framework, one of the main tasks is to compute the transitive closure of a network of relations between intervals (also called *path consistency* in a CSP-like terminology). Almost all previous path consistency algorithms proposed in the temporal reasoning literature were based on the constraint reasoning algorithms PC-1 and PC-2 (Mac77). In this paper, we first show that the most efficient of these algorithms is the one which stays the closest to PC-2. Afterwards, we propose a new algorithm, using the idea "one support is sufficient" (as AC-3 (Mac77) does for arc consistency in constraint networks). Actually, to apply this idea, we simply changed the way *composition-intersection* of relations was achieved during the path consistency process in previous algorithms.

Introduction

Reasoning about qualitative temporal information is essential in many artificial intelligence applications such as natural language processing, diagnosis, and planning. The interval algebra introduced by Allen (All83) is among the most suitable frameworks for such applications. However, qualitative reasoning in interval algebra (or IA) networks (i.e. networks of relations of the Allen's algebra constraining pairs of variables that represent intervals) is inherently intractable, as shown by Vilain and Kautz (VK86). Allen introduced a polynomial time algorithm which computes the transitive closure of an IA network. Transitive closure is also known as path consistency. Since the consistency problem is intractable in IA networks, we can expect path consistency to suffer from incompleteness. One can easily find, indeed, path consistent IA networks that are not consistent (All83). In fact, finding a consistent scenario (a consistent assignment of atomic relations to every edge in the network) is the "easiest" way to prove the consistency of an IA network. But, because of intractability, finding a consistent scenario will need a backtrack-based procedure.

It has widely been shown by the constraint reasoning community that a backtracking algorithm which does not incorporate some look-ahead processing is almost always subject to thrashing ((SG95), (HE80), (Pro93)), and then is not able to produce a solution in reasonable time. In constraint satisfaction problems (CSPs), different forms of weak partial consistency can be computed at each node of the search tree. The trick is to find the good compromise: pruning as much local inconsistencies as possible without slowing down too much the tree search. Usually, the amount of preprocessing performed is somewhere between nothing and arc consistency (Nad89).

In IA networks, it is not difficult to choose which amount of partial consistency will fit the best since all the partial consistency techniques weaker than path consistency are pruningless (LR92b). Arc consistency for example does not remove anything in IA networks because they are already arc consistent. Path consistency is then the only technique usable as a look-ahead scheme in a backtracking procedure (higher level partial consistencies are surely too expensive: they produce higher arity constraints).

Thus, the performance of reasoning in IA networks is completely bound to the performances of path consistency algorithms, as the performance of reasoning in constraint networks is closely related to the performances of arc consistency algorithms (BFR95). But, if arc consistency has been widely studied in CSPs ((Mac77), (MH86), (Bes94), (BFR95)), path consistency has involved little interest from the temporal reasoning community.

Some authors, underlining that path consistency is time expensive, chose to reduce the expressive power of Allen's algebra (usually restricting to pointizable relations) and then produced efficient but quite complex algorithms to deal with the restricted language they defined ((GM89), (GS93)).

In this paper, we take another way: rehabilitating path consistency by producing a simple and efficient algorithm which deals with the overall Allen's algebra. First, we recall the different path consistency algorithms already presented in the temporal reasoning literature. They are discussed, and some experimental comparisons are given, which lead us to consider the direct adaptation of PC-2 (Mac77) (given in (DMP91)) as the best existing algorithm to perform path consistency in IA networks. Afterwards, we present the kernel of all these algorithms (in which they spend almost all their running time), namely the function REVISE. We discuss its behavior and present the main result of this paper: a simple way to improve its efficiency. Finally, we give some experiments to show the efficiency of path consistency with our improvement compared to the classical version of REVISE.

The paper is organized as follows. Section 2 gives some background on IA networks. Section 3 recalls the different existing path consistency algorithms and discusses them. In section 4, the improvement of the function REVISE is given. Section 5 experimentally shows its efficiency. Section 6 gives a summary of the paper.

Preliminaries

Interval algebra networks

Allen's interval algebra has been described in (All83). The elements of the algebra are *relations* that may exist between intervals of time. Given two fixed intervals (i.e. two elements of $\{(s,e) \in \Re^2 / s < e\}$), only one of thirteen *atomic* relations can hold. The set $\mathcal{A}$ of these thirteen atomic relations is $\{b, m, o, fi, di, si, e, s, d, f, oi, mi, bi\}$, with b for *before*, m for *meets*, o for *overlaps*, f for *finishes*, s for *starts*, e for *equals*, d for *during*, and $\forall x \in \{b, m, o, s, d, f\} : xi$ for "the converse of x". For any $\alpha \in \mathcal{A}, \alpha^{-1}$ will denote the converse of α.

An *IA network* R (vB92) consists of variables $\{i, j, ...\}$ (whose values are intervals), and arcs (i,j) labeled with the set R_{ij} of admissible atomic relations between variables i and j. The relation between two variables is allowed to be a set (or disjunction) of atomic relations in order to represent indefinite information. Allen allows the relations between two variables to be any subset of $\mathcal{A}$ (2^{13} relations). For any IA network R we suppose that $R_{ji} = R_{ij}^{-1} = \{\alpha^{-1} / \alpha \in R_{ij}\}$.

A *scenario* S of R is an IA network with the same variables as R, where each arc (i,j) is labeled with a single atomic relation belonging to R_{ij}. A *consistent instantiation* of R is an instantiation of each variable of R to a fixed interval, such that for each arc (i,j), the atomic relation holding between the values of i and j belongs to R_{ij} (we say that the instantiation of i and j satisfies R_{ij}). A *consistent scenario* of R is a scenario of R admitting a consistent instantiation.

Allen showed that much of the information contained in an IA network can be implicit, due to the topological constraints of interval relationships. He gave a transitivity table TrT, which for each pair of atomic relations α and β furnishes the *composition* $\mathrm{TrT}[\alpha, \beta]$ of these relations. We can define the composition $\otimes$ of any relations ρ and ψ in $2^{\mathcal{A}}$ as: $\rho \otimes \psi = \bigcup_{\alpha \in \rho, \beta \in \psi} \mathrm{TrT}[\alpha, \beta]$.

The transitive closure of a network R is the network R^c obtained when all the transitive information is given explicitly, i.e. when for all length-2 paths i, k, j in R^c we have: $R^c_{ij} \subseteq R^c_{ik} \otimes R^c_{kj}$. Referring to the constraint reasoning literature, transitive closure is now usually called path consistency.

The strong result given in the theorem below permits us to forget the instantiations of the variables to intervals and to deal only with the relations between variables.

Property 1 ((VP87)) *Any path consistent IA network that is a scenario is also a consistent scenario.*

Instance generator

In the following, we will use randomly generated IA networks to compare the performances of different algorithms. A randomly generated IA network will be characterized by three parameters:

- **n**, the number of variables involved.
- the density **D**$= C/(n * (n-1)/2)$, where C is the number of edges $\{i,j\}$[1] chosen in the network not to be labeled by the universal relation (the set $\mathcal{A}$ containing the 13 atomic relations). The edges are chosen with a uniform distribution, but not connected networks[2] are discarded.
- the tightness **T**, i.e. the probability that an atomic relation is forbidden in a non universal relation.

Then, $D*n*(n-1)/2$ edges are labeled by a relation containing an average of $(1-T)*13$ atomic relations[3], the other edges being labeled by the universal relation.

[1] In fact, for each edge $\{i,j\}$ chosen, we generate a relation R_{ij} labeling the arc (i,j) and we label the arc (j,i) with $R_{ji} = R_{ij}^{-1}$.

[2] A network is said to be "not connected" if there exists a pair of variables such that all the paths linking them contain at least one edge labeled by the universal relation.

[3] When an empty relation is generated, it is discarded and another one is generated.

We do not use the simpler generator used for example in (LR92a), in which $D = 1.0$ and $T = 0.5$ for all the instances generated. Indeed, with this generator, as soon as $n > 14$, we only produce inconsistent networks, and the more n increases, the easier it is to find an inconsistency. Then, we cannot compare accurately the behavior of different algorithms. With the generator we propose, for any value of n, we can produce over-constrained networks (inconsistent), under-constrained networks (trivially consistent), and networks in the "transition range" (in which consistency –or inconsistency– is more difficult to demonstrate).

The usual measures of the complexity of an algorithm dealing with IA networks are the number of look-ups to the transitivity table TrT, and the running time. In fact, with a classical implementation, we will see that they are often strongly correlated.

Path consistency algorithms

Several path consistency algorithms (PC) have been proposed in the temporal reasoning literature to compute the transitive closure of an IA network. All of them incorporate the function REVISE (even if it is not explicitly written as for Allen's or Vilain and Kautz's versions (All83), (VK86)). REVISE(i,k,j) updates R_{ij} by considering the length-2 path from i to j through k, $R_{ij} \leftarrow R_{ij} \cap R_{ik} \otimes R_{kj}$, and returns true iff R_{ij} has been modified[4].

```
function REVISE(i,k,j): Boolean;
1  if (R_ik = A) or (R_kj = A) then return False; [5]
2  T ← ∅;
3  for α ∈ R_ik do
4        for β ∈ R_kj do T ← T ∪ TrT[α,β];
5  if R_ij ⊆ T then return False;
6  R_ij ← R_ij ∩ T; R_ji ← R_ij^-1;
7  return True;
```

In this section, we will quickly underline that these different versions of PC are not equivalent in efficiency simply because they do not propagate in the same way the modifications caused by REVISE in the network.

The comparison we will make is not exhaustive (it would have been impossible), but concerns the most representative versions of PC we found in the literature. We will use for our comparisons:

- a procedure *PC1$_{IA}$* (see Fig.1), presented in (DMP91), slightly improved from the version used for example by Ladkin and Reinefeld (LR92b), which was adapted from the PC-1 version of path consistency in constraint networks (Mac77), (Mon74).
- a procedure *PC-VK* (see Fig.1), due to Vilain and Kautz (VK86), which is a slight improvement of Allen's version (All83).
- a procedure *PC-vB* (see Fig.2), given in (vB92, page 314).
- a procedure *PC2$_{IA}$* (see Fig.2), adapted from the PC-2 version of path consistency in constraint networks (Mac77). It is presented in (DMP91).

Some of these procedures have been first presented in the context of point algebra networks. But the principle is exactly the same as for interval algebra networks.

```
procedure PC1_IA;
1  repeat
2     CHANGE← False;
3     for k,i,j ← 1 to n do if i ≠ j ≠ k ≠ i then
4        if REVISE(i,k,j) then
5              if R_ij = ∅ then exit "inconsistency";
6              CHANGE← True;
7  until not CHANGE;
```

```
procedure PC-VK;
1  Q ← {(i,j)/i < j};
2  while Q ≠ ∅ do
3        select and delete an arc (i,j) from Q;
4        for k ≠ i, k ≠ j do
5           if REVISE(i,j,k) then
6              if R_ik = ∅ then exit "inconsistency";
7              else Append(Q, {(i,k)});
8           if REVISE(k,i,j) then
9              if R_kj = ∅ then exit "inconsistency";
10             else Append(Q, {(k,j)});
```

Figure 1: procedures PC1$_{IA}$ and PC-VK.

The results given Table 1 lead us to a few comments on the four algorithms tested. First, it is clear that the number of table look-ups is strongly correlated to the number of times the function REVISE is called (since all the algorithms use the same function REVISE). PC1$_{IA}$, which has no technique of propagation of the modifications caused by REVISE (it checks the overall network at each loop, until no more changes occur) is obviously the worst algorithm for the number of table look-ups. PC-VK, which maintains a list of modified arcs to propagate can circumscribe the propagation of modifications, and then, has a better behavior. PC-vB can be seen as a refined version of PC-VK where the list of modified arcs (i,j) to propagate has been replaced by the set of length-2 paths (i,j,k) and

[4] It is a direct adaptation of the version given for constraint networks in (Mac77, page 112).

[5] This line is a trivial improvement of the classical REVISE to avoid processing useless length-2 paths.

$n/D/T$		$PC1_{IA}$	PC-VK	PC-vB	$PC2_{IA}$
50/0.5/0.1	#TableLookups	2,108,126	2,083,158	1,735,833	1,041,582
	time (in sec.)	1.03	1.02	1.15	0.66
50/0.5/0.38	#TableLookups	14,260,290	4,935,570	3,797,018	2,723,227
	time (in sec.)	8.80	3.03	2.79	1.95
50/0.5/0.6	#TableLookups	35,263	29,112	27,802	3,504
	time (in sec.)	0.03	0.02	0.18	0.08

Table 1: The four algorithms on three representative types of random problems: under–, over–, and middle–constrained (500 instances per type). 208 among 500 instances of (50/0.5/0.38) were found to have a path consistent closure. These networks are then in the transition range, where much propagation is needed to find the result. 500 among 500 instances of (50/0.5/0.1), and 0 among 500 instances of (50/0.5/0.6) were found to have a path consistent closure.

procedures PC-vB and $PC2_{IA}$;
1 $Q \leftarrow \bigcup_{i<j}$ Related-Paths(i,j); [6]
2 **while** $Q \neq \emptyset$ **do**
3 select and delete a path (i,k,j) from Q;
4 **if** REVISE(i,k,j) **then**
5 **if** $R_{ij} = \emptyset$ **then** exit "inconsistency";
6 **else** $Q \leftarrow Q \cup$ Related-Paths(i,j);

function Related-Paths(i,j); /* for PC-vB */
1 return $\{(i,j,k),(k,i,j)/k \neq i, k \neq j\}$;

function Related-Paths(i,j); /* for $PC2_{IA}$ */
1 return $\{(i,j,k)/i < k, k \neq j\} \cup \{(k,i,j)/k < j, k \neq i\}$
$\cup\{(k,j,i)/k < i, k \neq j\} \cup \{(j,i,k)/j < k, k \neq i\}$;

Figure 2: procedures PC-vB and $PC2_{IA}$ (they only differ by their function Related-Paths).

(k,i,j) to check with REVISE. The function Related-Paths(i,j) returns the set of length-2 paths that need to be considered if R_{ij} is changed. The use of a set instead of a list, and of length-2 paths instead of arcs permits to avoid useless calls to REVISE (an (i,k,j) cannot belong twice to the set). Finally, $PC2_{IA}$, also based on a set of length-2 paths to check, handles them via its function Related-Paths(i,j) in order to maximize "collisions" of length-2 paths in the set, then minimizing the number of length-2 paths to check.

From these comments, the four algorithms can be ordered from $PC1_{IA}$ to $PC2_{IA}$ via PC-VK and PC-vB, with respect to the number of table look-ups they perform.

Usually, running time and the number of table look-ups are strongly correlated. However, when path consistency is performed on very easy networks (under-constrained or over-constrained), where no propagation is necessary (the network is already path consistent or trivially inconsistent), the time needed by PC-vB and $PC2_{IA}$ to initialize the $O(n^3)$ set of length-2 paths can degrade their overall performance to such an extent that the time they need to achieve path consistency can exceed the time needed by $PC1_{IA}$ or PC-VK (see Table 1). Fortunately, when path consistency is used at each step of a search procedure, this expensive initialization is performed only once at the root of the search tree, and then its cost is widely outweighed by the savings it implies during the search. Nevertheless, on very large networks (more than 300 variables), the $O(n^3)$ space of the set of length-2 paths used in PC-vB and $PC2_{IA}$ can become a real drawback on personal machines, and then PC-VK can advantageously be used.

[6] For $PC2_{IA}$ it is equivalent to "$Q \leftarrow \{(i,k,j)/i < j, k \neq i, k \neq j\}$".

Improving the function REVISE

As we have seen in the previous section, REVISE is the kernel of all path consistency algorithms. They differ in the way they propagate the modifications, but they all use the same function REVISE. Hence, an improvement of the efficiency of this function would yield an equivalent improvement of the efficiency of any PC algorithm.

We can first remark that REVISE(i,k,j) computes the complete composition of R_{ik} and R_{kj} before testing whether R_{ij} is included in or not. It would be sufficient to check for each atomic relation γ in R_{ij} if there exist α in R_{ik} and β in R_{kj} such that γ is in $\mathtt{TrT}[\alpha,\beta]$. In fact, it is sufficient to find a "support" for each γ in R_{ij}, i.e. a proof of its viability in the length-2 path (i,k,j). This idea comes from the function REVISE(i,j) used in AC-3 to compute arc consistency in constraint networks (Mac77). However, if for each γ in R_{ij} we simply seek α in R_{ik} and β in R_{kj} such that γ is in $\mathtt{TrT}[\alpha,\beta]$ we will have a worst-case number of table look-ups for REVISE(i,k,j) bounded by $|R_{ij}| \times |R_{ik}| \times |R_{kj}|$, i.e. $O(|\mathcal{A}|^3)$, while it was only

bounded by $|R_{ik}| \times |R_{kj}|$, i.e. $O(|\mathcal{A}|^2)$, in the classical version. We can avoid this trap by using the following property of IA networks (the proof is given in (Bes96)).

Property 2 *R being an IA network, and i, j, k three variables in this network:*

$$\gamma \in R_{ik} \otimes R_{kj} \Leftrightarrow \exists \alpha \in R_{ik} / \mathrm{TrT}[\alpha^{-1}, \gamma] \cap R_{kj} \neq \emptyset$$

Then, for each γ in R_{ij} we can seek only α in R_{ik} such that $\mathrm{TrT}[\alpha^{-1}, \gamma]$ intersects R_{kj}. The number of table look-ups is then bounded by $|R_{ij}| \times |R_{ik}|$, i.e. $O(|\mathcal{A}|^2)$[7]. This is done in the function REVISE2.

```
function REVISE2(i, k, j): Boolean;
1  if (R_ik = A) or (R_kj = A) then return False;
2  CHANGE ← False;
3  for γ ∈ R_ij do
       begin
4          for α ∈ R_ik do
5              if TrT[α^-1, γ] ∩ R_kj ≠ ∅ then goto 2;
6          R_ij ← R_ij \ {γ};
7          CHANGE ← True;
       end
8  if CHANGE then R_ji ← R_ij^-1;
9  return CHANGE;
```

A trivial improvement of REVISE2 is performed in REVISE3. Indeed, the complexity of REVISE2 being bounded by $|R_{ij}| \times |R_{ik}|$, it is better to test first which one of $|R_{ik}|$ and $|R_{kj}|$ is the largest relation, and then to call REVISE2 with the appropriate path: (i, k, j) if $|R_{ik}| < |R_{kj}|$, (j, k, i) otherwise. The complexity of REVISE3 is then bounded by $|R_{ij}| \times min(|R_{ik}|, |R_{kj}|)$.

```
function REVISE3(i, k, j): Boolean;
1  if |R_ik| < |R_kj|
2  then return REVISE2(i, k, j)
3  else return REVISE2(j, k, i);
```

Experimental results

For our experiments, we chose to compare the classical version of $PC2_{IA}$ with a version of $PC2_{IA}$ using the function REVISE3 instead of REVISE (named $PC2_{IA}$-3). This choice is guided by the good performances of $PC2_{IA}$ when compared to the other path consistency algorithms. Nevertheless, REVISE3 could be implemented in any of the four algorithms presented in Section 3. The gain would reach the same ratio.

We tested $PC2_{IA}$ and $PC2_{IA}$-3 on different kinds of randomly generated interval networks. We took different sizes of networks (from 30 to 150 variables), different densities (from 0.1 to 1.0), and for each type, we varied the tightness from under-constrained to over-constrained networks. Fig. 3 reports the results for only one of these experiments; but it is representative of the general behavior, which is rather stable when changing the parameter values. (Complete experimentation is given in (Bes96)).

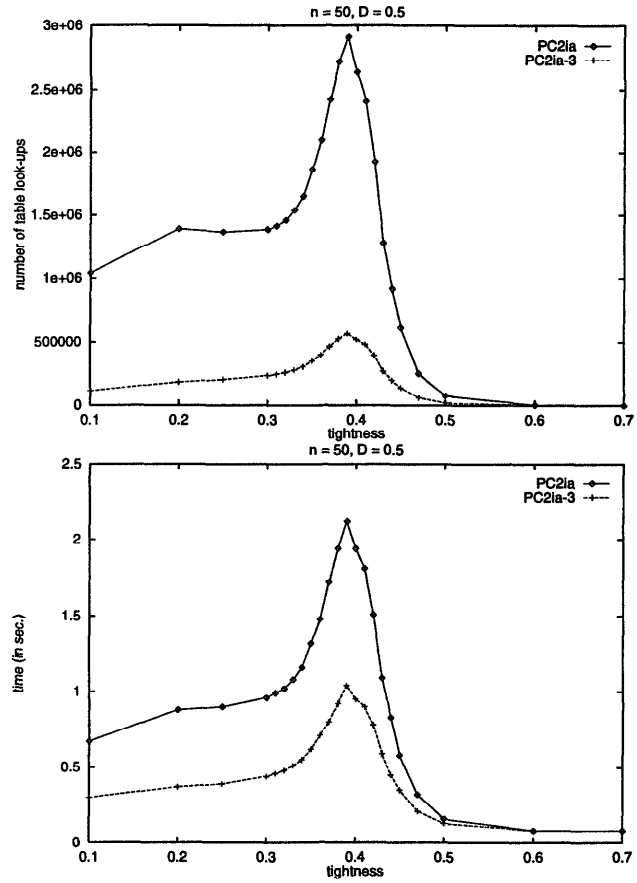

Figure 3: $PC2_{IA}$ and $PC2_{IA}$-3 on randomly generated networks with 50 variables and a density of 0.5 (500 instances for each value of the tightness).

These results show how much the new function REVISE3 overcomes the classical REVISE. Except on over-constrained networks, $PC2_{IA}$-3 is always 4 to 7 times better than $PC2_{IA}$ in number of table look-ups, and 2 to 4 times faster.

Summary and discussion

In this paper, we first showed that $PC2_{IA}$ is the path consistency algorithm which has the best behavior when some propagation is necessary to compute the transitive closure of an IA network (i.e. the cases where this computation is the most expensive). Afterwards, we discussed the way composition-intersection of rela-

[7] In the complexity analysis, we consider only the number of table look-ups. The $\cap$ and $\cup$ operations on relations are both considered as $O(1)$ operations since we can compute them in one cycle with a bit-vector implementation of relations (VK86).

tions is performed in PC algorithms, and we proposed an improved version of the function REVISE, namely REVISE3. We showed its efficiency on randomly generated IA networks.

The composition-intersection of relations being a general operation, not only useful in IA networks, the technique presented here will be usable in any type of networks based on an algebra where composition-intersection of relations is necessary. This will be the case in n-interval algebras (Lig90), (Lad86), or in networks representing spatial information (EH90), (RCC92).

We claim that the greater the number of relations in the algebra, the more REVISE3 will outperform REVISE. This is due to the feature of REVISE3, which looks for an unique "support" for each atomic relation while REVISE checks all the possible supports. This can be roughly compared to the respective features of AC-3 and AC-4 in constraint networks, where the more domain sizes increase, the more AC-3 outperforms AC-4. On algebras with many relations it would be worthwhile to study whether an AC-6 like version ((Bes94), (Sin95)) of PC would be interesting or not, and at what number of atomic relations it would start becoming efficient.

Acknowledgments I would like to thank Y. Hamadi, J.C. Régin, and J.F. Vilarem, who helped me to carry out the experiments, and Ian King who gave me the C++ code of his random number generator.

References

Allen, J. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11):832–843.

Bessière, C.; Freuder, E.; and Régin, J. 1995. Using inference to reduce arc consistency computation. In *Proceedings IJCAI'95*, 592–598.

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

Bessière, C. 1996. A simple way to improve path consistency processing in interval algebra networks. Technical Report 96-001, LIRMM.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Egenhofer, M., and Herring, J. 1990. A mathematical framework for the definition of topological relationships. In *Fourth International Symposium on Spatial Data Handling*, 803–813.

Gerevini, A., and Schubert, L. 1993. Efficient temporal reasoning through timegraphs. In *Proceedings IJCAI'93*, 648–654.

Ghallab, M., and Mounir Alaoui, A. 1989. Managing efficiently temporal relations through indexed spanning trees. In *Proceedings IJCAI'89*, 1297–1303.

Haralick, R., and Elliot, G. 1980. Increasing tree seach efficiency for constraint satisfaction problems. *Artificial Intelligence* 14:263–313.

Ladkin, P., and Reinefeld, A. 1992a. Effective solution of qualitative interval constraint problems. *Artificial Intelligence* 57:105–124.

Ladkin, P., and Reinefeld, A. 1992b. A symbolic approach to interval constraint problems. In *LNCS 737*. Springer-Verlag. 65–84.

Ladkin, P. 1986. Time representation: A taxonomy of interval relations. In *Proceedings AAAI'86*, 360–366.

Ligozat, G. 1990. Weak representations of interval algebra. In *Proceedings AAAI'90*, 715–720.

Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence* 8(1):99–118.

Mohr, R., and Henderson, T. 1986. Arc and path consistency revisited. *Artificial Intelligence* 28:225–233.

Montanari, U. 1974. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science* 7:95–132.

Nadel, B. 1989. Constraint satisfaction algorithms. *Computational Intelligence* 5:188–224.

Prosser, P. 1993. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence* 9(3):268–299.

Randell, D.; Cui, Z.; and Cohn, A. 1992. An interval logic for space based on 'connection'. In *Proceedings ECAI'92*, 394–398.

Singh, M. 1995. Path consistency revisited. In *Proceedings TAI'95*.

Smith, B., and Grant, S. 1995. Sparse constraint graphs and exceptionally hard problems. In *Proceedings IJCAI'95*, 646–651.

Valdés-Pérez, R. 1987. The satisfiability of temporal constraint networks. In *Proceedings AAAI'87*, 256–260.

van Beek, P. 1992. Reasoning about qualitative temporal information. *Artificial Intelligence* 58:297–326.

Vilain, M., and Kautz, H. 1986. Constraint propagation algorithms for temporal reasoning. In *Proceedings AAAI'86*, 377–382.