Processing Temporal Constraint Networks *

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

This paper describes new algorithms for processing quantitative and qualitative Temporal Constraint Satisfaction Problems (TCSP). In contrast to discrete Constraint Satisfaction Problems (CSP), enforcing path-consistency on quantitative TCSP is exponential, due to the fragmentation problem. Identifying the fragmentation problem allows us to design several efficient polynomial algorithms that are effective for detecting inconsistencies and explicating some implicit constraints. We discuss the tradeoffs between their effectiveness and efficiency both theoretically and empirically.

1 Introduction

Problems involving temporal constraints arise in various areas such as scheduling [15, 14], planning [13], temporal databases [4, 8, 10] and common sense reasoning [6, 17]. Several formalisms for expressing and reasoning about temporal constraints have been proposed, most notably, Allen's interval algebra [1], Vilain and Kautz's point algebra [20], linear inequalities of Malik and Binford [11] Valdez-Perez [18], Dean and McDermott's Time Map Management [4], Dechter, Meiri and Pearl's Temporal Constraint Satisfaction Problems (TCSP) [3] and Meiri's combined model of quantitative and qualitative networks [8]. Each of these representations is supported by specialized constraint-propagation algorithms. This paper extends the work on TCSPs by providing new algorithms which improve significantly on the algorithms suggested in [3, 16] and render the model practical.

The combined model [8] builds upon Allen's Interval Algebra. It offers a general network-based computational model for temporal reasoning which is capable of handling both qualitative and quantitative information. Variables represent either time points or time intervals, and constraints may be either metric (between points) or qualitative disjunctive relations. The model facilitates the following tasks:

1. Finding all feasible times a given event can occur.
2. Finding all relationships between two given events.
3. Finding one or more consistent scenarios.
4. Systematically explicating implicit constraints.
5. Representing the data in a *minimal network* form.

It is well known that the above tasks are NP-Hard [3]. The source of complexity stems from specifying relationships between pairs of variables as *disjunctions* of atomic relations.

Disjunctive constraints often arise in scheduling and planning. Logistics constraints, for example, are a possible source of disjunction. Consider the following example:

Example 1: Today's date is Jan 1. The task is to deliver a large cargo from New York to Los Angeles to be picked up either between Jan 8 and Jan 10, between Jan 15-16 or between Jan 18-19. The cargo must go through Chicago and Dallas. Using several big air transports, the cargo can be delivered from New York to Chicago in 1-2 days, but on the ground it takes about 10-11 days. From Chicago to L.A. it takes 3-4 days using numerous small air transports, while with ground shipments the whole operation requires 13-15 days. From Chicago to Dallas it takes 2-4 days by air or 10-13 days on the ground. From Dallas to L.A. there are several options that require 5-7 days, 11-12 days, 13-14 days or 16-19 days. The same personnel engaged in the transport from Chicago to Dallas must also supervise another direct transport from Newark to Phoenix.

Given the above constraints, we are interested in answering queries such as: "are these constraints satisfiable ?" or "when should the cargo be in Dallas ?" or "can the cargo arrive in L.A. on Jan 18-19 ?". The model of Temporal Constraint Satisfaction Problems (TCSP) provides a representation with which such queries can be answered.

The two central tasks in constraint processing are (1) to determine consistency of a set of constraints and (2) to

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

make implicit constraints explicit. Determining consistency usually involves computing a single solution while explicating all implicit constraints involves computing the minimal network.

Since determining consistency and computing the minimal network is not tractable, it is common to use polynomial approximation algorithms. The most common method is to enforce path-consistency [3]. As opposed to discrete CSPs, enforcing path-consistency on TCSPs is exponential. In [16] we introduce a polynomial algorithm for approximating path-consistency, called Upper-Lower-Tightening (ULT). Here we present an improvement on that algorithm, called Loose Path-Consistency (LPC), together with three variants called LPC-2, Loose Directional Path-Consistency (LDPC) and Partial LPC (PLPC). These algorithms are extended to process general networks of qualitative and quantitative constraints. The relative and absolute effectiveness of these algorithms is determined both theoretically and empirically.

The quantitative empirical evaluation is performed on selected benchmarks. We observe that the difficult problems lie in the region in which about half of the instances generated are solvable (and half not); this region is often referred to as the *transition region*. Using these problem distributions as benchmarks, we report that the methods presented in this paper are capable of improving efficiency of backtrack search on small problems by orders of magnitude.

The paper is organized as follows. Section 2 describes the TCSP model. Section 3 described several new polynomial algorithms and section 4 describes a complete backtracking algorithm. Finally, section 5 presents an empirical evaluation and section 6 concludes.

2 Temporal Constraint Networks

A combined qualitative and quantitative constraint network [8] involves a set of variables and a set of binary constraints over pairs of variables. There are two types of variables, point and interval variables. The constraint T_{ij} between a pair of variables, X_i, X_j is described by specifying a set of allowed relations, namely

$$T_{ij} \stackrel{\text{def}}{=} (X_i\ r_1\ X_j)\ \vee\ \cdots\ \vee\ (X_i\ r_k\ X_j). \qquad (1)$$

There are three types of relations:

1. A point-point relation between two point variables, X_i, X_j, is quantitative, and has the form

   $$a \leq X_j - X_i \leq b$$

   where a and b are constants.

2. A point-interval relation between a point variable and an interval variable, is qualitative, and is a subset of { **before, starts, during, finishes, after** } abbreviated { b, s, d, f, a } respectively (see Table 1 for illustration).

3. An interval-interval relation between two interval variables is qualitative, and is a subset of

   $$\left\{ \begin{array}{c} \textbf{before}, \textbf{after}, \textbf{meets}, \textbf{met-by}, \\ \textbf{overlaps}, \textbf{overlaps-by}, \textbf{during}, \textbf{contains}, \textbf{equals}, \\ \textbf{starts}, \textbf{started-by}, \textbf{finishes}, \textbf{finished-by} \end{array} \right\}$$

   abbreviated { b,bi, m,mi, o,oi, d,di, =, s,si, f,fi } respectively (see Table 2 for illustration).

Table 1: The 5 qualitative point-interval relations.

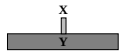
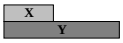
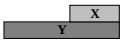
Relation	Symbol	Inverse	Example
X before Y	b	bi	
X starts Y	s	si	
X during Y	d	di	
X finishes Y	f	fi	
X after Y	a	ai	

Table 2: The 13 qualitative interval-interval relations.

Relation	Symbol	Inverse	Example
X before Y	b	bi	
X equal Y	=	=	
X meets Y	m	mi	
X overlaps Y	o	oi	
X during Y	d	di	
X starts Y	s	si	
X finishes Y	f	fi	

The structure of a Temporal Constraint Network can be represented by a constraint graph, in which a node corresponds to a variable (point or interval) and an edge is labeled by the elements in the disjunctive constraint, namely either by the set of intervals (if point-point constraint) or by the set of allowed qualitative relations. (see Figure 1).

Example 2: Consider the logistics problem in Example 1. Let X_1 be the time the cargo arrived in Chicago, X_2 the time it arrived in Dallas and X_3 the time it arrived in L.A. Let I_1 be the time interval during which the cargo was shipped from Chicago to Dallas and let I_2 be the time at which the smaller cargo was shipped directly from Newark to Phoenix. Let X_0 ="Jan 1". The constraint graph is shown in Figure 1. The fact that the same personnel need to supervise the shipment from Chicago to Dallas and the shipment from Newark to Phoenix is represented by the constraint specifying

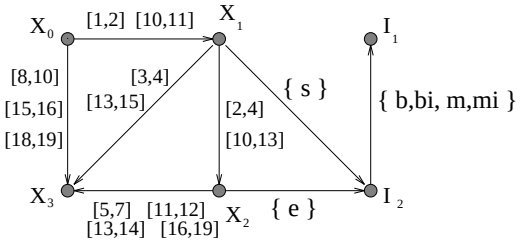


Figure 1: The constraint graph of the logistics problem.

that I_1 and I_2 do not overlap. Such a constraint can not be represented by point-point relations.

2.1 The quantitative TCSP Model

We first restrict the discussion to quantitative networks that involve only point variables. Algorithms on this model are extended to process combined qualitative and quantitative networks.

A quantitative 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

$$T \stackrel{\text{def}}{=} (I_1, \ldots, I_n) = \{[a_1, b_1], \ldots, [a_n, b_n]\}.$$

For a unary constraint T_i over X_i, the set of intervals restricts its 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

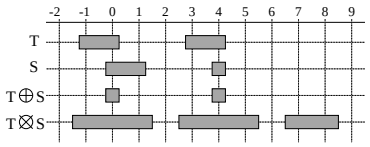
$$(a_1 \leq X_j - X_i \leq b_l) \cup \ldots \cup (a_n \leq X_j - X_i \leq b_n).$$

All intervals are assumed to be pairwise disjoint.

A TCSP be represented by a *directed constraint 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 as illustrated in Figure 1. A special time point X_0 is introduced to represent the "beginning of the world". All times can be specified relative to X_0 and thus each unary constraint T_i can be represented as a binary constraint T_{0i} (having the same interval representation).

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 such that the network describes the same set of solutions. The *minimal network* is such that all its *domains* and *constraints* are *minimal*.

Path-Consistency algorithms are commonly used to tighten the network. To enforce path-consistency on TCSPs, there is a need to define the $\oplus, \otimes$ operations.



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

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

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 $r = t + s$.

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 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 repeatedly applying the relaxation operation $T_{ij} \leftarrow T_{ij} \oplus (T_{ik} \otimes T_{kj})$, using algorithm PC-2, as described in Figure 3. Some problems may benefit from a weaker version, called DPC, which is more efficient.

The complexity of PC-2 and DPC 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, i.e. the difference between the lowest and highest numbers specified. In contrast to discrete CSPs, however, enforcing path-consistency on TCSPs is problematic when the range R is large or the domains are continuous. An upper bound on the number of intervals in $T \otimes S$ is $|T| \cdot |S|$, where $|T|, |S|$ are the number of intervals in T, S respectively. As a result, the number of intervals in the path-consistent network can be exponential in the number of intervals per constraint in the input network[1].

Example 3: To illustrate the difficulty in enforcing path-consistency, consider the network presented in Figure 4, with 3 variables, 3 constraints and 3 intervals per constraint. After enforcing path-consistency, two constraints remain unchanged in the path-consistent network while the third is broken into 10 subintervals. As this behavior is repeated over several triangles in the network, the number of intervals becomes exponential.

[1] yet bounded by R when integer domains are used

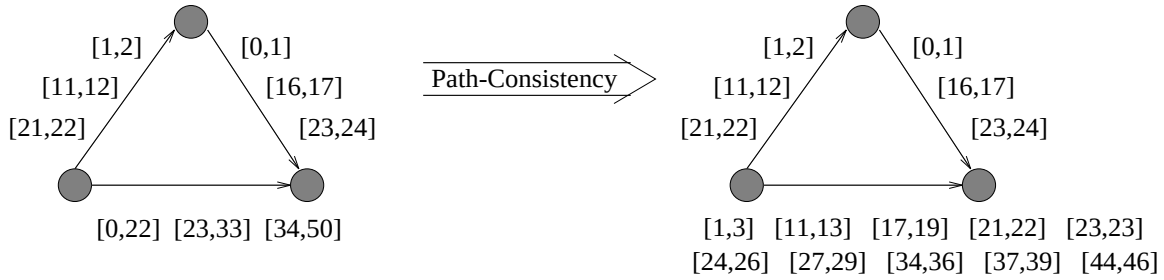


Figure 4: The fragmentation problem.

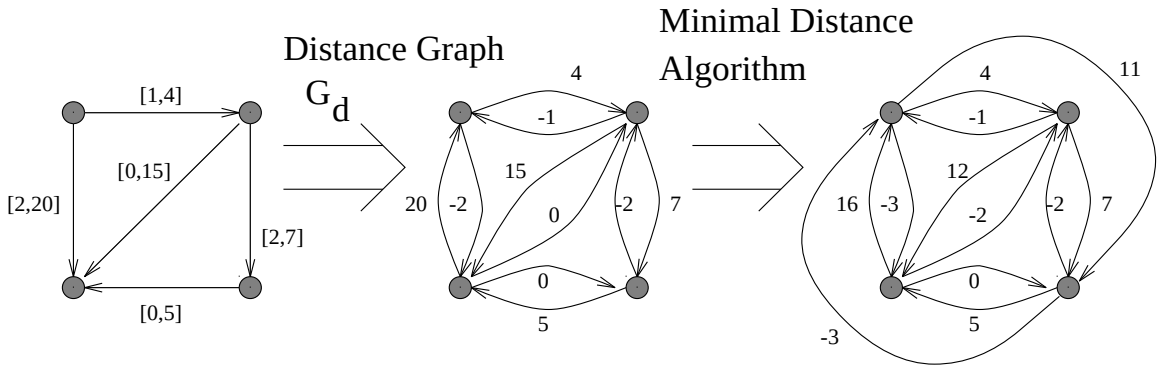


Figure 5: Processing an STP.

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. $Q \leftarrow Q \cup\ RELATED\text{-}PATHS((i,k,j))$
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 [3].

A special class of problems which does not exhibit such an exponential blow-up is the *Simple Temporal Problem* (STP) [3], where only a single interval is specified per constraint.

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 $X_j - X_i \leq w_{ij}$, as illustrated in Figure 5. An STP is consistent iff the corresponding d-graph G_d has no negative cycles and the minimal network of the STP corresponds to the *minimal distances* of G_d. An all-pairs shortest path procedure (Figure 5) is equivalent to enforcing path-consistency. Therefore, PC-2 is polynomial and complete for STPs [3].

The source of intractability of enforcing path-consistency stems from the fact that the relaxation operation $T_{ij} \leftarrow T_{ij} \oplus (T_{ik} \otimes T_{kj})$ may increase the number of intervals in T_{ij}. As a result, we attempt to approximate path-consistency. The basic algorithm for approximating path-consistency, called Upper Lower Tightening (ULT) [16], utilizes the fact that an STP is tractable. 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. The key observation is that this process can only decrease the number of intervals per constraint.

3 Loose Path-Consistency

In the following, an improvement of ULT is presented, called *Loose Path-Consistency (LPC)*, which is more effective in removing intervals. The idea is to use a loose intersection operation, denoted $\triangleleft$, as defined below.

Definition 3 : Let $T = \{I_1, I_2, \ldots, I_r\}$ and $S = \{J_1, J_2, \ldots, J_s\}$ be two constraints. The *loose intersection*, $T \triangleleft S$ consists of the intervals $\{I'_1, \ldots, I'_r\}$ such that $\forall i\ \ I'_i = [L_i, U_i]$ where $[L_i, U_i]$ are the lower and upper bounds of the intersection $I_i \oplus S$.

Algorithm Loose Path-Consistency (LPC)

1. **input:** N
2. $N'' \leftarrow N$
3. **repeat**
4. $\quad N \leftarrow N''$
5. $\quad$ compute N', N''.
6. **until** $\exists i,j \quad (T''_{ij} = \phi)$; inconsistency, or
   $\quad$ or $\forall i,j \quad |T''_{ij}| = |T_{ij}|$; no interval removed.
7. **if** $\exists i,j \quad (T''_{ij} = \phi)$ **then output** "inconsistent."
   $\quad$ **else output:** N''.

Figure 6: The Loose Path-Consistency (LPC) algorithm.

It is easy to see that (1) the number of intervals in T_{ij} is not increased when assigning $T_{ij} \leftarrow T_{ij} \triangleleft (T_{ik} \otimes T_{kj})$, (2) $\forall k \quad T_{ij} \supseteq T_{ij} \triangleleft (T_{ik} \otimes T_{kj}) \supseteq T_{ij} \oplus (T_{ik} \otimes T_{kj})$ and (3) $T \triangleleft S \neq S \triangleleft T$.

Example 4: Let $T = \{[1,4],[10,15]\}$ and $S = \{[3,11],[14,19]\}$. Then $T \triangleleft S = \{[3,4],[10,15]\}$, $S \triangleleft T = \{[3,11],[14,15]\}$ while $S \oplus T = \{[3,4],[10,11],[14,15]\}$.

By Definition 2, a constraint T_{ij} is path-consistent iff $T_{ij} \subseteq \oplus_{\forall k}(T_{ik} \otimes T_{kj})$. When replacing the intersection operator $\oplus$ with the loose intersection operator $\triangleleft$, we solve the fragmentation problem.

Definition 4: We define N', N'' as follows: (see Figure 7 for a sample trace)

- N' is derived from N by assigning
$$T'_{ij} = \oplus_{\forall k}(T_{ik} \otimes T_{kj}).$$
- N'' is derived from N' by loosely intersecting
$$T''_{ij} = T_{ij} \triangleleft T'_{ij}.$$

Algorithm LPC is presented in Figure 6. The network N' is a relaxation of N and therefore loosely intersecting it with N results in an equivalent network.

Lemma 1: *Let N be the input to LPC and R be its output.*

1. *The networks N and R are equivalent.*
2. *Every iteration of LPC (excluding the last) removes at least one interval from one of the constraints.*

Proof: *Part 1: The network N' is a relaxation of N. Therefore* loosely intersecting *N' with N results in an equivalent network. Part 2: If no interval was removed, the algorithm terminates.* □

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

Proof: *Computing N' requires processing every triangle in the network once, thus requires $O(n^3k^2)$ steps. Because in every iteration at least one interval is removed, there are at most ek iterations, resulting in a complexity of $O(n^3k^3e)$.* □

Algorithm LPC-2

1. $Q \leftarrow \{(i,k,j) | (i<j) \text{ and } (k \neq i,j)\}$
2. **while** $Q \neq \{\}$ **do**
3. $\quad$ select and delete a path (i,k,j) from Q
4. $\quad T'_{ij} \leftarrow T_{ij} \triangleleft (T_{ik} \otimes T_{kj})$
5. $\quad$ **if** $T'_{ij} = \{\}$ **then** exit (inconsistency)
6. $\quad$ **if** $|T'_{ij}| < |T_{ij}|$ **then**
   $\quad\quad Q \leftarrow Q \cup$ *RELATED-PATHS*$((i,k,j))$
7. $\quad T_{ij} \leftarrow T'_{ij}$
8. **end-while**

Algorithm DLPC

1. **for** $k \leftarrow n$ downto 1 by -1 **do**
2. $\quad$ **for** $\forall i,j < k$ such that $(i,k),(k,j) \in E$ **do**
3. $\quad\quad T'_{ij} \leftarrow T_{ij} \triangleleft (T_{ik} \otimes T_{kj})$
4. $\quad\quad$ **if** $T'_{ij} = \{\}$ **then** exit (inconsistency)
5. $\quad\quad$ **if** $|T'_{ij}| < |T_{ij}|$ **then** $E \leftarrow E \cup (i,j)$
6. $\quad\quad T_{ij} \leftarrow T'_{ij}$
7. $\quad$ **end-for**
8. **end-for**

Figure 9: Algorithms LPC-2 and DLPC.

Example 5: Consider the network in Figure 4. As shown in Figure 8, applying LPC tightens the network and does not increase the number of intervals in any of the constraints. For example, loosely intersecting $[0,22]$ with the 9 intervals of $([1,2],[11,12],[21,22]) \otimes ([0,1],[16,17],[23,24])$ results in one interval $[1,22]$. The approximation is due to using loose intersection rather than strict intersection.

Algorithm LPC is conceptually different than PC-2. While an iteration of LPC is divided into two sequential stages that involve the whole network, algorithm PC-2 uses simpler operations and allows parallel execution. We therefore present two variations on LPC, called Loose Path-Consistency-2 (LPC-2) and Directional Loose Path-Consistency (DLPC), as presented in Figure 9. These algorithms differ from PC-2 and DPC only by using the loose intersection $\triangleleft$ operator instead of the full intersection $\oplus$ operator.

Theorem 2: *Given a network N, let n be the number of variables, e be the number of constraints and k be the maximum number of intervals per constraint.*

1. *Algorithms LPC-2 and DLPC terminate in $O(nk^2(n^2+ke))$, $O(n^3k^2)$ steps respectively and compute an equivalent network.*
2. *Algorithm LPC computes a tighter networks than algorithm ULT; algorithm LPC-2 computes a tighter network than DLPC; algorithm LPC computes a tighter network than LPC-2.*

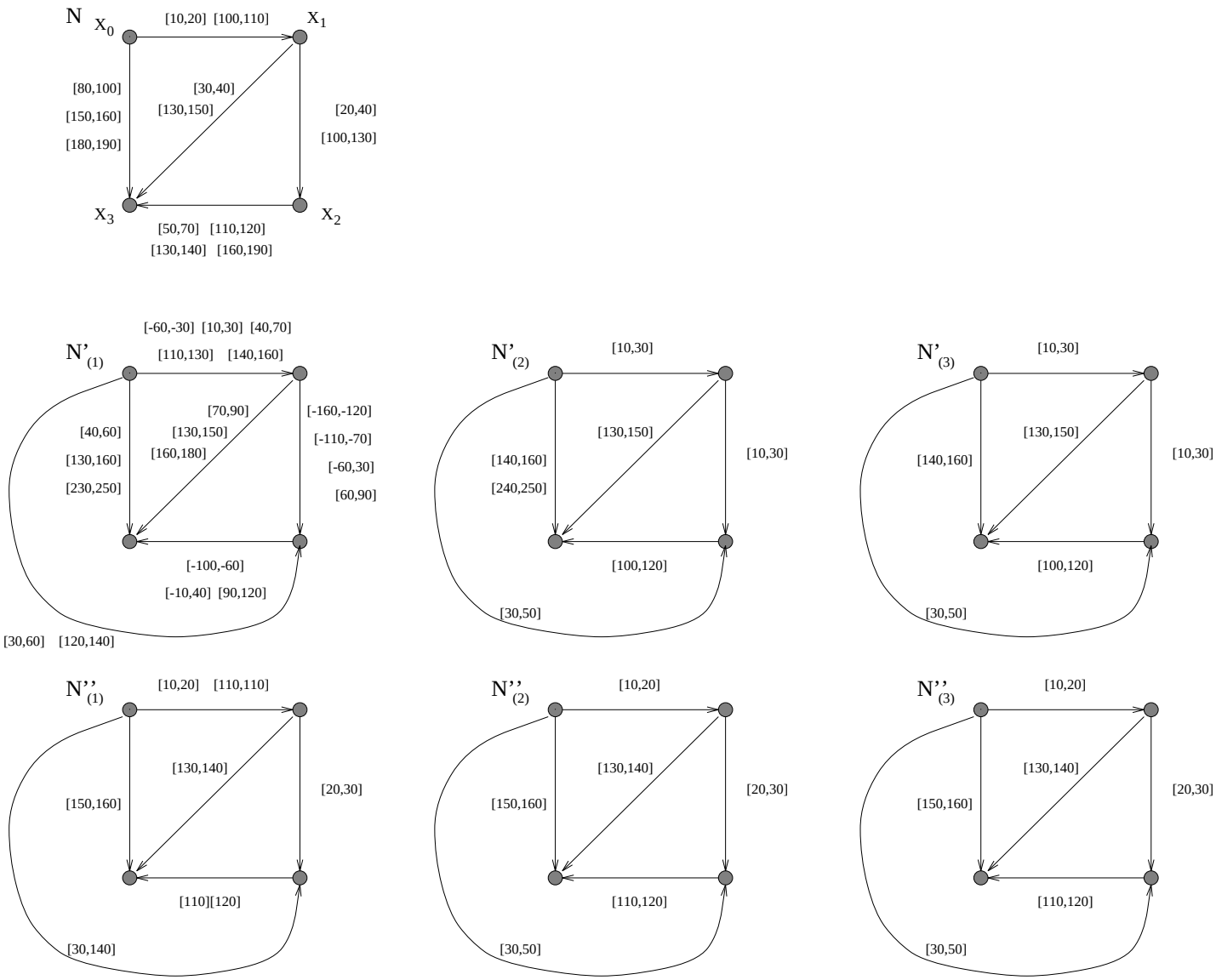


Figure 7: A sample run of LPC on the quantitative constraints specified in Example 1. We start with N and compute $N'_{(1)}$, $N''_{(1)}$. Thereafter, we perform a second iteration in which we compute $N'_{(2)}$, $N''_{(2)}$ and finally, in the third iteration, there is no change. The first iteration removes 7 intervals while the second iteration removes a single interval. In addition, LPC explicates an induced constraint T_{02}, and thus is capable of inferring new facts that were not specified initially about the times event X_2 can occur. Note that applying ULT on the network N will have no effect.

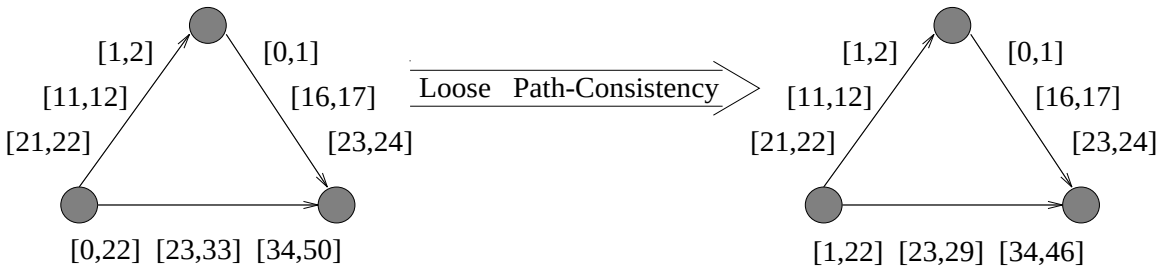


Figure 8: Solving the fragmentation problem.

Another variation of LPC is Partial LPC (PLPC) in which the relaxation operation $T_{ij} \leftarrow T_{ij} \triangleleft (T_{ik} \otimes T_{kj})$ is performed only if T_{ij} and at least one of T_{ik}, T_{kj} is specified in the input network (i.e. non-universal).

The relative effectiveness of the various algorithms is measured by the relative tightness of the networks they compute. The partial order with respect to effectiveness (tightness) is presented in Figure 10. A directed edge from algorithm $\mathcal{A}_1$ to $\mathcal{A}_2$ indicates that $\mathcal{A}_2$ computes a tighter network than $\mathcal{A}_1$. Note that algorithms PC-2 and DPC are exponential.

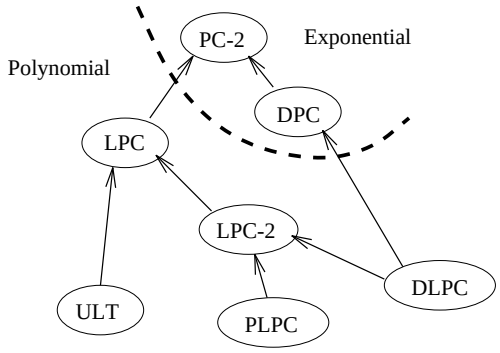


Figure 10: The partial order on the effectiveness.

3.1 Answering Queries

Path-consistency algorithms and its variations may tighten all the constraints, including those that were not specified (i.e. universal) in the input network. As a result the query "when can event X_i occur ?" can be answered by simple table lookup, even when the constraint T_{0i} is not specified in the input network. Similarly, when asking "how long after event X_i can event X_j occur ?", the answer is given by the constraint T_{ij}.

However, the constraints algorithm LPC computes provide only approximate answers. Supposed we have already preprocessed with LPC and we are given the query "can event X_i occur at time t ?". If $t \notin T_{0i}$ then the answer is 'No'; otherwise, the answer is 'Maybe' because there is no guarantee that the event can occur at every point in T_{0i}. For example, consider the network in Figures 4,8. The path-induced constraint is broken into 10 subintervals while LPC computes a looser constraint of 3 intervals. Thus, there are points that are allowed by the Loose-Path-Consistent constraint in Figure 8 but not allowed by the Path-Consistent network in Figure 4. Note that enforcing path-consistency is also not complete, i.e. is not sufficient to guarantee a 'Yes' answer.

3.2 Extending LPC to Combined Networks

We next extend algorithm LPC to process networks of combined qualitative and quantitative constraints. As defined in Section 2, the combined model involves three types of constraints: point-point (quantitative), point-interval and interval-interval (qualitative). To extend the applicability of LPC to the combined qualitative and quantitative networks, we modify the semantics of the $\triangleleft, \otimes$ operators.

Let T_{ij}, T_{ik}, T_{kj} be the constraints on variables X_i, X_j, X_k. For computing $T'_{ij} \leftarrow T_{ij} \triangleleft (T_{ik} \otimes T_{kj})$ we distinguish between five exhaustive and mutually exclusive cases:

Case 1: If X_i, X_j, X_k are interval variables then Allen's transitivity table [1] is used to compute $T_{ik} \otimes T_{kj}$ and the $\triangleleft$ operator is interpreted as the usual intersection operator.

Case 2: If both X_i, X_j are interval variables and X_k is a point variable then Meiri's transitivity tables [8] are used to compute $T_{ik} \otimes T_{kj}$ and the $\triangleleft$ operator is interpreted as the usual intersection.

Case 3: If exactly one of X_i, X_j is an interval variable and X_k is a point variable, then the quantitative point-point constraint, T_{ik} or T_{kj}, is translated into a qualitative point-point constraint (using $<, >, =$) and Meiri's transitivity tables [8] are used to compute $T_{ik} \otimes T_{kj}$; the $\triangleleft$ operator is interpreted as the usual intersection.

Case 4: If X_i, X_j are point variables and X_k is an interval variable then $T_{ik} \otimes T_{kj}$ is computed using Meiri's tables [8]. If $T_{ik} \otimes T_{kj} \neq \{<, >\}$ then the resulting constraint is translated into a single interval and the $\triangleleft$ operator is interpreted as the $\oplus$ operator in Definition 1; otherwise, to avoid increasing the number of intervals in T_{ij}, we set $T'_{ij} \leftarrow T_{ij}$, i.e. no change .

Case 5: If all of X_i, X_j, X_k are point variables then the composition operation used is described by Definition 1 and the $\triangleleft$ operator is described in Definition 3.

With these new definitions of the operators $\otimes, \triangleleft$, we can apply algorithms LPC, LPC-2, DLPC as described in Figures 6,9.

4 General Backtracking.

Algorithm LPC and its variants are useful for detecting inconsistencies and explicating implicit constraints, but they do not find a solution (scenario). A brute-force algorithm for solving a temporal network decomposes it into separate tractable subnetworks by selecting a single interval from each quantitative constraints and a single relation from a qualitative constraint [8, 3]. Each subnetwork is then solved separately and the solutions are combined. Alternatively, a naive backtracking algorithm will successively label constraints by basic labels, as long as the resulting labeling is consistent [8, 3]. Once inconsistency is detected, the algorithm backtracks. A more sophisticated algorithm may perform *forward checking* to reduce the number of future possible interval assignment during the labeling process.

Definition 5: [8] A *basic label* of an arc $i \rightarrow j$ is a selection of (1) a single interval from the interval set T_{ij} for quantitative constraints, and (2) a single relation for qualitative constraints. A *singleton labeling* of N is a selection of a basic label for *all* the constraints in N and a *partial labeling* of N is such that *some* constraints are assigned basic labels.

Backtracking on temporal constraint networks involves repeatedly selecting a basic label from constraints and propagating the implications of this instantiation throughout the network. For processing qualitative networks, Ladkin and Reinefeld [7] proposed the following algorithm. First perform path-consistency. If no inconsistency was detected, select one constraint, label it by an atomic relation from that constraint and enforce path-consistency again to propagate the implications of that labeling (i.e. perform forward checking). Repeat instantiating constraints and enforcing path-consistency until all constraints are labeled by a single atom. If, at some point, inconsistency was detected, backtrack. The improvements Ladkin and Reinefeld introduce in their implementation are (1) to perform path-consistency using Belman-Ford algorithm, and (2) not to perform path-consistency on the subnetwork that is already singly labeled (since it is already consistent). While Ladkin and Reinefeld's algorithm is applicable to qualitative interval algebra networks only, our algorithm is applicable to the combined model. Instead of enforcing complete Path-Consistency on the quantitative constraints we propose to enforce Loose Path-Consistency. In addition, we propose two improvements: (1) instead of using a stack that requires $O(n^4)$ space[2] we use some indexing technique that eliminates the need for a stack and requires to keep only two copies of the input network; (2) we do not instantiate constraints that were universal in the input network (but became non-universal as a result of constraint propagation).

In addition to propagating constraints during backtrack search, algorithms ULT, LPC and their variations are useful for preprocessing *before* initiating search. These algorithms reduce the number of disjuncts in the constraints, i.e. the number of intervals in quantitative constraints and the number of allowed relation in qualitative constraints. As a result, the branching factor is reduced, less dead-ends are encountered and the search becomes more efficient. This is in contrast to algorithms PC-2 and DPC, which are not useful for preprocessing before search since they increase the fragmentation, thus increase the branching factor and render backtrack search less efficient [16].

5 Empirical Evaluation

In this section we evaluate the algorithms presented in previous sections using artificially generated problem distributions as a benchmark. Although we are capable of solving large problems, too much time is required for performing systematic analysis and obtaining statistically significant results. Therefore, we report results obtained on small problems.

In our benchmarks, the number of constraints, e, was fixed and the constraints were chosen such that every possible subset of e constraints out of the total $\binom{n}{2}$ is equally likely to occur. In addition, the generator guarantees that the constraint graph is connected. The parameters with which problem instances were generated are:

- the number of variables n,
- the number of constraints e,
- for quantitative point-point constraint:
  - the number of intervals per constraint k,
  - the range of the constraints R, and
  - the tightness of the constraints α,
- the number of atoms in every point-interval constraint β,
- the number of atoms in every interval-interval constraint γ.

5.1 Comparing Path-Consistency Algorithms

In Theorem 2 and Figure 10 we have described the qualitative relationships between the various algorithms. We next present a quantitative empirical comparison. In [16] algorithms PC-2 and LPC were compared with ULT. It was observed that both PC-2 and DPC are exponential in the number of intervals per constraint in the input network; the complexity of algorithm ULT was almost constant. Despite the fact that ULT was orders of magnitude more efficient, it was able to detect inconsistency in most of the cases PC-2 did. As we will show next, algorithm LPC and its variations are significantly more effective than ULT. Since we have already shown that LPC is worst case polynomial, there is no need to report comparison with PC-2 and DPC.

We compare the effectiveness and efficiency of algorithms LPC-2 and DLPC. By effectiveness we mean the ability to detect inconsistencies and by efficiency we mean the execution time. We measure effectiveness with respect to algorithm LPC-2 rather than to PC-2 because the former is polynomial (and almost as efficient as ULT) while the latter is exponential. The columns labeled "Acc of PLPC" and "Acc of DLPC" specify the fraction of cases PLPC, DLPC detected inconsistency given that LPC-2 did. The columns labeled "# Op $< alg >$" describe the number of revision operations $T_{ij} \leftarrow T_{ij} \triangleleft (T_{ik} \otimes T_{kj})$ made by algorithm $< alg >$. We use this measure since, being machine independent, it is more informative than execution time.

As observed in Table 3, while LPC-2 is the most effective, DLPC is more efficient. The intermediate algorithm, Partial LPC-2, seems to be the most beneficial

[2] since there are $O(n^2)$ entries of size $O(n^2)$ each - this was the major problem in [7]

# of Consts	Acc of PLPC	Acc of DLPC	Acc of ULT−2	# Op. LPC−2	# Op. PLPC	# Op. DLPC	Time LPC−2	Time PLPC	Time DLPC	Time ULT−2
32 vars, 100% interval variables (pure qualitative), 200 reps.										
250	100%	100%	100%	17K	13K	11K	0.621	0.467	0.417	0.621
300	100%	98%	100%	20K	17K	15K	0.748	0.632	0.551	0.748
350	100%	92%	100%	25K	22K	19K	0.886	0.807	0.689	0.886
400	100%	79%	100%	28K	27K	23K	1.001	0.970	0.807	1.001
450	100%	71%	100%	30K	30K	26K	1.056	1.056	0.907	1.056
500	100%	73%	100%	28K	28K	25K	0.971	0.971	0.885	0.971
32 vars, 50% interval variables (mixed), 200 reps.										
150	100%	100%	100%	13K	6K	5K	0.210	0.121	0.082	0.163
200	99%	98%	97%	18K	11K	8K	0.283	0.200	0.135	0.174
250	98%	93%	95%	23K	17K	11K	0.374	0.306	0.199	0.308
300	96%	63%	65%	26K	22K	15K	0.456	0.406	0.266	0.422
350	98%	32%	89%	27K	25K	20K	0.460	0.440	0.325	0.426
400	100%	46%	98%	24K	23K	20K	0.406	0.402	0.347	0.385
450	100%	86%	100%	20K	20K	19K	0.400	0.400	0.343	0.379
500	100%	100%	100%	16K	16K	16K	0.359	0.353	0.294	0.331
32 vars, 100% point variables (pure quantitative), 200 reps.										
150	98%	92%	90%	25K	12K	5K	0.546	0.400	0.165	0.132
200	99%	25%	15%	27K	17K	8K	0.623	0.533	0.259	0.162
250	100%	70%	45%	14K	11K	10K	0.380	0.350	0.315	0.181
300	100%	99%	77%	9K	8K	8K	0.287	0.275	0.270	0.164
350	100%	100%	94%	7K	7K	7K	0.244	0.241	0.235	0.126
400	100%	100%	100%	6K	6K	6K	0.211	0.212	0.204	0.105

Table 3: Effectiveness and efficiency of LPC-2, DLPC, Partial LPC and ULT-2. The effectiveness is measured with respect to LPC-2. The problems generated have 32 variables, tightness of interval-interval constraints is 7 relations out of 13 allowed, namely $\gamma = 7/13$, for point-interval 4 out of 5 are allowed, namely $\beta = 4/5$, and for point-point constraints $\alpha = 45\%$. The results are averaged over 200 repetitions.

since for the cost of a small degradation of effectiveness we gain a significant speedup. Algorithm ULT-2 is an extension of algorithm ULT reported in [16] capable of processing networks with combined qualitative and quantitative constraints. We observe that ULT-2 is less effective than LPC yet more efficient. In the vast majority of the cases the quality of the approximation obtained by PLPC, DLPC and ULT-2 to LPC-2 was more than 60%. Since the space of all possible parameterizations is very large, we have focused on what we believe is representative of cases which favor LPC-2, thus showing the improvements at their worst.

5.2 Backtracking

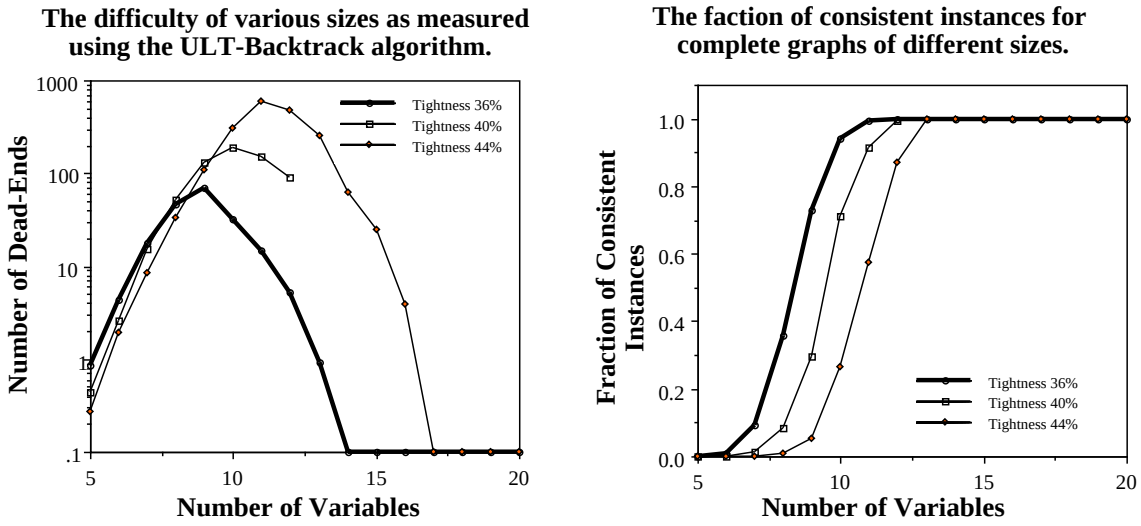
In the rest of this section we report empirical results obtained for backtracking. As we demonstrate in Figure 12, the difficulty of problems must be carefully measured in the region where there is transition from solvable to unsolvable instances. In Figure 12a we repeat the experiment conducted in [7], on *qualitative* networks, yet here we report results on *quantitative* networks, and observe that the peak in difficulty lies in the region where about 50% of the problems are satisfiable. In Figure 12b we observe the same pattern when changing the tightness of the constraints. Consequently, in the following we will emphasize using these difficult problems as benchmarks for comparing the various algorithms.

In figure 13 we compare the effectiveness of algorithms ULT and LPC for pruning dead-ends when used for both preprocessing before the search and constraint propagation within the search. The original backtrack algorithm as presented in [3] is compared with algorithm ULT reported in [16] and algorithm LPC presented in this paper.

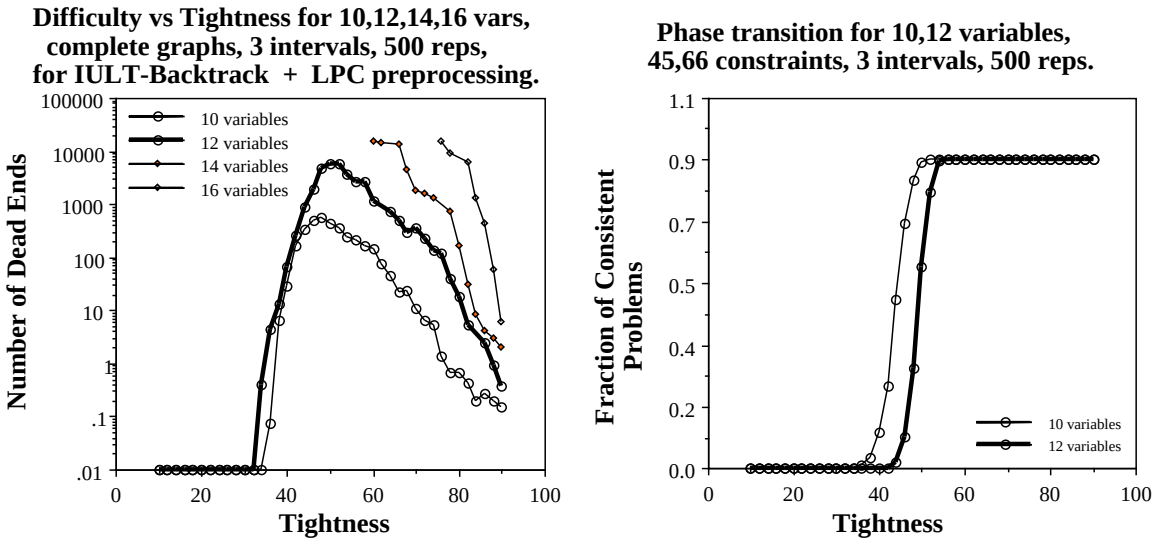
As we implemented the original algorithm, labeled "Old-Backtrack", we immediately observed that even for tiny problems with 5-6 variables the algorithm would not terminate. Surprisingly, after preprocessing with the path-consistency algorithm PC-2, problems became even harder to solve, due to the increased fragmentation. In contrast, preprocessing with ULT results in problems that on which naive backtrack search is manageable. We therefore used the "Old-Backtrack" [3] preprocessed with ULT as our reference point.

To prune dead-ends we apply either ULT or LPC after each constraint instantiation. This results in *reducing* the number of *incorrect* intervals that remain to be instantiated and thus significantly reduces the number of dead-ends. We compare two backtracking algorithms, one in which ULT is used, called ULT-Backtrack, and one in which LPC is used, called LPC-Backtrack; ULT-Backtrack is preprocessed with ULT and LPC-Backtrack is preprocessed by LPC.

The results are reported in Figure 13. We observe tremendous improvements in pruning dead-ends. Since LPC is significantly more effective than ULT (Table 3), the number of dead-ends encountered by ULT-Backtrack is orders of magnitude larger than the number of dead-



(a) The difficulty as tightness is constant.



(b) The difficulty as a function of tightness.

Figure 12

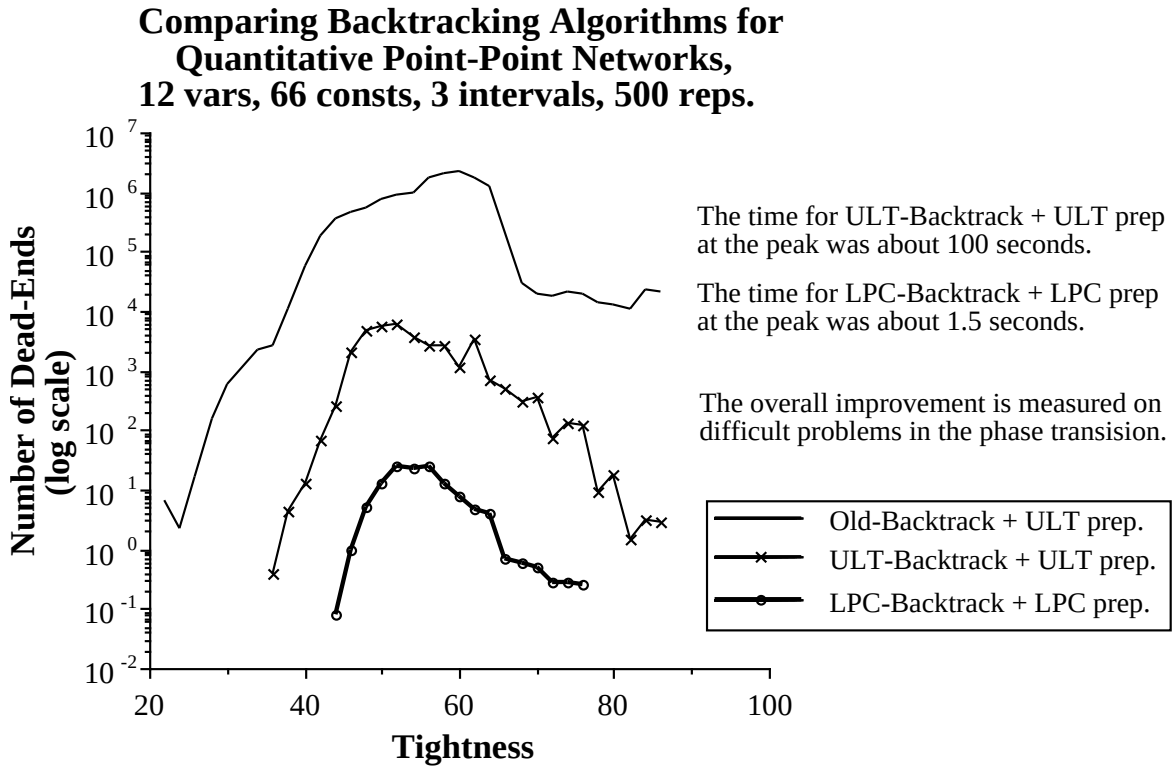
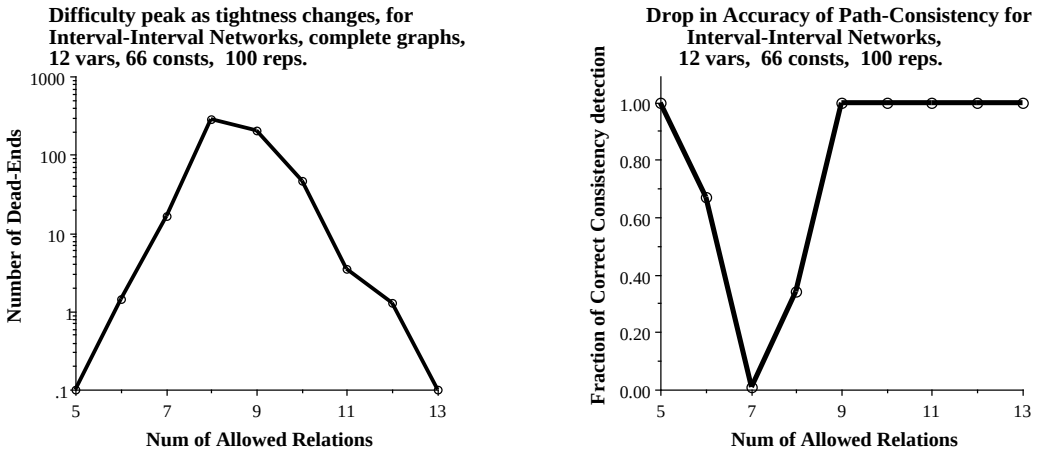


Figure 13: A comparison of various backtracking algorithms.



(a) (b)

Figure 14: The difficulty as a function of tightness.

ends encountered by LPC-Backtrack. Note that the experiments performed by Ladkin and Reinefeld [7] report that path-consistency is very effective for pruning backtrack search in *qualitative* temporal networks and here we report that the same pattern is repeated for quantitative networks.

Finally, we report results obtained with backtracking on qualitative networks in Figure 14. In the experiments performed by Ladkin and Reinefeld [7], the tightness was 50%, and most of the problems generated were inconsistent. In our experiments we observe that when 9 out of 13 interval relations were allowed, all the problems generated were consistent. As we report in Figure 14, when 6 out of 13 interval relations are allowed, the problems were about 200 times easier than those at the peak (Figure 14a). Since these results are obtained on small problems with 12 variables, by extrapolating, for large problems the difference in difficulty is huge. In addition, for 50% tightness, path-consistency is still effective since it correctly detects inconsistency in more than 60% of the cases (Figure 14b). We can therefore conclude that empirical evaluation must be performed using carefully selected problem distributions as benchmarks.

6 Conclusion

In this paper we report several new algorithms for processing Temporal Constraint Satisfaction Problems (TCSP). We identify the fragmentation problem which explains why, in contrast to discrete Constraint Satisfaction Problems (CSP), enforcing path-consistency on quantitative TCSPs is exponential. Identifying this problem allows us to provide with efficient yet effective polynomial algorithms that effectively detect inconsistency, explicate implicit constraints and prune deadends in backtrack search.

When enforcing path-consistency on quantitative TCSPs, as intervals are broken into several smaller subintervals, the number of intervals per constraint increases, resulting in exponential blowup. To avoid fragmentation, we define loose intersection. Using the loose intersection operator, we derive a new series of algorithms, called LPC, LPC-2, DLPC and PLPC that are efficient yet effective. These algorithm are compared both theoretically (qualitatively) and empirically (quantitatively).

The quantitative empirical evaluation is performed on carefully selected benchmarks. We observe that the difficult problems lie in the region in which about half of the instances generated are solvable (and half is not); this region is often referred to as the *transition region*. Using these problem distributions as benchmarks, we report that the methods presented in this paper are capable of improving the efficiency of backtrack search on small problems by orders of magnitude. Thus, by extrapolating, the speedup obtained for large problems is tremendous.

References

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

[2] Cheesman, P, Kanefsky, B., Taylor, W., 1991. Where the Really Hard Problems Are. In *Proc. of IJCAI-91*, 163-169.

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

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

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

[6] Hanks, S., McDermott, D.V., 1986. Default reasoning, nonmonotonic logics, and the frame problem. In *Proc. of AAAI-86*, 328-333.

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

[8] Meiri, I., 1991. Combining Qualitative and Quantitative constraints in temporal reasoning, Ph.D. Thesis, UCLA 1991.

[9] Mitchell, D., Selman, B., Levesque, H., 1992. Hard and Easy Distributions of SAT Problems, In *Proc. of AAAI-92*.

[10] Kautz, H., Ladkin, P. 1991. integrating metric and qualitative temporal reasoning, In *Proc. of AAAI-91*, 241-246.

[11] Malik, J., Binford, T.O., 1983. Reasoning in time and space, In *Proc. of IJCAI-83*, 343-345.

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

[13] McDermott, D.V., 1982. A temporal logic for reasoning about processes and plans, *Cognitive Science* 6:101-155.

[14] Poesio, M., Brachman, R. J. 1991. Metric Constraints for Maintaining Appointments: Dates and Repeated Activities. In *Proc. AAAI-91*, 253-259.

[15] Sateh, N., 1991. Look-Ahead techniques for Micro-opportunistic Job Shop Scheduling, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, March 1991.

[16] Schwalb, E., Dechter, R., 1993. Coping with Disjunctions in Temporal Constraint Satisfaction Problems, In *Proc. AAAI-93*, 127-132.

[17] Shoham, Y., 1988. Reasoning about Change: Time and causation from stand point of Artificial Intelligence, MIT press, Cambridge, MA 1988.

[18] Valdez-Perez. R.E., 1986. Spatio-Temporal Reasoning with inequalities, Artificial Intelligence Laboratory, AIM-875, MIT, Cambridge.

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

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