Coping With Disjunctions in Temporal Constraint Satisfaction Problems *

Eddie Schwalb, Rina Dechter
Department of Information and Computer Science
University of California at Irvine, CA 92717
eschwalb@ics.uci.edu, dechter@ics.uci.edu

Abstract

Path-consistency algorithms, which are polynomial for discrete problems, are exponential when applied to problems involving quantitative temporal information. The source of complexity stems from specifying relationships between pairs of time points as disjunction of intervals. We propose a polynomial algorithm, called ULT, that approximates path-consistency in *Temporal Constraint Satisfaction Problems* (TCSPs). We compare ULT empirically to *path-consistency* and *directional path-consistency* algorithms. When used as a preprocessing to backtracking, ULT is shown to be 10 times more effective then either DPC or PC-2.

1. Introduction

Problems involving temporal constraints arise in various areas of computer science such as scheduling, circuit and program verification, parallel computation and common sense reasoning. Several formalisms for expressing and reasoning with temporal knowledge have been proposed, most notably Allen's interval algebra (Allen 83), Vilain and Kautz's point algebra (Vilain 86, Vanbeek 92), and Dean & Mcdermott's Time Map Management (TMM) (Dean & McDermott 87).

Recently, a framework called *Temporal Constraint Problem (TCSP)* was proposed (Dechter Meiri & Pearl 91), in which network-based methods (Dechter & Pearl 88, Dechter 92) were extended to include continuous variables. In this framework, variables represent time points and *quantitative* temporal information is represented by a set of unary and binary constraints over the variables. This model was further extended in (Meiri 91) to include *qualitative* information. The advantage of this framework is that it facilitates the following tasks: (1) finding all feasible times a given event can occur, (2) finding all possible relationships between two given events, (3) finding one or more scenarios consistent with the information provided, and (4) representing the data in a *minimal network* form that can provide answers to a variety of additional queries.

*This work was partially supported by NSF grant IRI-9157636, by Air Force Office of Scientific Research, AFOSR 900136, and by Xerox grant.

It is well known that all these tasks are NP-hard. The source of complexity stems from specifying relationships between pairs of time points as disjunctions of intervals. Even enforcing path-consistency, which is polynomial in discrete problems, becomes worst-case exponential in the number of intervals in each constraint. On the other hand, simple temporal problems having only one interval per constraint are tractable and can be solved by path-consistency. Consequently, we propose to exploit the efficiency of processing simple temporal problems for approximating path consistency. This leads to a polynomial algorithm, called *ULT*.

We compare ULT empirically with path-consistency (PC-2) and directional path-consistency (DPC). Our results show that while ULT is always a very efficient algorithm, it is most accurate (relative to full path-consistency enforced by PC-2) for problems having a small number of intervals and high connectivity. When used as a preprocessing procedure before backtracking, ULT is 10 times more effective then DPC or PC.

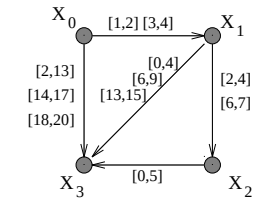
The paper is organized as follows. Section 2 presents the TCSP model. Section 3 presents algorithm ULT; Section 4 presents the empirical evaluation and the conclusion is presented in Section 5.

2. The TCSP Model

A TCSP involves a set of variables, $X_1, \ldots, X_n$, having continuous domains, each representing a time point. Each constraint T is represented by a set of intervals $T \equiv (I_1, \ldots, I_n) = \{[a_1, b_1], \ldots, [a_n, b_n]\}$. For a unary constraint T_i over X_i, the set of intervals restricts the domain such that $(a_1 \leq X_i \leq b_1) \cup \ldots \cup (a_n \leq X_i \leq b_n)$. For a binary constraint T_{ij} over X_i, X_j, the set of intervals restricts the permissible values for the distance $X_j - X_i$; namely it represents the disjunction $(a_1 \leq X_j - X_i \leq b_l) \cup \ldots \cup (a_n \leq X_j - X_i \leq b_n)$. All intervals are pairwise disjoint.

A *binary TCSP* can be represented by a *directed con-*

straint graph, where nodes represent variables and an edge $i \rightarrow j$ indicates that a constraint T_{ij} is specified. Every edge is labeled by the interval set (Figure 1). A special time point X_0 is introduced to represent the "beginning of the world". Because all times are relative to X_0, thus we may treat each unary constraint T_i as a binary constraint T_{0i} (having the same interval representation). For simplicity, we choose $X_0 = 0$.



$T_{01} = \{[1,2],\ [3,4]\}$
$T_{03} = \{[2,13],\ [14,17],\ [18,20]\}$
$T_{12} = \{[2,4],\ [6,7]\}$
$T_{13} = \{[0,4],\ [6,9],\ [13,15]\}$
$T_{23} = \{[0,5]\}$

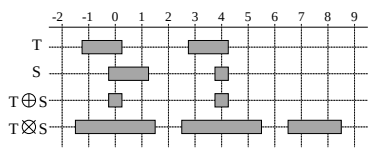
$T_{01} \equiv (1 \leq X_1 \leq 2) \cup (3 \leq X_1 \leq 4)$
$T_{03} \equiv (2 \leq X_3 \leq 13) \cup (14 \leq X_3 \leq 17) \cup (18 \leq X_3 \leq 20)$
$T_{12} \equiv (2 \leq X_2 - X_1 \leq 4) \cup (6 \leq X_2 - X_1 \leq 7)$
$T_{13} \equiv (0 \leq X_3 - X_1 \leq 4) \cup (6 \leq X_3 - X_1 \leq 9) \cup (13 \leq X_3 - X_1 \leq 15)$
$T_{23} \equiv (0 \leq X_3 - X_2 \leq 5)$

Figure 1: A graphical representation of a TCSP where $X_0 = 0,\ X_1 = 1.5,\ X_2 = 4.5,\ X_3 = 8$ is a solution.

A tuple $X = (x_1, \ldots, x_n)$ is called a *solution* if the assignment $X_1 = x_1, \ldots, X_n = x_n$ satisfies all the constraints. The network is *consistent* iff at least one solution exists. A value v is a *feasible value* of X_i if there exists a solution in which $X_i = v$. The *minimal domain* of a variable is the set of all *feasible values* of that variable. The *minimal constraint* is the tightest constraint that describes the same set of solutions. The *minimal network* is such that its domains and constraints are minimal.

Definition 1: Let $T = \{I_1, \ldots, I_l\}$ and $S = \{J_1, \ldots, j_m\}$ be two sets of intervals which can correspond to either unary or binary constraints.

1. The *intersection* of T and S, denoted by $T \oplus S$, admits only values that are allowed by both of them.
2. The *composition* of T and S, denoted by $T \otimes S$, admits only values r for which there exists $t \in T$ and $s \in S$ such that $r = t + s$.



$T = \{[-1.25, 0.25],\ [2.75, 4.25]\}$
$S = \{[-0.25, 1.25],\ [3.77, 4.25]\}$
$T \oplus S = \{[-0.25, 0.25],\ [3.75, 4.25]\}$
$T \otimes S = \{[-1.50, 1.50],\ [2.50, 5.50],\ [6.50, 8.50]\}$

Figure 2: A pictorial example of the $\oplus$ and $\otimes$ operations.

The $\otimes$ operation may result in intervals that are not pairwise disjoint. Therefore, additional processing may be required to compute the disjoint interval set.

Definition 2: The *path-induced* constraint on variables X_i, X_j is $R_{ij}^{path} = \oplus_{\forall k}(T_{ik} \otimes T_{kj})$. A constraint T_{ij} is *path-consistent* iff $T_{ij} \subseteq R_{ij}^{path}$ and *path-redundant* iff $T_{ij} \supseteq R_{ij}^{path}$. A network is *path-consistent* iff all its constraints are *path-consistent*.

A general TCSP can be converted into an equivalent *path-consistent* network by applying the relaxation operation $T_{ij} \leftarrow T_{ij} \oplus T_{ik} \otimes T_{kj}$, using algorithm PC-2 (Figure 3). Some problems may benefit from a weaker version, called DPC, which can be enforced more efficiently.

Algorithm PC-2
1. $Q \leftarrow \{(i,k,j) | (i < j) and (k \neq i, j)\}$
2. **while** $Q \neq \{\}$ **do**
3. select and delete a path (i,k,j) from Q
4. **if** $T_{ij} \neq T_{ik} \otimes T_{kj}$ **then**
5. $T_{ij} \leftarrow T_{ij} \oplus (T_{ik} \otimes T_{kj})$
6. **if** $T_{ij} = \{\}$ **then** exit (inconsistency)
7. $Q \leftarrow Q \cup$ *RELATED-PATHS*$((i,k,j))$
8. **end-if**
9. **end-while**

Algorithm DPC
1. **for** $k \leftarrow n$ downto 1 by -1 **do**
2. **for** $\forall i, j < k$ such that $(i,k), (k,j) \in E$ **do**
3. **if** $T_{ij} \neq T_{ik} \otimes T_{kj}$ **then**
4. $E \leftarrow E \cup (i,j)$
5. $T_{ij} \leftarrow T_{ij} \oplus (T_{ik} \otimes T_{kj})$
6. **if** $T_{ij} = \{\}$ **then** exit (inconsistency)
7. **end-if**
8. **end-for**
9. **end-for**

Figure 3: Algorithms PC-2 and DPC (Dechter Meiri & Pearl 91).

3. Upper-Lower Tightening (ULT)

The relaxation operation $T_{ij} \leftarrow T_{ij} \oplus T_{ik} \otimes T_{kj}$ increases the number of intervals and may result in exponential blow-up. As a result, the complexity of PC-2 and DPC is exponential in the number of intervals, but can be bounded by $O(n^3R^3)$ and $O(n^3R^2)$, respectively, where n is the number of variables and R is the range of the constraints. When running PC-2 on random instances, we encountered problems for which path-consistency required 11 minutes on toy-sized problems with 10 variables, range of [0,600], and with 50 input intervals in each constraint. Evidently, PC-2 is computationally expensive (also observed by (Poesio 91)).

A special class of TCSPs that allow efficient processing is the Simple Temporal Problem (STP) (Dechter Meiri & Pearl 91). In this class, a constraint has a single interval. An STP can be associated with a directed edge-weighted graph, G_d, called a *distance graph*, having the same vertices as the constraint graph G; each edge $i \rightarrow j$ is labeled by a weight w_{ij} representing the constraint $X_j - X_i \leq w_{ij}$ (Figure 4). An STP is consistent iff the corresponding d-graph G_d has no negative cycles and the minimal network of the STP

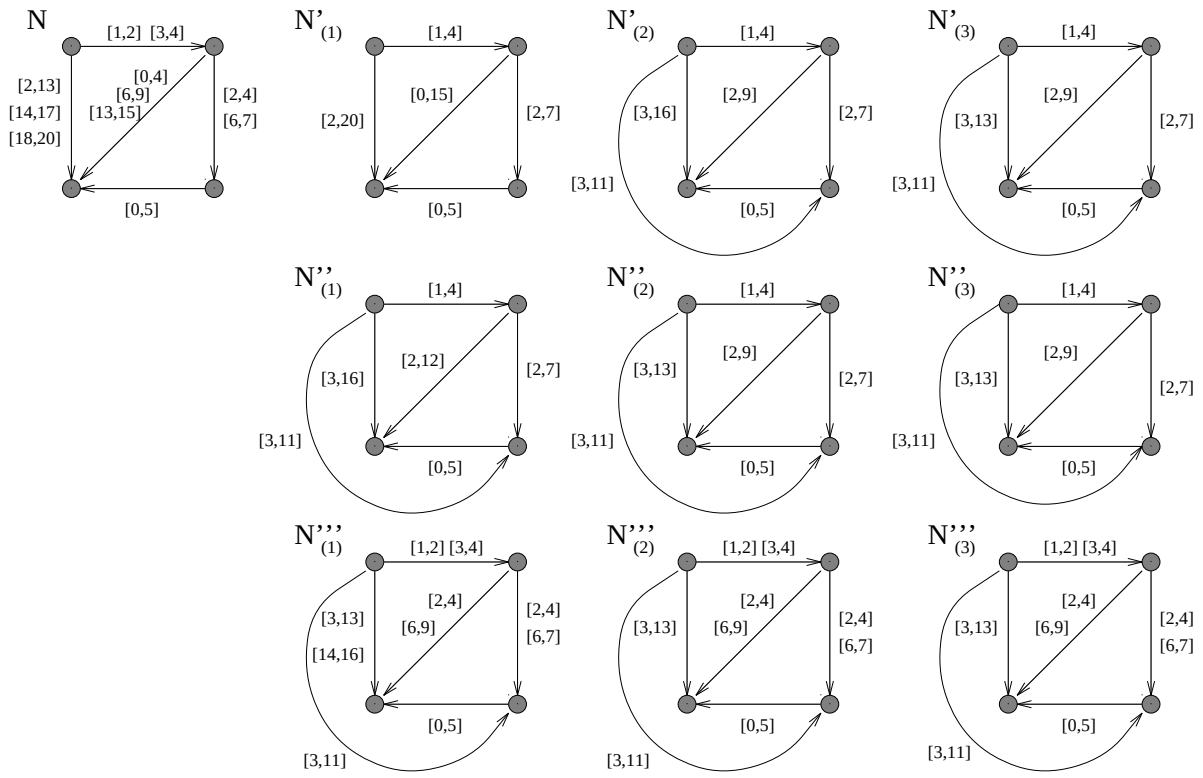


Figure 5: A sample run of ULT. We start with N (Figure 1) and compute $N'_{(1)}$, $N''_{(1)}$, $N'''_{(1)}$. Thereafter, we perform a second iteration in which we compute $N'_{(2)}$, $N''_{(2)}$, $N'''_{(2)}$ and finally, in the third iteration, there is no change. The first iteration removes two intervals, while the second iteration removes one.

corresponds to the *minimal distances* of G_d. For a processing example, see figure 4. Alternatively, the minimal network of an STP can be computed by PC-2 in $O(n^3)$ steps.

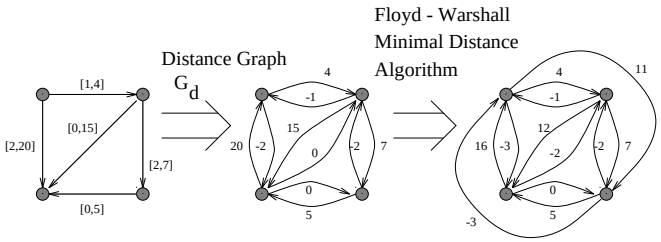


Figure 4: Processing an STP. The minimal network is in Figure 5, network $N''_{(1)}$

Motivated by these results, we propose an efficient algorithm that approximates path-consistency. The idea is to use the extreme points of all intervals associated with a single constraint as one big interval, yielding an STP, and then to perform path-consistency on that STP. This process will not increasing the number of intervals.

Definition 3: Let $T_{ij} = [I_1, \ldots, I_m]$ be the constraint over variables X_i, X_j and let L_{ij}, U_{ij} be the lowest and highest value of T_{ij}, respectively. We define N', N'', N''' as follows (see Figure 5):

Algorithm Upper-Lower Tightening (ULT)

1. **input:** N
2. $N''' \leftarrow N$
3. **repeat**
4. $N \leftarrow N'''$
5. compute N', N'', N'''.
6. **until** $\forall ij \;\; (L'''_{ij} = L_{ij})$ *and* $(U'''_{ij} = U_{ij})$ *or* $\exists ij \;\; (U'''_{ij} < L'''_{ij})$
7. **if** $\exists ij \;\; (U'''_{ij} < L'''_{ij})$ **output:** "Inconsistent." otherwise **output:** N'''

Figure 6: The ULT algorithm.

- N' is an STP derived from N by relaxing its constraints to $T'_{ij} = [L_{ij}, U_{ij}]$.
- N'' is the minimal network of N'.
- N''' is derived from N'' and N by intersecting $T'''_{ij} = T''_{ij} \oplus T_{ij}$.

Algorithm ULT is presented in Figure 6. We can show that ULT computes a network equivalent to its input network.

Lemma 1: *Let N be the input to ULT and R be its output. The networks N and R are equivalent.*

Regarding the effectiveness of ULT, we can show that

Lemma 2: *Every iteration of ULT (excluding the last) removes at least one interval.*

This can be used to show that

Theorem 1: *Algorithm ULT terminates in* $O(n^3ek + e^2k^2)$ *steps where* n *is the number of variables,* e *is the number of edges, and* k *is the maximal number of intervals in each constraint.*

In contrast to PC-2, ULT is guaranteed to converge in $O(ek)$ iterations even if the interval boundaries are not rational numbers. For a sample execution see Figure 5.

Algorithm ULT can also be used to identify *path-redundancies*.

Definition 4: A constraint T_{ij} is *redundant-prone* iff, after applying ULT, T'''_{ij} is redundant in N'''.

Lemma 3: T'''_{ij} *is* path-redundant *in* N''' *if* $T''_{ij} \subseteq T_{ij}$ *and* $T''_{ij} = \oplus_{\forall k}(T''_{ik} \otimes T''_{kj})$.

Corollary 1 : *A single interval constraint* T_{ij} *is* redundant-prone *iff* $T''_{ij} = \oplus_{\forall k}(T''_{ik} \otimes T''_{kj})$.

Consequently, after applying ULT to a TCSP, we can test the condition in Corollary 1 and eliminate some redundant constraints.

A brute-force algorithm for solving a TCSP decomposes it into separate STPs by selecting a single interval from each constraint (Dechter Meiri & Pearl 91). Each STP is then solved separately and the solutions are combined. Alternatively, a naive backtracking algorithm will successively assign an interval to a constraint, as long as the resulting STP is consistent.[1] Once inconsistency is detected, the algorithm backtracks. Algorithm ULT can be used as a preprocessing stage to reduce the number of intervals in each constraint and to identify some *path-redundant* constraints. Since every iteration of ULT removes at least one interval, the search space is pruned. More importantly, if ULT causes redundant constraints to be removed, the search space may be pruned exponentially in the number of constraints removed. Note however that the number of constraints in the initial network is e while following the application of ULT, DPC or PC-2, the constraint graph becomes complete thus the number of constraints is $O(n^2)$.

4. Empirical Evaluation

We conducted two sets of experiments. One comparing the efficiency and accuracy of ULT and DPC relative to PC-2, and the other comparing their effectiveness as a preprocessing to backtracking.

Our experiments were conducted on randomly generated sets of instances. Our random problem generator uses five parameters: (1) n, the number of variables, (2) k, the number of intervals in each constraint, (3) R = [Inf,Sup], the range of the constraints, (4) T_I, the tightness of the constraints, namely, the fraction of values allowed relative to the interval [Inf,Sup], and (5) P_C, the probability that a constraint T_{ij} exists. Intuitively, problems with dense graphs and loose constraints with many intervals should be more difficult.

[1] We call this process "labeling the TCSP".

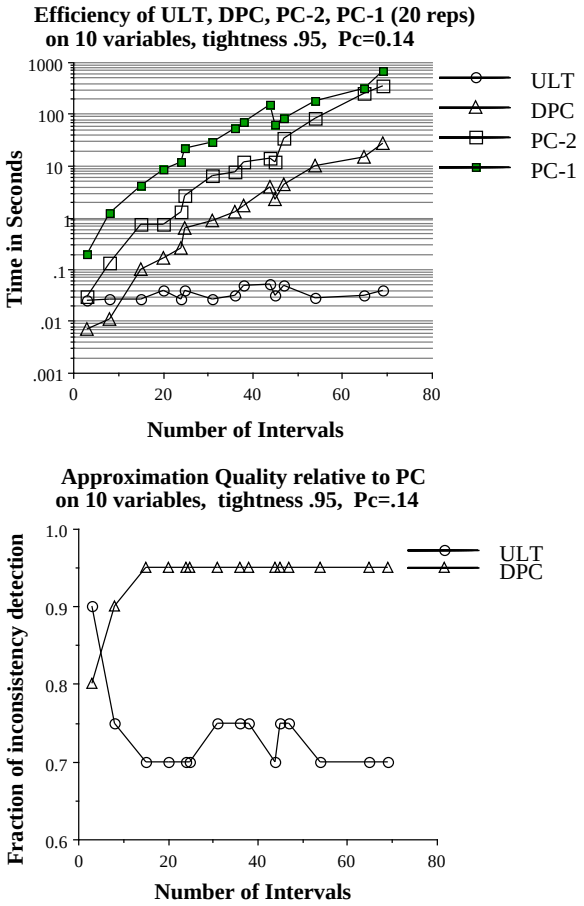


Figure 8: The execution time and quality of the approximation obtained by DPC and ULT to PC. Each point represents 20 runs on networks with 10 variables, .95 tightness, connectivity $P_C = .14$ and range $[0, 600]$.

To evaluate the quality of the approximation achieved by ULT and DPC relative to PC-2, we counted the number of cases in which ULT and DPC detected inconsistency given that PC-2 detected one. From Figure 8, we conclude that when the number of intervals is small, DPC is faster than ULT but produces weaker approximations. When the number of intervals is large, DPC is much slower but is more accurate. In addition, we observe that when the number of intervals is very large, DPC computes a very good approximation to PC-2, and runs about 10 times faster.

In Figure 9 we report the relative quality of ULT and DPC for a small (3) and for a large (20) number of intervals, as a function of connectivity. As the connectivity increases, the approximation quality of both

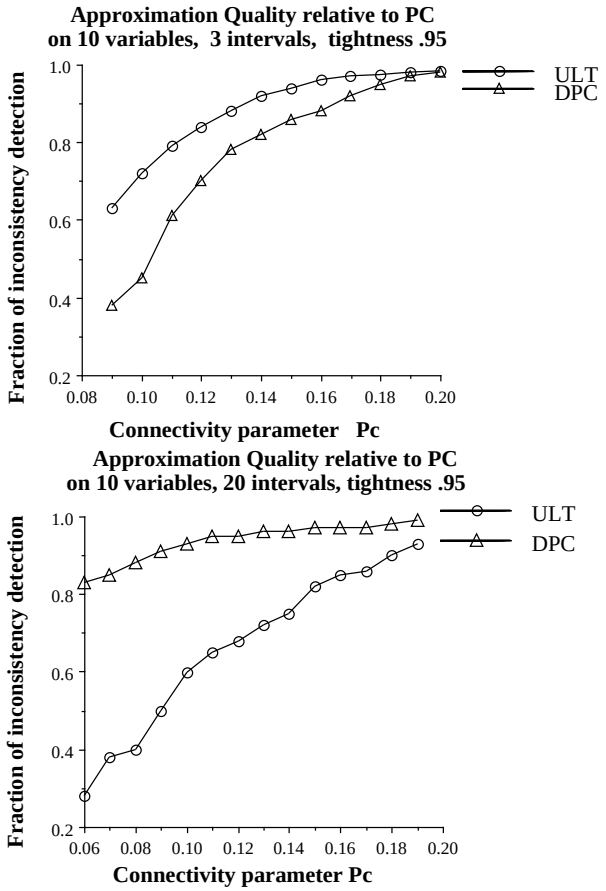


Figure 9: Quality of the approximation vs connectivity on problems with 10 variables, tightness .95 and range $[0, 600]$. We measured on problems of 3 intervals (top) where each point represents 1000 runs, and on 20 intervals (bottom) where each point represents 100 runs.

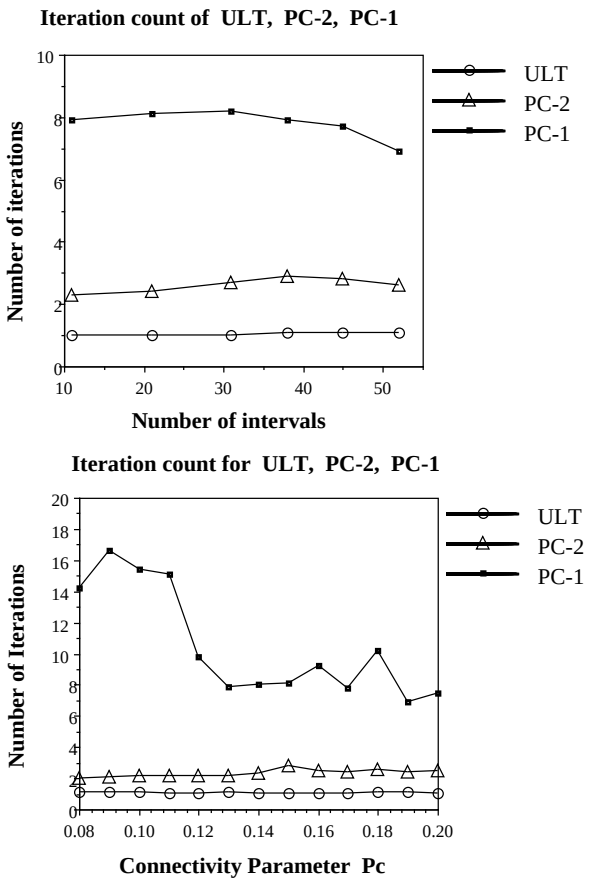


Figure 10: The number of iterations ULT, PC-2 and PC-1 performed (excluding the termination iteration) on problems with 10 variables, 20 intervals, $P_C = .14$, and tightness .95. Each point represents 20 runs.

ULT and DPC increases. Note again that ULT is more accurate for a small number of intervals while *DPC* dominates for large number of intervals.

We measured the number of iterations performed by ULT, PC-2, and PC-1[2] (Figure 10). We observe that for our benchmarks, ULT performed 1 iteration (excluding the termination iteration) in most of the cases, while PC-1 and PC-2 performed more (DPC performs only one iteration).

In the second set of experiments we tested the power of *ULT*, *DPC* and *PC-2* as preprocessing to backtracking. Without preprocessing, the problems could not be solved using the naive backtracking. Our preliminary tests ran on 20 problem instances with 10 variables and 3 intervals and none terminated before 1000000 STP checks. We therefore continued with testing backtracking following preprocessing by ULT, DPC and PC-2.

[2] A brute force path-consistency algorithm (Dechter Meiri & Pearl).

Testing the consistency of a labeling requires solving the corresponding STP. An inconsistent STP represents a dead-end. Therefore, we counted the number of inconsistent STPs tested before a single consistent one was found and the overall time required (including preprocessing). The results are presented in Figure 11 on a **logarithmic scale**. We observe that ULT was able to remove intervals effectively and appears to be the most effective as a preprocessing procedure. For additional experiments with path-consistency for qualitative temporal networks see (Ladkin & Reinefeld 92).

Summary and conclusion

In this paper we presented a polynomial approximation algorithm to path-consistency for temporal constraint problems, called ULT. Its complexity is $O(n^3ek+e^2k^2)$, in contrast to path-consistency which is exponential in k, where n is the number of variables, e is the number of constraints and k is the maximal number of intervals per constraint. We also argued that *ULT* can be used to effectively identify some *path-redundancies*.

We evaluated the performance of DPC and ULT em-

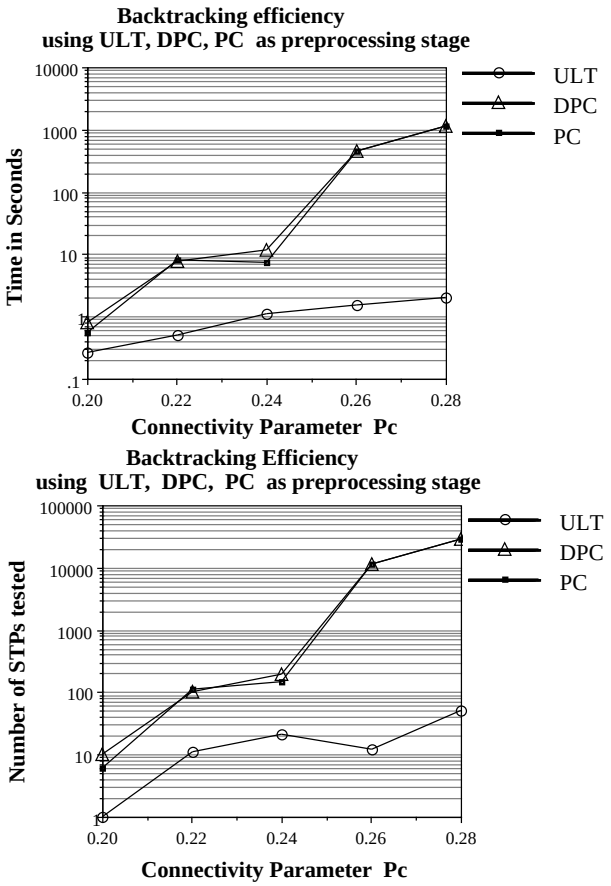


Figure 11: Backtracking performance following preprocessing by ULT, PC-2 and PC-1 respectively, on problems with 10 variables, 3 intervals, and tightness .95. Each point represents 20 runs. The time includes preprocessing.

pirically by comparing their run-time and quality of output relative to PC-2. The results show that while *ULT* is always very efficient, it is most accurate (i.e. it generates output closer to PC-2) for problems having a small number of intervals and high connectivity. Specifically, we saw that: 1. The complexity of both PC-2 and DPC grows exponentially in the number of intervals, while the complexity of ULT remains almost constant. 2. When the number of intervals is small, DPC is faster but produces weaker approximations relative to *ULT*. When the number of intervals is large, DPC is much slower but more accurate. 3. For a large number of intervals, DPC computes a very good approximation to PC-2, and runs about 10 times faster. 4. When used as a preprocessing procedure before backtracking, ULT is shown to be 10 times more effective then either DPC or PC-2.

Finally, our experimental evaluation is by no means complete. We intend to conduct additional experiments with wider range of parameters and larger problems.

References

Allen, J.F. 1983. Maintaining knowledge about temporal intervals. *CACM* 26(11):832-843.

Dechter, R.; Meiri, I.; Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49:61-95.

Dechter, R.; 1992. "Constraint Networks", *Encyclopedia of Artificial Intelligence*, (2nd ed.) (Wiley, New York, 1992) 276-285.

Dechter, R.; Pearl, J. 1988. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence* 34:1-38.

Dechter, R. 1990. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence* 41:273-312.

Dechter, R.; Dechter, A. 1987. Removing redundancies in constraint networks. In *Proc. of AAAI-87*, 105-109.

Dean, T.M.; McDermott, D. V. 1987. Temporal data base management. *Artificial Intelligence* 32:1-55.

Freuder, E.C. 1985. A sufficient condition of backtrack free search. *JACM* 32(4):510-521.

Ladkin, P. B.; Reinefeld, A. 1992. Effective solution of qualitative interval constraint problems. *Artificial Intelligence* 57:105-124.

Meiri, I. 1991. Combining qualitative and quantitative constraints in temporal reasoning. In *Proc. of AAAI-91*, 260-267.

Meiri, I.; Dechter, R.; Pearl, J. 1991. Tree decomposition with application to constraint processing. In *Proc. of AAAI-91*, 241-246.

Kautz, H.; Ladkin, P. 1991. Intergating metric and qualitative temporal reasoning", In *Proc. of AAAI-91*, 241-246.

Koubarakis, M. 1992. Dense time and temporal constraints with $\neq$. In *Proc. KR-92*, 24-35.

Poesio, M.; Brachman, R. J. 1991. Metric constraints for maintaining appointments: Dates and repeated activities. In *Proc. AAAI-91*, 253-259.

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

Vilain, M.; Kautz, H. 1986. Constraint propagation algorithms for temporal information. In *Proc AAAI-86*, 377-382.