Networks of qualitative interval relations: combining circuit consistency and path consistency in the search for a solution

A. Isli and H. Bennaceur

Laboratoire d'Informatique de Paris-Nord
CNRS URA 1507, Inst. Galilée, Université Paris-Nord
Villetaneuse, F-93430, France

Abstract

We first define the concept of circuit consistency for interval networks. We then provide an example showing that even for atomic networks circuit consistency is not complete. Then we describe how to combine circuit consistency and path consistency in existing search algorithms. Instead of applying path-consistency at each node of the search tree, as do existing algorithms, the idea of our method is to apply path consistency only at nodes of level $(n-1)n/2$, and circuit consistency in the remaining nodes of the search tree (n is the number of variables of the input network). Circuit consistency filtering is potentially less effective than path consistency filtering; however circuit consistency is computationally one factor less expensive. We provide experiments showing the performances of the search in the two cases (a) when circuit consistency and path consistency are combined and (b) when only path consistency is used.

Area: Time and constraints.

Keywords: Interval algebra, Temporal constraint satisfaction, circuit consistency, path consistency, backtracking search.

1 Introduction

Reasoning in the interval algebra defined by Allen [Allen 83] is known to be NP-hard [Vilain et al. 86]. One solution is to restrict the expressiveness in such a way to obtain a subclass behaving "well" while still representing a large class of interesting problems; this solution makes a tradeoff between tractability and expressiveness. Three such subclasses are known, which are the convex subclass ($\mathcal{C}$), the pointizable subclass ($\mathcal{P}$) and the ORD-Horn subclass ($\mathcal{H}$); they are related by $\mathcal{C} \subset \mathcal{P} \subset \mathcal{H}$. All of them are subalgebras of the whole interval algebra (they are closed under composition, intersection and transposition), making networks expressed in either of them closed under path consistency, which solves the consistency problem in these cases.

Many existing systems based on Allen's algebra restricts the expressiveness to either of the above three subclasses; even to either of $\mathcal{C}$ and $\mathcal{P}$. There are however real-life problems for which neither of the subclasses gets use: diagnosis problems, scheduling problems and planning problems. For instance, it is known that in such problems the relations $\{<, >\}$ and $\{<, m, mi, >\}$ are crucial; and none of the three subclasses contains them. For instance, in scheduling expressing that two tasks I and J are disjunctive is extremely solicited (this is expressed by $I\{<, >\}J$ or $I\{<, m, mi, >\}J$).

Another solution is to design a general backtracking algorithm that deals with any problem, and eventually behaves well for all or part of real-life problems. In this paper, we propose to combine circuit consistency and path consistency in such algorithms. Instead of applying path consistency at every node of the search space, as does, for instance, Ladkin and Reinefeld's algorithm [Ladkin et al. 92, van Beek 92], our method applies path consistency only at nodes of level $(n-1) \times n/2$ (where n is the number of variables of the input network), and circuit consistency in the remaining nodes of the search space. Circuit consistency is less filtering than path consistency; however, it is one factor less expensive. We provide experimental results comparing Ladkin and Reinefeld's algorithm (1) as provided by its authors, and (2) as we propose in the paper.

The paper is organized as follows. Section 2 provides some background on qualitative interval networks. Section 3 defines the concept of circuit consistency, while Section 4 presents an algorithm making a network circuit consistent. The complexity analysis of circuit consistency is studied in Section 5. Section 6 gives some background on existing backtracking search algorithms, while Section 7 explains how to combine circuit consistency and path consistency during the search in those algorithms. Section 8 is devoted to experimental results. Finally, Section 9 summarizes the paper.

2 Qualitative interval networks

Some background on qualitative interval networks is needed.

In [Allen 83], Allen defined what is known in the literature as the interval algebra (IA). Temporal information is expressed in IA as relations on pairs of intervals; these relations will be referred to in the sequel as interval relations. An atomic interval relation stands for one of the thirteen alternative, exclusive configurations a pair of intervals can stand in:

before (<), meets (m), overlaps (o), starts (s), during (d), finishes (f); their respective converses after (>), met-by (mi), overlapped-by (oi), started-by (si), contains (di), finished-by (fi); equals (=), which is its proper converse. For instance, interval I is during interval J is expressed as (I d J). In turn, an interval relation is a disjunction of atomic interval relations; this is used to allow a pair of intervals to stand in one of many configurations. The set representation of an interval relation $(Ir_1J) \vee \ldots \vee (Ir_nJ)$ $(n \leq 13)$ is $I\{r_1, \ldots, r_n\}J$. For instance, interval I is before, overlaps or contains interval J is expressed by the disjunction $(I < J) \vee (IoJ) \vee (IdiJ)$, the set representation of which is $I\{<, o, di\}J$.

A qualitative interval network, P, consists of:

1. a finite number of variables, say $I_1, \ldots, I_n$, ranging over the set $\{(a, b) \in \mathbb{R}^2 : a < b\}$ of intervals, and
2. interval relations on pairs of the variables, the constraints of the network.

Given an n-variable interval network P, its graphical representation is defined as usual: the vertices are the variables of the network, and if a constraint $I_iR_{ij}I_j$ is specified on the pair (I_i, I_j) of variables then the arc (I_i, I_j) is created and labeled by R_{ij}. On the other hand, the matrix representation consists of an $n \times n$-matrix M_P defined as follows:

$\forall i, j = 1 \ldots n$, if a constraint $I_iR_{ij}I_j$ is specified on the pair (I_i, I_j) of variables then $M_P[i, j] = R_{ij}$, otherwise $M_P[i, j] = Allen^{13}$ (where $Allen^{13}$ is the universal interval relation: the set of all thirteen atomic interval relations).

A solution, or consistent instantiation, to P consists of an assignment $(I_1 = (a_1, b_1), \ldots, I_n = (a_n, b_n))$ of values to the variables satisfying all the constraints in the network; i.e. such that for all $i, j = 1 \ldots n$, if a constraint $I_iR_{ij}I_j$ is specified on the pair (I_i, I_j) of variables then the interval relation $(a_i, b_i)R_{ij}(a_j, b_j)$ holds. P is consistent if it contains a solution; it is minimal if any solution of any two-variable subnetwork extends to a global solution [Montanari 74, Mackworth 77]; it is globally consistent if any solution of any subnetwork extends to a global solution [Freuder 82]. Global consistency is also called strong n-consistency.

3 The concept of circuit consistency

Bennaceur, in [Bennaceur 94], defined the concept of circuit consistency for discrete networks. We define the concept for interval networks.

Definition 1 *Let P be an n-variable interval network. P is circuit consistent if there exists an ordering, say $I_1, \ldots, I_n$, of the variables such that for all $i \neq j \in \{1, \ldots, n\}$:*

$$R_{i((j \bmod n)+1)} \subseteq R_{ij} \otimes R_{j((j \bmod n)+1)}$$

where $(j \bmod n)$ stands for $(j$ modulo $n)$, and $\otimes$ for composition of interval relations.

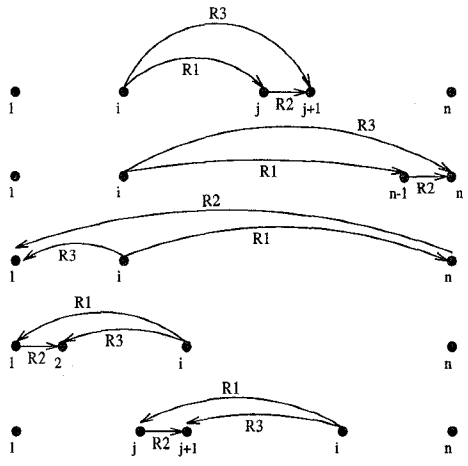

Figure 1: Illustration of the concept of circuit consistency: in each case, the condition $(R_3 \subseteq R_1 \otimes R_2)$ holds.

Example 1 *Consider the 6-variable interval network defined as follows:*

$$\begin{cases} R_{23} = R_{56} = oi, \\ R_{15} = R_{35} = m, \\ R_{ij} = o, \text{ for all other pair } (i, j), i < j. \end{cases}$$

The variables are $X_1, X_2, \ldots, X_6$; R_{ij} stands for the label on edge (X_i, X_j); the label on edge $(X_i, X_j), i > j$, is not given explicitely: it is equal to the converse (or transpose) of R_{ji}.

The network is circuit consistent. Indeed, it respects the definition of circuit consistency with respect to the ordering $X_1, X_2, \ldots, X_6$ of its variables. However, the triangle (X_1, X_3, X_5), and hence the whole network, is not path inconsistent.

As a side remark, verifying whether an atomic interval network[1] is circuit consistent with respect to the ordering $X_1, \ldots, X_n$ of variables is equivalent to verifying whether all triangles, $(X_i, X_{succ(i)}, X_j)$, containing two consecutive edges are path consistent.

Example 1 clearly shows that circuit consistency is not complete for the consistency problem of interval networks: an interval network may be inconsistent even if it is circuit consistent.

4 Achieving circuit consistency

Figure 2 presents an algorithm achieving circuit consistency for an n-variable interval network. The algorithm makes use of the following notations, for $i \in \{1, \ldots, n\}$:

$$succ(i) = \begin{cases} i+1 \text{ if } i < n, \\ 1 \text{ if } i = n. \end{cases}$$

$$pred(i) = \begin{cases} i-1 \text{ if } i > 1, \\ n \text{ if } i = 1. \end{cases}$$

[1] As it will be defined later, an interval network is atomic if for all $i, j = 1 \ldots n$, the label R_{ij} on edge (X_i, X_j) is an atomic interval relation.

1. **Input :** An n-variable interval network P.

2. **Output :** The network P made circuit consistent with respect to the ordering $I_1, \ldots, I_n$ of its variables.

   Method : The algorithm makes use of a data structure *Queue*. Initially, *Queue* contains those pairs $(i, succ(i))$ such that the constraint $R_{i(succ(i))}$ is not the universal relation. Then the algorithm proceeds by removing pairs $(i, succ(i))$ from *Queue* until *Queue* becomes empty. For each pair $(i, succ(i))$ that it removes, the following points are performed:

   $j := succ(i)$;
   While $(j \neq pred[i])\{$
   $Temp := R_{i(succ(j))}$;
   $R_{i(succ(j))} := R_{i(succ(j))} \cap R_{ij} \otimes R_{j(succ(j))}$;
   If $R_{i(succ(j))} \subset Temp\{$
   $R_{(succ(j))i} := transpose(R_{i(succ(j))})$;
   add the pair $(succ(j), succ(succ(j)))$ to *Queue*;
   $\}$
   $j := succ(j)$;
   $\}$

3. **The whole algorithm :**

   While (*Queue* is not empty)$\{$
   remove a pair $(i, succ(i))$ from *Queue*;
   $j := succ(i)$;
   While $(j \neq pred[i])\{$
   $Temp := R_{i(succ(j))}$;
   $R_{i(succ(j))} := R_{i(succ(j))} \cap R_{ij} \otimes R_{j(succ(j))}$;
   If $R_{i(succ(j))} \subset Temp\{$
   $R_{(succ(j))i} := transpose(R_{i(succ(j))})$;
   add the pair $(succ(j), succ(succ(j)))$ to *Queue*;
   $\}$
   $j := succ(j)$;
   $\}$
   $\}$

Figure 2: The circuit consistency algorithm.

5 Complexity analysis

The data structure *Queue* used in the algorithm of Figure 2 contains at most n pairs $(i, succ(i))$ at a time. A pair can be put into *Queue* at most thirteen times. Moreover, for each removing of a pair $(i, succ(i))$ from *Queue* $O(n)$ operations (compositions, intersections and transpositions) are performed (see the inner loop of the algorithm in Figure 2). It follows that the computational complexity of the algorithm is $O(n^2)$.

6 Backtraking search: existing algorithms

Given an interval network P, we define a refinement of P to be any network on the same set of variables in which every constraint is included in or equal to the corresponding constraint in P; i.e. for all constraint $IR'J$ of the refinement, the corresponding constraint IRJ of P is such that $R' \subseteq R$. A refinement is atomic (resp. convex, pointizable, ORD-Horn) when all its constraints are atomic (resp. convex, pointizable, ORD-Horn); i.e. they are of the form IRJ where R is an atomic (resp. convex, pointizable, ORD-Horn) relation.

It is known that when a network contains only atomic labels on its edges the path-consistency method [Montanari 74, Mackworth 77] answers the consistency problem. Valdés-Pérez [Valdés-Pérez 87] used this result to design a backtracking search algorithm in the style of backjumping [Gaschnig 78] for the consistency problem of interval networks.

Another backtracking search algorithm for the consistency problem of general interval networks is the one of Ladkin and Reinefeld [Ladkin et al. 92, van Beek 92] which is based on a subclass $\mathcal{C}$ of the interval algebra verifying the following points:

1. $\mathcal{C}$ contains all thirteen atomic relations, and is closed under path-consistency (i.e. path-consistency applied to a network expressed in $\mathcal{C}$ leads to a network expressed in $\mathcal{C}$);

2. path-consistency detects inconsistency for networks expressed in $\mathcal{C}$.

$\mathcal{C}$ may be the set of atomic relations or any of the convex, the pontizable or the ORD-Horn subclasses:

1. each of these three subclasses is a subalgebra of the whole interval algebra: it is closed under transposition, intersection and composition;

2. the consistency problem of a network expressed in either of the subclasses is solved by the path-consistency algorithm;

3. the ORD-Horn subclass is the unique greatest subclass among the tractable subclasses that contain all thirteen atomic relations [Nebel et al. 94].

[2] If $R_{i(succ(j))} \subset Temp$ then $R_{i(succ(j))}$ has been successfully modified, implying that $R_{(succ(j))i}$ may constrain relations $R_{(succ(j))k}$ to modification; for this reason, the pair $(succ(j), succ(succ(j)))$ is added to *Queue*.

Let n be the number of variables and $I_1, \ldots, I_n$ the variables of an interval network. The algorithm of Ladkin and Reinefeld defines an instantiation ordering, say

$$(I_1, I_2), (I_1, I_3), \ldots, (I_1, I_n), \ldots,$$

$$(I_j, I_{j+1}), \ldots, (I_j, I_n), \ldots, (I_{n-1}, I_n),$$

of the edges and constructs in a depth-first manner the labeled tree described in Figure 3 until a consistent refinement is found or all the tree is constructed without finding such a refinement.

1. create the root of the tree, and label it by $PC(P)$,[2] where P is the original network;

   /*

   * the root is of level 1;

   * if a node n_1 is of level ℓ then all node n_2 which is

   * an immedite successor of n_1 is of level $\ell + 1$;

   */

2. while there are nodes yet not marked:

   (a) consider a node s yet not marked;

   (b) let ℓ be the level of s and P' its label;

   (c) mark s;

   (d) *if* $\ell \neq (n-1) \times n/2$ *and* P' is not the empty network:[3]

   i. let (I, J) be the ℓth edge in the instantiation ordering;

   ii. let $label(I, J)$ be the label on edge (I, J) in the network P';

   iii. decompose $label(I, J)$ into a union $label(I, J) = \bigcup_{i=1}^{m} c_i$ of elements of $\mathcal{C}$;

   iv. for all c_i, $i = 1 \ldots m$

   create an immediate successor for n, and

   label it by $PC(P'[label(I, J) \leftarrow c_i])$;[4]

Figure 3: Ladkin and Reinefeld's algorithm.

7 Combining circuit consistency and path consistency in search algorithms

Ladkin and Reinefeld's algorithm is modified in the following manner. We first apply circuit consistency to the whole network; i.e. we replace step 1 in Figure 3 by the following:

1. create the root of the tree, and label it by $CC(P)$,[5] where P is the original network;

[3] $PC(P)$ is the network obtained by applying path consistency to P.

[4] If $\ell = (n-1) \times n/2$ and the label P of node s is not the empty network then P is a consistent refinement of the original network.

[5] $P'[label(I, J) \leftarrow c_i]$ is the network obtained by substituting c_i to the label on edge (I, J) of P'.

[5] $CC(P)$ is the network obtained by applying circuit consistency to network P.

We then check whether the current node in the search tree is of level $(n-1) \times n/2$. If this is the case then the corresponding edge is the last in the instantiation ordering, and the algorithm calls the path-consistency procedure. Otherwise, there are more than one edge yet not instantiated, and the algorithm calls the circuit consistency procedure. Hence, all we need, in addition to the modification above, is to modify step 2d in Figure 3 in the following manner.

1. *if* P' is not the empty network:

   i. let (I, J) be the ℓth edge in the instantiation ordering;

   ii. let $label(I, J)$ be the label on edge (I, J) in the network P';

   iii. decompose $label(I, J)$ into a union $label(I, J) = \bigcup_{i=1}^{m} c_i$ of elements of $\mathcal{C}$;

   iv. **if** $(\ell < (n-1) \times n/2)$ **then**

   for all c_i, $i = 1 \ldots m\{$

   create an immediate successor for n, and

   label it by $CC(P[label(I, J) \leftarrow c_i])$;

   $\}$

   else for all c_i, $i = 1 \ldots m\{$

   create an immediate successor for n, and

   label it by $PC(P[label(I, J) \leftarrow c_i])$;

   $\}$

8 Experiments

In this section, we provide experimental tests on the performances of Ladkin and Reinefeld's search algorithm in the following two cases: (1) the algorithm in which only path consistency is used during the search, and (2) the algorithm in which circuit consistency and path consistency are combined.

We compared the two algorithms on randomly generated problems. We took into account the following three parameters:

1. the size of the problems, i.e. the number of variables;

2. the density, i.e. the fraction of the number of non universal constraints to $n \times (n-1)/2$, where n is the number of variables;

3. the tightness, i.e. the fraction of forbidden atomic relations in the non universal constraints to 13.

We randomly generated a first list of problems according to the following values of the parameters:

1. $8 \leq size \leq 15$;

2. $0.7 \leq density \leq 0.9$; and

3. $9/13 \leq tightness \leq 12/13$.

Then a second list according to the following values of the parameters:

1. $8 \leq size \leq 15$;
2. $density = 0.05$; and
3. $tightness = 1/13$.

The comparison is based on the number of calls to the composition of two relations. Composition is the main time consuming during the search. We also computed the number of nodes of the search space effectively spanned. The respective results are reported in the two tables below, where CC and PC stand for circuit consistency and path consistency, respectively.

nb of variables	PC alone		CC and PC combined	
	nb of nodes	nb of calls	nb of nodes	nb of calls
8	0	28	0	11
9	0	18	0	15
10	0	31	0	12
11	0	20	0	10
12	0	20	0	11
13	0	46	0	12
14	0	23	0	11
15	0	34	0	11

nb of variables	PC alone		CC and PC combined	
	nb of nodes	nb of calls	nb of nodes	nb of calls
8	29	377	28	393
9	37	555	36	578
10	46	817	45	815
11	56	1083	55	1103
12	67	1447	66	1459
13	79	1885	78	1873
14	92	2386	91	2373
15	105	3025	105	2947

The first case corresponds to dense and tight problems, which are difficult to satisfy (that is to say, the probability for such a problem to be inconsistent is almost 1). The results clearly show that for these problems, combining circuit consistency and path consistency gets more use than path consistency alone.

For this first case the number of nodes of the search space effectively spanned is equal to 0. This is due to the fact that the problems are trivially inconsistent, and that inconsistency is detected by the first path consistency or the first circuit consistency, depending on whether the search uses PC alone or combines CC and PC (before calling the procedure backtrack).

The second case corresponds to problems of very weak tightness and very weak density; that is to say to underconstrained problems. These problems are trivially consistent. The first path consistency or the first circuit consistency, depending on whether the search uses PC alone or combines CC and PC, succeeds; then a solution is found with backtrack free (the table shows clearly that the number of visited nodes is $n \times (n-1)/2$, where n is the number of variables). As the second table shows, the two methods present similar performances for this second class of problems.

Discussion : Except for the two cases reported above, using PC alone, as suggested by Ladkin and Reinefeld [Ladkin et al. 92], gets more use than combining the two filtering methods. Indeed, for the other problems we generated in our experiments, which are not reported here, the algorithm with PC alone answers in a reasonable amount of time while the algorithm with CC and PC combined takes much more time, in some cases giving no answer. This explains that, at least for the generated problems, the filtering power of PC is more important than the one of CC.

We believe however that the study of combining CC and PC during the search, not necessarily in the way presented in the paper, should be pursued. An alternative may be the following:

1. consider two integer values α and β such that $1 \leq \alpha < \beta \leq n$, where n is the number of variables of the input network;
2. then during the search if the level ℓ of the current edge is such that $(\ell \ mod \ \beta) \leq \alpha$ apply CC otherwise apply PC;
3. finally, to make the algorithm complete, apply PC when instantiating the last edge.

9 Summary and future work

We extended the definition of circuit consistency to qualitative interval networks. We then showed how to modify Ladkin and Reinefeld's search algorithm in such a way to combine circuit consistency (CC) and path consistency (PC) during the search, instead of using only PC. CC is less filtering than PC; however, it is one factor less expensive. We presented experimental results comparing the performances of (1) Ladkin and Reinefeld's algorithm [Ladkin et al. 92] in which only PC is used during the search, and (2) the algorithm in which CC and PC are combined.

The experiments presented show that, at least for the problems randomly generated, combining CC and PC gets more use only in some cases. However, as we discussed it in last section, other ways of combining CC and PC exist, that may beat, eventually for classes of real-life problems, the algorithm with PC alone.

Finally the work presented in this paper extends in a natural way to search algorithms in quantitative temporal networks [Dechter et al. 91].

References

[Allen 83] **Allen, J.F.**, *Maintaining Knowledge About Temporal Intervals*, Communications of the ACM 26 (11) (1983) 832 – 843.

[Bennaceur 94] **Bennaceur, H.**, Partial Consistency for Constraint Satisfaction Problems, in: Proceedings ECAI-94, Amsterdam, (1994) 120 – 124.

[Dechter et al. 91] **Dechter, R., Meiri, I.** and **Pearl, J.**, *Temporal Constraint Networks*, Artificial Intelligence 49 (1991) 61 – 95.

[Freuder 82] **Freuder, E.C.**, *A Sufficient Condition for Backtrack-free Search*, Journal of the ACM 29 (1982) 24 – 32.

[Gaschnig 78] **Gaschnig, J.**, *Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing assignment problems*, in: Proceedings of the Second Biennial Conference of the

Canadian Society for Computational Studies of Intelligence, Toronto, Ont., (1978) 268 – 277.

[Isli et al. 95] **Isli, A.** and **Bennaceur, H.**, *Networks of qualitative interval relations: combining circuit consistency and path consistency in the search for a solution*, Pré-publication, L.I.P.N., Université Paris 13, France (to appear).

[Ladkin et al. 92] **Ladkin, P.** and **Reinefeld, A.**, *Effective solution of qualitative constraint problems*, Artificial Intelligence 57 (1992) 105 – 124.

[Mackworth 77] **Mackworth, A.K.**, *Consistency in Networks of Relations*, Artificial Intelligence 8 (1977) 99 – 118.

[Montanari 74] **Montanari, U.**, *Networks of Constraints: Fundamental Properties and Applications to Picture Processing*, Information Sciences 7 (1974) 95 – 132.

[Nebel et al. 94] **Nebel, B.** and **Bürckert, H.J.**, *Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra*, in: Proceedings AAAI-94, Seattle, WA, (1994) 356 – 361.

[Valdés-Pérez 87] **Valdés-Pérez, R.E.**, *The satisfiability of temporal constraint networks*, in: Proceedings AAAI-87, Seattle, Washington, (1987) 256 – 260.

[van Beek 92] **van Beek, P.**, *Reasoning about Qualitative Temporal Information*, Artificial Intelligence 58 (1992) 297 – 326.

[Vilain et al. 86] **Vilain, M.** and **Kautz, H.**, *Constraint Propagation Algorithms for Temporal Reasoning*, in: Proceedings AAAI-82, Philadelphia, PA, (1986) 377 – 382.