Temporal constraint networks*

Rina Dechter**

Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel

Itay Meiri and Judea Pearl

Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, CA 90024, USA

Abstract

Dechter, R., I. Meiri and J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1991) 61–95.

This paper extends network-based methods of constraint satisfaction to include continuous variables, thus providing a framework for processing temporal constraints. In this framework, called temporal constraint satisfaction problem (TCSP), variables represent time points and temporal information is represented by a set of unary and binary constraints, each specifying a set of permitted intervals. The unique feature of this framework lies in permitting the processing of metric information, namely, assessments of time differences between events. We present algorithms for performing the following reasoning tasks: finding all feasible times that a given event can occur, finding all possible relationships between two given events, and generating one or more scenarios consistent with the information provided.

We distinguish between simple temporal problems (STPs) and general temporal problems, the former admitting at most one interval constraint on any pair of time points. We show that the STP, which subsumes the major part of Vilain and Kautz's point algebra, can be solved in polynomial time. For general TCSPs, we present a decomposition scheme that performs the three reasoning tasks considered, and introduce a variety of techniques for improving its efficiency. We also study the applicability of path consistency algorithms as preprocessing of temporal problems, demonstrate their termination and bound their complexities.

**Present address: Information and Computer Science Department, University of California, Irvine, CA, USA.

1. Introduction

Problems involving temporal constraints arise in various areas of computer science such as scheduling, program verification, and parallel computation. Research in common-sense reasoning [22, 41], natural language understanding [3, 24], and planning [34], has identified new types of temporal reasoning problems, specific to AI applications. Several formalisms for expressing and reasoning about temporal knowledge have been proposed, most notably, Allen's interval algebra [2], Vilain and Kautz's point algebra [49], linear inequalities (Malik and Binford [33], Valdés-Pérez [46]), and Dean and McDermott's time map [13]. Each of these representation schemes is supported by a specialized constraint-directed reasoning algorithm. At the same time, extensive research has been carried out over the past years on problems involving general constraints (Montanari [36], Mackworth [31], Gaschnig [21], Freuder [19, 20], Haralick and Elliott [23], Nudel [37], Dechter and Pearl [16]), yet much of this work has not been extended to problems involving temporal constraints.

This paper presents a unified approach to temporal reasoning based on constraint-network formalism. Using this formalism, we were able to develop:

(1) a formal basis for relating various algorithmic schemes, permitting the analysis of their complexity and range of applicability;
(2) an economical representation, called a *minimal network*, which encodes all temporal relations between a pair of event points, including absolute bounds on their time difference;
(3) an efficient scheme of generating specific temporal scenarios, consistent with the given constraints.

We envision a temporal reasoning system to consist of a temporal knowledge base, a routine to check its consistency, a query answering mechanism and an inference mechanism capable of discovering new information. The primitive entities in the knowledge base are *propositions* with which we normally associate temporal intervals, e.g., "I was driving a car" or "the book was lying on the table"; each interval representing the time period during which the corresponding proposition holds. The temporal information might be relative (e.g., "P_1 occurred before P_2"), or metric (e.g., "P_1 had started at least 3 hours before P_2 was terminated"). To express less specific information, disjunctive sentences may also be needed (e.g., "you can come in before or after lunch hour"); a subclass of such sentences will be addressed in this paper. We also allow references to absolute time (such as 4:00 p.m.), and to the duration of propositions (e.g., "P lasted at least two hours"). Given temporal information of this kind, we want to derive answers to queries such as: is it possible that proposition P holds at time t? what are the possible times at

which proposition P holds? what are the possible temporal relationships between two propositions P_1 and P_2?

There have been several suggestions of how to represent temporal information. If propositions stand for events, and each proposition P_i is associated with an interval $I_i = [a_i, b_i]$, then information about the timing of events can be expressed by means of constraints on the intervals or their associated beginning and ending points. Allen [2] defined temporal knowledge to consist of constraints on the 13 possible relationships that can exist between any pair of intervals. Since finding all the feasible relationships between a given pair of intervals is intractable, Vilain and Kautz [49] suggested that the information be expressed by means of constraints on the beginning and ending point of each interval. This approach gives rise to a polynomial time algorithm, but can handle only a limited class of problems. Recently, Ladkin and Maddux [27] have proposed an algebraic approach to problems similar to those posed by Allen and Vilain and Kautz.

One of the requirements of our system is the ability to deal with metric information. Since both Allen's interval algebra and Vilain and Kautz's point algebra do not offer a convenient mechanism for dealing with such information, we take a different approach. We consider time points as the variables we wish to constrain—where a time point may be a beginning or an ending point of some event, as well as a neutral point of time such as 4:00 p.m. Malik and Binford [33] and Valdés-Pérez [46] suggested constraining the *temporal distance* between time points. Namely, if X_i and X_j are two time points, a constraint on their temporal distance would be of the form $X_j - X_i \leq c$, which gives rise to a set of linear inequalities on the X_i's. Such constraints, however, are insufficient; we must allow disjunctive sentences. Consider the following example:

Example 1.1. John goes to work either by car (30–40 minutes), or by bus (at least 60 minutes). Fred goes to work either by car (20–30 minutes), or in a carpool (40–50 minutes). Today John left home between 7:10 and 7:20, and Fred arrived at work between 8:00 and 8:10. We also know that John arrived at work about 10–20 minutes after Fred left home. We wish to answer queries such as: "Is the information in the story consistent?", "Is it possible that John took the bus, and Fred used the carpool?", "What are the possible times at which Fred left home?", and so on.

Let P_1 be the proposition "John was going to work", and P_2 the proposition "Fred was going to work". P_1 and P_2 are associated with intervals $[X_1, X_2]$ and $[X_3, X_4]$, respectively, where X_1 represents the time John left home while X_4 represents the time Fred arrived at work. Several temporal constraints are given in the story. From the fact that it takes John either 30–40 minutes or more than 60 minutes to get to work, the temporal distance between X_1 and X_2 is constrained by

$$30 \leqslant X_2 - X_1 \leqslant 40 \quad \text{or} \quad X_2 - X_1 \geqslant 60 \, . \tag{1.1}$$

Similar constraints apply to $X_4 - X_3$ and $X_2 - X_3$. Choosing $X_0 = 7{:}00$ a.m., the fact that John left home between 7:10 and 7:20 imposes the constraint

$$10 \leqslant X_1 - X_0 \leqslant 20 \, . \tag{1.2}$$

The constraint on $X_4 - X_0$ assumes a similar form.

This paper introduces a framework based on constraint-network formalism for representing and processing such problems. Within this framework several solution methods are established. Section 2 presents the temporal constraint satisfaction problem (TCSP). Section 3 deals with a restricted, simpler TCSP (called STP), solvable in polynomial time. Sections 4–6 offer some techniques for solving the general TCSP: decomposition into several STPs, approximation schemes, and network-based approaches. Section 7 relates the TCSP model to other temporal reasoning models, while Section 8 provides a summary and concluding remarks.

2. The TCSP model

The definitions needed for describing a temporal constraint satisfaction problem follow closely those developed for the general CSP [36]. A *temporal constraint satisfaction problem* (*TCSP*) involves a set of variables, $X_1, \ldots, X_n$, having continuous domains; each variable represents a time point. Each constraint is represented by a set of intervals[1]:

$$\{I_1, \ldots, I_n\} = \{[a_1, b_1], \ldots, [a_n, b_n]\} \, . \tag{2.1}$$

A unary constraint, T_i, restricts the domain of variable X_i to the given set of intervals; namely, it represents the disjunction

$$(a_1 \leqslant X_i \leqslant b_1) \vee \cdots \vee (a_n \leqslant X_i \leqslant b_n) \, . \tag{2.2}$$

A binary constraint, T_{ij}, constrains the permissible values for the distance $X_j - X_i$; it represents the disjunction

$$(a_1 \leqslant X_j - X_i \leqslant b_1) \vee \cdots \vee (a_n \leqslant X_j - X_i \leqslant b_n) \, . \tag{2.3}$$

We assume that constraints are always given in a *canonical form* where all intervals are pairwise disjoint.

A *network of binary constraints* (a *binary TCSP*) consists of a set of variables, $X_1, \ldots, X_n$, and a set of unary and binary constraints. Such a

---

[1] For simplicity we assume closed intervals; however, the same treatment applies to open and semi-open intervals.

network can 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; it is labeled by the interval set. Each input constraint, T_{ij}, implies an equivalent constraint T_{ji}; however, only one of them will usually be shown in the constraint graph. A special time point, X_0, is introduced to represent the "beginning of the world". 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 assume $X_0 = 0$. The constraint graph of Example 1.1 is given in Fig. 1.

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. A value v is a *feasible value* for variable X_i, if there exists a solution in which $X_i = v$. The set of all feasible values of a variable is called the *minimal domain*. The network is *consistent* if at least one solution exists.

We define the following binary operations on constraints: union, intersection and composition, respecting their usual set-theoretic definitions.

Definition 2.1. Let $T = \{I_1, \ldots, I_l\}$ and $S = \{J_1, \ldots, J_m\}$ be constraints, i.e., sets of intervals of a real variable t (t corresponds to $X_j - X_i$ in case of binary constraints).

(1) The *union* of T and S, denoted by $T \cup S$, admits only values that are allowed by either one of them, namely,

$$T \cup S = \{I_1, \ldots, I_l, J_1, \ldots, J_m\} \, . \tag{2.4}$$

(2) The *intersection* of T and S, denoted by $T \oplus S$, admits only values that are allowed by both of them, namely,

$$T \oplus S = \{K_1, \ldots, K_n\} \, , \tag{2.5}$$

where $K_k = I_i \cap J_j$ for some i and j. Note that $n \leq l + m$.

(3) The *composition* of T and S, denoted by $T \otimes S$, admits only values r for which there exist $t \in T$ and $s \in S$, such that $t + s = r$, namely,

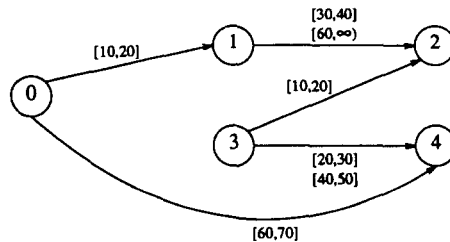$$T \otimes S = \{K_1, \ldots, K_n\} \, , \tag{2.6}$$

Fig. 1. A constraint graph representing Example 1.1.

where $K_k = [a + c, b + d]$ for some $I_i = [a, b]$ and $J_j = [c, d]$. Note that $n \leq l \times m$.

A pictorial illustration of the intersection and composition operations is given in Fig. 2. Note that for some of these operations the resulting interval representation is not in canonical form. For instance, the composition operation results in four intervals; however, due to overlap, only three of them appear in the canonical form. These three operations parallel the usual operations of union, intersection and composition in general constraint networks [36]. In particular, when T and S represent binary constraints on the differences $X_j - X_i$ and $X_k - X_j$, respectively, $T \otimes S$ admits only pairs of values (x_i, x_k) for which there exists a value x_j, such that (x_i, x_j) is permitted by T and (x_j, x_k) is permitted by S.

These operations are extended to operations on networks in the usual way. Given networks T and S, on the same set of variables, we define

$$(T \cup S)_{ij} = T_{ij} \cup S_{ij}$$ (2.7)

$$(T \oplus S)_{ij} = T_{ij} \oplus S_{ij} ,$$ (2.8)

where i and j range over all pairs of variables.

A partial order among constraints can be defined as follows. A binary constraint T is *tighter* than S, denoted by $T \subseteq S$, if every pair of values allowed by T is also allowed by S; namely, for every interval $I \in T$ there exists an
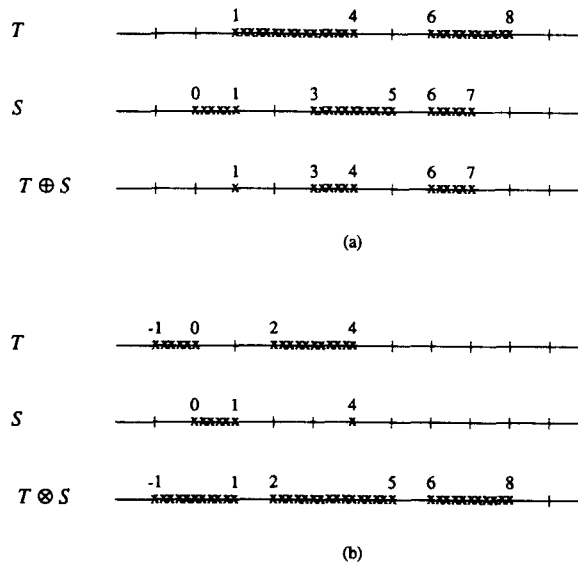

Fig. 2. Operations on constraints: (a) intersection, (b) composition.

interval $J \in S$ such that $I \subseteq J$. The tightest constraint is the *empty constraint*, $\emptyset$ (if the network contains an empty constraint, then it is trivially inconsistent). The most relaxed constraint is the *universal constraint*, $(-\infty, \infty)$. Edges corresponding to universal constraints are usually omitted from the constraint graph.

A partial order among binary constraint networks having the same set of variables can be defined as follows. A network T is tighter than network S, denoted $T \subseteq S$, if the partial order $\subseteq$ is satisfied for all the corresponding constraints; namely, for all pairs i, j, $T_{ij} \subseteq S_{ij}$. Two networks are *equivalent* if they represent the same solution set. A network may have many equivalent representations; in particular, there is one equivalent network which is minimal with respect to $\subseteq$, called the *minimal network* (note that the minimal network is unique because equivalent networks are closed under intersection). The arc constraints specified by the minimal network are called the *minimal constraints*.

A network is *decomposable*[2] [36], if every locally consistent assignment[3] to any set of variables, S, can be extended to a solution. The importance of decomposability lies in facilitating the construction of a solution by a *backtrack-free search* [20].

Given a constraint network, the first interesting problem is to determine its consistency. If the network is consistent we may wish to find some specific solutions, each representing a possible scenario, or to answer queries concerning the set of all solutions. The interesting queries are:

(1) "What are the possible times at which X_i could occur?" (asking for the minimal domain of X_i).
(2) "What are all the possible relationships between X_i and X_j?" (asking for the minimal constraint between X_i and X_j).

Computing the full minimal network would provide answers to all such queries. The rest of the paper presents several techniques for solving these tasks.

3. The simple temporal problem (STP)

A TCSP in which all constraints specify a single interval is called a *simple temporal problem* (*STP*). In such a network, each edge, $i \rightarrow j$, is labeled by an interval, $[a_{ij}, b_{ij}]$, which represents the constraint

$$a_{ij} \leqslant X_j - X_i \leqslant b_{ij} .$$ (3.1)

Alternatively, the constraint can be expressed as a pair of inequalities:

---

[2] In [36] decomposability is defined for minimal networks only.
[3] An assignment of values to a set of variables, S, is *locally consistent*, if it satisfies the constraints applicable to S; namely, those involving only variables in S (including the unary constraints).

$$X_j - X_i \leq b_{ij} \, , \tag{3.2}$$

$$X_i - X_j \leq -a_{ij} \, . \tag{3.3}$$

Thus, solving an STP amounts to solving a set of linear inequalities on the X_i's.

The problem of solving a system of linear inequalities is well known in the operations research literature. It can be solved by the (exponential) simplex method [10] or Khachiyan's algorithm [25], which is rather complicated in practice. Fortunately, the special class of linear inequalities characterizing the STP admits a simpler solution; the inequalities are given a convenient graph representation, to which a shortest paths algorithm can be applied [6, 29, 30, 42]. In the AI literature, a similar data structure, called a *time map*, was introduced by Dean and McDermott [13] to facilitate planning, but was not formulated mathematically.

Formally, we associate an STP with a directed edge-weighted graph, $G_d = (V, E_d)$, called a *distance graph* (to be distinguished from the constraint graph). It has the same node set as G, and each edge, $i \rightarrow j$, is labeled by a weight a_{ij}, representing the linear inequality $X_j - X_i \leq a_{ij}$. In Example 1.1, if we assume that John used a car and Fred used a carpool, we get an STP having

$$T_{12} = \{[30, 40]\} \quad \text{and} \quad T_{34} = \{[40, 50]\} \, , \tag{3.4}$$

and a distance graph as depicted in Fig. 3.

Each path from i to j in G_d, $i_0 = i$, $i_1, \ldots, i_k = j$, induces the following constraint on the distance $X_j - X_i$:

$$X_j - X_i \leq \sum_{j=1}^{k} a_{i_{j-1}, i_j} \, . \tag{3.5}$$

If there is more than one path from i to j, then it can be easily verified that the intersection of all the induced path constraints yields

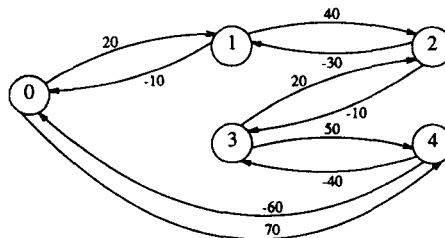$$X_j - X_i \leq d_{ij} \, , \tag{3.6}$$

Fig. 3. A distance graph representing a portion of Example 1.1.

where d_{ij} is the length of the shortest path from i to j. Based on this observation, the following condition for consistency of an STP can be established:

Theorem 3.1 (Shostak [42], Liao and Wong [30], Leiserson and Saxe [29]). *A given STP, T, is consistent if and only if its distance graph, G_d, has no negative cycles.*

Proof. Suppose there is a negative cycle, C, consisting of nodes $i_1, \ldots, i_k = i_1$. Summing the inequalities along C yields $X_{i_1} - X_{i_1} < 0$, which cannot be satisfied. Conversely, if there is no negative cycle in G_d, then the shortest path between each pair of nodes is well-defined. For any pair of nodes, i and j, the shortest paths satisfy $d_{0j} \leq d_{0i} + a_{ij}$; thus

$$d_{0j} - d_{0i} \leq a_{ij} . \tag{3.7}$$

Hence, the tuple $(d_{01}, \ldots, d_{0n})$ is a solution of the given STP. $\square$

Corollary 3.2. *Let G_d be the distance graph of a consistent STP. Two consistent scenarios are given by:*

$$S_1 = (d_{01}, \ldots, d_{0n}) , \tag{3.8}$$

$$S_2 = (-d_{10}, \ldots, -d_{n0}) , \tag{3.9}$$

which assign to each variable its latest and earliest possible time, respectively.

Proof. The proof of Theorem 3.1 shows that S_1 is a solution. To show that S_2 is a solution, note that for all i and j, $d_{i0} \leq a_{ij} + d_{j0}$, or

$$(-d_{j0}) - (-d_{i0}) \leq a_{ij} , \tag{3.10}$$

yielding S_2 as a solution. $\square$

From the above discussion it follows that a given STP can be effectively specified by a complete directed graph, called *d-graph*, where each edge, $i \to j$, is labeled by the shortest path length, d_{ij}, in G_d; it corresponds to a more explicit representation of our STP (see (3.5) and (3.6)).

Theorem 3.3 (Decomposability). *Any consistent STP is decomposable relative to the constraints in its d-graph.*

Proof. It suffices to show that any instantiation of a subset S of k variables

$(1 \leq k < n)$ that satisfies all the shortest path constraints applicable to S, is extensible to any other variable. This will be shown by induction on $|S| = k$.

For $k = 1$, S consists of a single variable, X_i, instantiated to x_i. We will show that for any other variable, X_j, we can find an assignment $X_j = v$ which satisfies the shortest path constraints between them. The value v must satisfy

$$-d_{ji} \leq v - x_i \leq d_{ij} .$$
$$(3.11)$$

Since all cycles in the distance graph are nonnegative, we have $d_{ji} + d_{ij} \geq 0$ and, hence, there exists a value v satisfying (3.11).

Assume that the theorem holds for $|S| = k - 1$; we must show that it holds for $|S| = k$. Without loss of generality, let $S = \{X_1, \ldots, X_k\}$, and let $\{X_i = x_i \mid 1 \leq i \leq k\}$ be an assignment that satisfies the shortest path constraints among the variables in S. Let $X_{k+1} \not\in S$. We need to find a value $X_{k+1} = v$ which satisfies the shortest path constraints between X_{k+1} and all variables in S. In other words, v must satisfy

$$v - x_i \leq d_{i,k+1} ,$$
$$(3.12)$$

$$x_i - v \leq d_{k+1,i} ,$$
$$(3.13)$$

for $i = 1, \ldots, k$, or

$$v \leq \min\{x_i + d_{i,k+1} \mid 1 \leq i \leq k\} ,$$
$$(3.14)$$

$$v \geq \max\{x_i - d_{k+1,i} \mid 1 \leq i \leq k\} .$$
$$(3.15)$$

Suppose the minimum is attained at i_0, and the maximum at j_0. Thus, v must satisfy

$$x_{j_0} - d_{k+1,j_0} \leq v \leq x_{i_0} + d_{i_0,k+1} .$$
$$(3.16)$$

Since x_{i_0} and x_{j_0} satisfy the constraint between them, we have

$$x_{j_0} - x_{i_0} \leq d_{i_0,j_0} .$$
$$(3.17)$$

This, together with $d_{i_0,j_0} \leq d_{i_0,k+1} + d_{k+1,j_0}$, yields

$$x_{j_0} - d_{k+1,j_0} \leq x_{i_0} + d_{i_0,k+1} .$$
$$(3.18)$$

Therefore, there exists a value v which satisfies the condition of (3.16). $\square$

The importance of Theorem 3.3 lies in providing an efficient algorithm for assembling a solution to a given STP; we simply assign to each variable any

value that satisfies the *d*-graph constraints relative to previous assignments (starting with $X_0 = 0$). Decomposability guarantees that such a value can always be found, regardless of the order of assignment. A second by-product of decomposability is that the domains characterized by the *d*-graph are minimal.

Corollary 3.4. *Let* G_d *be the distance graph of a consistent STP. The set of feasible values for variable* X_i *is* $[-d_{i0}, d_{0i}]$.

Proof. According to Theorem 3.3, the assignment $X_0 = 0$ can be extended by assigning any value v satisfying $v \in [-d_{i0}, d_{0i}]$ to X_i. This assignment, in turn, can be extended to a full solution. Thus, v is a feasible value. $\square$

We have noted that the *d*-graph represents a tighter, yet equivalent network of the original STP. From Theorem 3.3 we can now conclude that this new network is the minimal network.

Corollary 3.5. *Given a consistent STP, T, the equivalent STP, M, defined by*

$$\forall i, j, \quad M_{ij} = \{[-d_{ji}, d_{ij}]\}, \tag{3.19}$$

is the minimal network representation of T.

Proof. See Appendix A. $\square$

Illustration. Consider the distance graph of Fig. 3. Since there are no negative cycles, the corresponding STP is consistent. The shortest path distances, d_{ij}, are shown in Table 1. The minimal domains are $10 \leq X_1 \leq 20$, $40 \leq X_2 \leq 50$, $20 \leq X_3 \leq 30$ and $60 \leq X_4 \leq 70$. In particular, one special solution is the tuple $(d_{01}, \ldots, d_{04})$, namely, the assignment

$$\{X_1 = 20, X_2 = 50, X_3 = 30, X_4 = 70\}, \tag{3.20}$$

which selects for each variable its latest possible time. According to this

Table 1
Lengths of shortest paths in the distance graph of Fig. 3.

	0	1	2	3	4
0	0	20	50	30	70
1	−10	0	40	20	60
2	−40	−30	0	−10	30
3	−20	−10	20	0	50
4	−60	−50	−20	−40	0

Table 2
The minimal network corresponding to Fig. 3.

	0	1	2	3	4
0	[0]	[10, 20]	[40, 50]	[20, 30]	[60, 70]
1	[−20, −10]	[0]	[30, 40]	[10, 20]	[50, 60]
2	[−50, −40]	[−40, −30]	[0]	[−20, −10]	[20, 30]
3	[−30, −20]	[−20, −10]	[10, 20]	[0]	[40, 50]
4	[−70, −60]	[−60, −50]	[−30, −20]	[−50, −40]	[0]

solution, John left home at 7:10 and arrived at work at 7:50, while Fred left home at 7:30 and arrived at work at 8:10. The minimal network is given in Table 2. Notice that the minimal network is symmetric in the sense that if $T_{ij} = \{[a, b]\}$, then $T_{ji} = \{[-b, -a]\}$. An alternative scenario, in which John used a bus and Fred used a carpool (i.e., $T_{12} = \{[60, \infty)\}$ and $T_{34} = \{[40, 50]\}$), results in a negative cycle and is therefore inconsistent.

The d-graph of an STP can be constructed by applying *Floyd–Warshall*'s all-pairs-shortest-paths algorithm [38] to the distance graph (see Fig. 4). The algorithm runs in time $O(n^3)$, and detects negative cycles simply be examining the sign of the diagonal elements d_{ii}. It constitutes, therefore, a polynomial time algorithm for determining the consistency of an STP, and for computing both the minimal domains and the minimal network. Once the d-graph is available, assembling a solution requires only $O(n^2)$ time, because each successive assignment needs to be checked against previous assignments and is guaranteed to remain unaltered. Thus, finding a solution can be achieved in $O(n^3)$ time.

All-pairs-shortest-paths algorithm

1. for $i := 1$ to n do $d_{ii} \leftarrow 0$;
2. for $i, j := 1$ to n do $d_{ij} \leftarrow a_{ij}$;
3. for $k := 1$ to n do
4.   for $i, j := 1$ to n do
5. $d_{ij} \leftarrow \min\{d_{ij}, d_{ik} + d_{kj}\}$;

Fig. 4. Floyd–Warshall's algorithm.

4. The general TCSP

Having solved the STP, we now return to the general problem in which edges may be labeled by several intervals. Davis [12] showed that determining consistency for a general TCSP is NP-hard.

Theorem 4.1 (Davis [12]).
  (i) *Deciding consistency for a TCSP is NP-hard.*
  (ii) *Deciding consistency for a TCSP with no more than two intervals per edge is NP-hard.*

Proof. (i) Reduction from 3-coloring. Let $G = (V, E)$ be a graph to be colored. We construct a TCSP, T, in the following way. For each node, V_i, we introduce a variable, X_i, and a unary constraint on X_i,

$$X_i \in \{[1], [2], [3]\} \,, \tag{4.1}$$

where [1], [2] and [3] stand for the three admissible colors. With each edge $(i, j) \in E$ we associate a binary constraint

$$X_j - X_i \in \{[-2], [-1], [1], [2]\} \,. \tag{4.2}$$

Equation (4.2) restricts X_i and X_j to have different colors. Hence, T is consistent if and only if G is 3-colorable.

(ii) Again, reduction from 3-coloring. We construct a TCSP, T, as follows. For each node, V_i, we introduce two variables, X_i' and X_i'', having domains

$$X_i' \in \{[1], [2, 3]\} \,, \tag{4.3}$$

$$X_i'' \in \{[1, 2], [3]\} \,, \tag{4.4}$$

and restrict X_i' and X_i'' to be equal:

$$X_i' = X_i'' \,. \tag{4.5}$$

This forces X_i' and X_i'' to assume integer values as in (4.1). To restrict the colors of nodes V_i and V_j to be different, the following binary constraints are introduced:

$$X_j' - X_i' \in \{[-2], [-1, 2]\} \,, \tag{4.6}$$

$$X_j'' - X_i' \in \{[-2, -1], [1, 2]\} \,, \tag{4.7}$$

$$X_j' - X_i'' \in \{[-2, 1], [2]\} \,. \tag{4.8}$$

T is consistent if and only if the graph is 3-colorable. □

A straightforward way of solving the general TCSP is to decompose it into several STPs, solve each one of them, and then combine the results. Given a

binary TCSP, T, we define a *labeling* of T as a selection of one interval from each constraint. Each labeling defines an STP graph whose edges are labeled by the selected intervals. We can solve any of the TCSP tasks by considering all its STPs. Specifically, the original network is consistent iff there is a labeling whose associated STP is consistent. Any solution of T is also a solution of one of its STPs and vice versa. Also, the minimal network of T can be computed from the minimal networks associated with its individual STPs, as stated in the following theorem:

Theorem 4.2. *The minimal network, M, of a given TCSP, T, satisfies $M = \bigcup_l M_l$, where M_l is the minimal network of the STP defined by labeling l, and the union is over all the possible labelings.*

Proof. We first note that the solution set of T is identical to the union of the solution sets of its labelings. Hence, $\bigcup M_l$ is equivalent to T. M is by definition the tightest of all networks equivalent to T, and therefore $M \subseteq \bigcup M_l$. Now suppose that M is strictly tighter than $\bigcup M_l$. Then, there exist a pair of variables, i and j, a labeling, s, and a value, d, such that $d \in (M_s)_{ij}$ but $d \notin M_{ij}$. Let x and y be values of the variables i and j, respectively, such that $y - x = d$. According to the minimality of M_s, this partial assignment can be extended to a solution of s, which is also a solution of T; hence $d \in M_{ij}$, yielding a contradiction. Therefore, $\bigcup M_l \subseteq M$. $\square$

Illustration. The minimal network of Example 1.1 is shown in Table 3. In this case, only 3 of the 4 possible labelings contribute to the minimal network.

The complexity of solving a general TCSP by generating all the labelings and solving them independently is $O(n^3 k^e)$, where k is the maximum number of intervals labeling an edge, and e is the number of edges.

Table 3
The minimal network of Example 1.1.

	0	1	2	3	4
0	[0]	[10, 20]	[40, 60] [70]	[20, 50]	[60, 70]
1	[−20, −10]	[0]	[30, 40] [60]	[10, 30] [40]	[40, 60]
2	[−70] [−60, −40]	[−60] [−40, −30]	[0]	[−20, −10]	[0, 30]
3	[−50, −20]	[−40] [−30, −10]	[10, 20]	[0]	[20, 30] [40, 50]
4	[−70, −60]	[−60, −40]	[−30, 0]	[−50, −40] [−30, −20]	[0]

This brute-force enumeration process can be pruned significantly by running a backtracking search on a meta-CSP whose variables are the TCSP's edges, and the domains are the possible intervals. Backtrack assigns an interval to an edge, as long as the condition of Theorem 3.1 is satisfied and, if no such assignment is possible, it backtracks.

Formally, let T be a given TCSP, and let $G = (V, E)$ be its associated constraint graph. Let $CSP(T)$ be a discrete CSP with variables $X_1, \ldots, X_m$, where $m = |E|$, and variable X_i corresponds to edge $e_i \in E$. The domain of X_i consists of the intervals $I_1, \ldots, I_k$ that label e_i in G. The constraints are not given explicitly (as a list of allowed or disallowed combinations), instead, any assignment, $\{X_{i_1} = I_{i_1}, \ldots, X_{i_s} = I_{i_s}\}$, is consistent, if and only if the corresponding STP is consistent. Clearly, each solution of $CSP(T)$ corresponds to a consistent labeling of G and, thus, any algorithm that finds all the solutions of $CSP(T)$ can be used to solve T. A backtrack algorithm that computes the minimal network of a TCSP is shown in Fig. 5. It is defined by two recursive procedures: **Forward** and **Go-back**. The first extends a current partial assignment if possible, and the second handles dead-end situations. The procedures maintain a list of candidate intervals, C_i, for each variable X_i.

Forward$(I_1, \ldots, I_i)$
1. if $i = m$ then
2. $M \leftarrow M \cup$ Solve-STP$(I_1, \ldots, I_m)$, and
3.   Go-Back$(I_1, \ldots, I_m)$;
4. $C_{i+1} \leftarrow \emptyset$;
5. for every I_j in D_{i+1} do
6.   if Consistent-STP$(I_1, \ldots, I_i, I_j)$ then
7. $C_{i+1} \leftarrow C_{i+1} \cup \{I_j\}$;
8. If $C_{i+1} \neq \emptyset$ then
9. $I_{i+1} \leftarrow$ first element in C_{i+1}, and
10.   remove I_{i+1} from C_{i+1}, and
11.   Forward$(I_1, \ldots, I_i, I_{i+1})$
12. else
13.   Go-Back$(I_1, \ldots, I_i)$;

Go-back$(I_1, \ldots, I_i)$
1. if $i = 0$ then exit
2. if $C_i \neq \emptyset$ then
3. $I_i \leftarrow$ first element in C_i, and
4.   remove I_i from C_i, and
5.   Forward$(I_1, \ldots, I_i)$
6. else
7.   Go-back$(I_1, \ldots, I_{i-1})$;

Fig. 5. A backtrack algorithm.

Backtrack is initiated by calling "Forward" with $i = 0$, namely, the instantiated list is empty. The procedure "Solve-STP$(I_1, \ldots, I_m)$" returns the minimal network of the STP defined by $\{I_1, \ldots, I_m\}$. The procedure "Consistent-STP$(I_1, \ldots, I_i, I_j)$" determines if the partial STP defined by $\{I_1, \ldots, I_i, I_j\}$ is consistent; it can be done either by using an all-pairs-shortest-paths algorithm, or by an improved algorithm to be described in Section 5. At the beginning of the algorithm, $M = \emptyset$ and, upon termination, M contains the minimal network (if $M = \emptyset$, then the network is inconsistent). If our task is to find a single solution, then once we find a consistent labeling we may construct a solution using the technique described in the previous section.

Although the worst-case complexity of this approach is also $O(n^3 k^e)$, it enables us to utilize enhancement techniques which, in practice, prove to substantially reduce the complexity of backtrack below its worst-case value. Such techniques include backjumping [21], variable ordering [15, 20, 40], value ordering [16, 23] and learning schemes [14]. Moreover, with some investment of storage space, the work done on any partial instantiation can be utilized toward its extension (without redoing the problem afresh), and this reduces the time complexity to $O(n^2 k^e)$.

In the following sections we will present alternative approaches for solving the general TCSP. In particular, Section 5 discusses path consistency algorithms that can be used either as an approximation, or as a preprocessing step before applying backtracking. Section 6 shows how the topology of the constraint graph can be exploited to yield more efficient algorithms.

5. Path consistency algorithms

Imposing local consistency among subsets of variables may serve as a preprocessing step to improve backtrack. Local consistency algorithms, especially path consistency, might also serve as a good approximation scheme which often yields the minimal network. In this section we study the applicability of path consistency and its weaker version, directional path consistency, in the TCSP framework.

Floyd–Warshall's algorithm, used for solving the STP, can be considered a *relaxation* algorithm—in every step of the process the label of an edge is updated by an amount that depends only on the current labels of adjacent edges. In fact, there is a rich family of similar algorithms [1, 7, 28, 39, 43, 44], all based on the same principle. Montanari [36] was the first to use such an algorithm, called *path consistency*, in the context of constraint satisfaction problems. This was further explored and analyzed by Mackworth [31], and Mackworth and Freuder [32].

Pursuing its traditional role [31, 36], path consistency in the context of a TCSP is defined as follows:

Definition 5.1. A path through nodes $i_0, i_1, \ldots, i_m$ is *path consistent* iff for any pair of values, v_0 and v_m, such that $v_m - v_0 \in T_{i_0 i_m}$, there exists a sequence of values, $v_1, \ldots, v_{m-1}$, such that $v_1 - v_0 \in T_{i_0 i_1}$, $v_2 - v_1 \in T_{i_1 i_2}, \ldots$, and $v_m - v_{m-1} \in T_{i_{m-1} i_m}$. A *network* is path consistent iff every path is consistent.

Using the operations $\oplus$ and $\otimes$ (denoting intersection and composition), Montanari's path consistency algorithm (equivalent to Mackworth's [31] PC-1) is shown in Fig. 6. The algorithm imposes local consistency among triplets of variables until a fixed point is reached, or until some constraint becomes empty indicating an inconsistent network. Clearly, the algorithm computes a network which is equivalent to the original one. For discrete-domain CSPs, Montanari showed that the algorithm terminates and that the resulting network is indeed path consistent. In our case, due to the continuous domains of TCSPs, one cannot guarantee that the algorithm terminates. It is clear, however, that running the algorithm indefinitely will result in a limit network. Each step of the algorithm yields a tighter network, and since the network is bounded below by the minimal network, a limit point is assured. Moreover, analysis shows that for all practical purposes PC-1 terminates in a finite number of steps. This will be shown in two parts; first for STPs, then for general TCSPs.

Comparing Figs. 4 and 6, PC-1 is seen to be a generalization of the all-pairs-shortest-paths algorithm. When applied to an STP, the relaxation step that updates T_{ij} amounts to two local operations of updating the shortest path length, d_{ij}, in Floyd–Warshall's algorithm. Therefore:

Theorem 5.2. *Applying PC-1 to an STP network is identical to applying Floyd–Warshall's algorithm to its distance graph.*

An immediate corollary of this theorem is that PC-1 terminates and produces a path consistent network. See also [11, 31, 36] for additional relationships between shortest paths algorithms and path consistency.

Regarding general TCSPs, two questions must be addressed; first, does PC-1 terminate and compute a path consistent network and, second, is the resulting

---

Algorithm PC-1
1. repeat
2. $S \leftarrow T$;
3.   for $k := 1$ to n do
4.   for $i, j := 1$ to n do
5. $T_{ij} \leftarrow T_{ij} \oplus T_{ik} \otimes T_{kj}$, and
6.   if $T_{ij} = \emptyset$ then exit (the network is inconsistent);
7. until $S = T$;

Fig. 6. A path consistency algorithm.

network minimal. We will next show that the answer to the first question is affirmative while the answer to the second is negative.

It is simple to show that PC-1 terminates for *integral* TCSPs, where the extreme points of all intervals are integers. This is so because each intersection operation at Step 5 must tighten a constraint by an integral amount. For nonintegral TCSPs, the same argument holds if the extreme points are rational numbers (these will be called *rational* TCSPs); we simply multiply all quantities by the greatest common divisor of the extreme points. This was shown more formally by Ladkin [26]. Thus, since all practical problems are expressible by rational numbers, PC-1 can be regarded as terminating. Once termination has been ascertained, the path consistency of the resulting network can be established by straightforward application of Montanari's proof [36]; the continuous nature of temporal domains plays no role. Summarizing, we have:

Theorem 5.3. *Algorithm PC-1 computes a path consistent network.*

Now that we have established that PC-1 terminates and computes a path consistent network, the question arises whether the resulting network is minimal. Montanari showed that when the constraints obey the distributivity property (i.e., that composition distributes over intersection), any path consistent network is both minimal and decomposable. Moreover, in such a case only one application of the main loop (Steps 1–7) is sufficient for reaching the fixed point. When constraints are defined by one interval (the STP case), the distributivity property holds and indeed, for this case, the path consistent network is minimal (Corollary 3.5), decomposable (Theorem 3.3), and requires only one iteration (see Floyd–Warshall's algorithm). Unfortunately, distributivity does not hold for the multi-interval TCSP, as can be seen in the following example:

Example 5.4. Consider the network shown in Fig. 7 where, for convenience, both directions of each edge are explicitly given. There are two paths from node 1 to node 3, representing the constraints $T_{13} = \{[25, 50]\}$ and $S_{13} = \{[0, 30], [40, 50]\}$ (the latter is obtained by composing T_{12} with T_{23}). Perform-
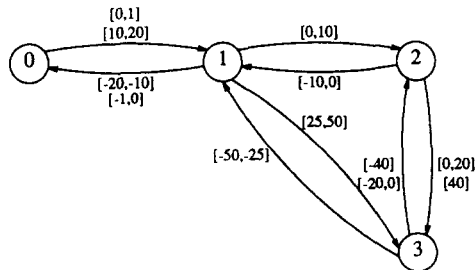

Fig. 7. A nondistributive network.

ing intersection first, and then composition, we get

$$T_{01} \otimes (T_{13} \oplus S_{13}) = \{[0, 1], [10, 20]\} \otimes \{[25, 30], [40, 50]\}$$

$$= \{[25, 31], [35, 70]\} . \tag{5.1}$$

Performing composition first, and then intersection, results in

$$(T_{01} \otimes T_{13}) \oplus (T_{01} \otimes S_{13}) = \{[0, 31], [40, 51], [10, 50], [50, 70]\}$$
$$\oplus \{[25, 51], [35, 70]\}$$

$$= \{[25, 70]\} . \tag{5.2}$$

Clearly, distributivity does not hold. Indeed, if we apply path consistency to this network then after one iteration we have $T_{03} = \{[25, 70]\}$, whereas in the minimal network (shown in Table 4), $M_{03} = \{[25, 31], [35, 70]\}$. Interestingly, another application of the main loop does result in a fixed point which is also the minimal network (see Section 6).

In general CSPs it is well known that path consistency may not converge to the minimal network. The next example (tailored after Montanari [36]) will demonstrate that this phenomenon persists also in temporal problems; path consistency does not even detect inconsistency.

Example 5.5. Consider the 3-coloring problem on K_4, the complete graph of four nodes. The problem is obviously inconsistent and at the same time path consistent—every set of three nodes can be 3-colored. Translating this problem into TCSP notation, as in the proof of Theorem 4.1, yields the desired example. The problem consists of four variables, $X_1, \ldots, X_4$, each having a domain $\{[1], [2], [3]\}$, connected by six binary constraints

$$X_j - X_i \in \{[-2], [-1], [1], [2]\} , \tag{5.3}$$

Table 4
The minimal network of Example 5.4.

	0	1	2	3
0	[0]	[0, 1] [10, 20]	[0, 30]	[25, 31] [35, 70]
1	[−20, −10] [−1, 0]	[0]	[0, 10]	[25, 30] [40, 50]
2	[−30, 0]	[−10, 0]	[0]	[15, 20] [40]
3	[−70, −35] [−31, −25]	[−50, −40] [−30, −25]	[−40] [−20, −15]	[0]

for $i, j = 1, \ldots, 4$, $i \neq j$. The resulting network is already path consistent, yet PC-1 will fail to detect its inconsistency.

A more efficient path consistency algorithm is the temporal equivalent of Mackworth's PC-2 [31], shown in Fig. 8. The function REVISE$((i, k, j))$ updates T_{ij} by considering the length-2 path from i to j through k,

$$T_{ij} \leftarrow T_{ij} \oplus T_{ik} \otimes T_{kj} , \tag{5.4}$$

and returns true if T_{ij} has been modified. The function RELATED-PATHS$((i, k, j))$ returns the set of length-2 paths that need to be considered if T_{ij} is changed. The details of RELATED-PATHS are given in [31].

For discrete CSPs, path consistency can be achieved in time polynomial in n, the number of variables, and k, the maximum domain size [32]. We will now show that the temporal mirror of PC-2 achieves path consistency in $O(n^3 R^3)$, where R is the maximum *range* of any constraint (expressed in terms of the coarsest possible time units).

Definition 5.6. Let $T_{ij} = \{[a_1, b_1], \ldots, [a_n, b_n]\}$. The *range* of T_{ij} is $b_n - a_1$. The range of a TCSP is the maximum range over all constraints.

Theorem 5.7. *Temporal path consistency can be achieved in $O(n^3 R)$ relaxation steps and $O(n^3 R^3)$ arithmetic operations, where R is the range of the TCSP expressed in the coarsest possible time units.*

Proof. The worst-case running time of PC-2 occurs when every constraint interval is decreased by only one time unit each time it is tightened by REVISE. In this case, if R is the maximum constraint range, each constraint might be updated $O(R)$ times. Also, in the worst case, when a constraint is modified $O(n)$ paths are added to Q (see [31]). Thus, if we use the number of relaxation steps (calls to REVISE) as the complexity measure, then, since there are $O(n^2)$ constraints, the total complexity of PC-2 is $O(n^3 R)$. A more realistic measure would be the number of arithmetic operations. Each relaxation operation, $A \oplus B \otimes C$, where l, m and n are the number of intervals in A, B and C,

---

Algorithm PC-2

1. $Q \leftarrow \{(i, k, j) \mid (i < j) \text{ and } (k \neq i, j)\}$;
2. while Q is not empty do
3.   select and delete a path (i, k, j) from Q, and
4.   if REVISE$((i, k, j))$ then $Q \leftarrow Q \cup$ RELATED-PATHS$((i, k, j))$;

Fig. 8. PC-2—a more efficient path consistency algorithm.

respectively, involves $O(l + m \times n)$ arithmetic operations. Thus, since each relaxation step may involve as many as $O(R^2)$ operations, the total time is $O(n^3 R^3)$. □

For comparison to chronological backtracking, note that R must be at least as large as k (the number of intervals per constraint). However, $O(k^e)$ may reflect a lower complexity than $O(R^3)$, in case the edges are labeled by a few intervals.

Although path consistency algorithms are not guaranteed to compute the minimal network, they often provide a practical alternative and a complementary approach to the decomposition scheme. Moreover, they are readily amenable to parallel and distributed computation. In preliminary experiments on small random problems (each consisting of 5–7 variables), PC-1 always found the minimal network.[4] On the basis of these experiments, it appears that path consistency will substantially reduce the amount of work done by backtracking. To fully assess the benefits of the path consistency scheme, full-scale experimental studies should be undertaken.

Some problems may benefit from a weaker version of path consistency, called *directional path consistency* [16], that can be enforced more efficiently.

Definition 5.8 [16]. Let d be an ordering on the variables, and let X_i precede X_j in d iff $i < j$. A constraint graph, G, is *directional path consistent* with respect to d, if for every pair of values, v_i and v_j, such that $v_j - v_i \in T_{ij}$, and for every $k > i, j$, there exists a value v_k such that $v_k - v_i \in T_{ik}$ and $v_j - v_k \in T_{kj}$.

Given a TCSP, T, its associated constraint graph, $G = (V, E)$, and an ordering, d, directional path consistency can be achieved by algorithm DPC, shown in Fig. 9, which is the temporal counterpart of that given in [16].

DPC is similar to PC-1, but unlike the latter, it is a single pass algorithm. Note also that in Step 4, the set of edges E is increased dynamically by the relaxation operation of Step 3. The network defined by the final set of edges is called the *induced graph*.

Algorithm DPC

1. for $k := n$ down to 1 by -1 do
2.   for all $i, j < k$ such that $(i, k), (j, k) \in E$ do
3. $T_{ij} \leftarrow T_{ij} \oplus T_{ik} \otimes T_{kj}$, and
4. $E \leftarrow E \cup (i, j)$, and
5.   if $T_{ij} = \emptyset$ then exit (the network is inconsistent);

Fig. 9. DPC—an algorithm enforcing directional path consistency.

[4] Yaara Levi and Margalit Pinkas, personal communication.

If one of the constraints becomes empty (at Step 5), then the original network must have been inconsistent. However, as in the case of nontemporal CSPs, we are not guaranteed that the algorithm will always detect an inconsistency if one exists. Next we show that such a guarantee can be assured for STPs.

Definition 5.9. Let T be a TCSP. A cycle $i_0, \ldots, i_k = i_0$ is called *valid* if and only if

$$0 \in T_{i_0, i_1} \otimes \cdots \otimes T_{i_{k-1}, i_k} . \tag{5.5}$$

Lemma 5.10. *A given STP, T, is consistent if and only if all the cycles in its constraint graph are valid.*

Proof. See Appendix B. $\square$

Theorem 5.11. *Given an STP, T, algorithm DPC halts at Step 5 if and only if the network is inconsistent.*

Proof. The *only if* part is trivial; we will show the *if* part. Suppose the network is inconsistent; then, according to Lemma 5.10, there exists an invalid cycle C. Let the nodes of C be the set $\{i_1, \ldots, i_k\}$, and order it along d, namely, i_j will be processed after i_k whenever $j < k$. We next prove the following lemma.

Lemma 5.12. *For all j, $0 \leqslant j \leqslant k - 3$, when node i_{k-j} is about to be processed (Step 1), there exists an invalid cycle C_j, consisting of nodes $\{i_1, \ldots, i_{k-j}\}$.*

Proof. By induction on j. The lemma holds for $j = 0$ because the cycle $C_0 = C$ was assumed to be invalid in the original network, and DPC can only render constraints tighter. Thus, C_0 must remain invalid when node i_k is processed.

Assume the lemma holds for $j - 1$, $j > 0$. By the induction hypothesis, when node i_{k-j+1} was about to be processed, there was an invalid cycle C_{k-j+1} consisting of nodes $\{i_1, \ldots, i_{k-j+1}\}$. Let s and r be the neighbors of i_{k-j+1} in C_{k-j+1}, and let P_{rs} be the path from r to s in C_{k-j+1}. When node i_{k-j+1} is processed, the constraint T_{sr} is tightened, and the newly created cycle is

$$C_{k-j} = (s, r) \cup P_{rs} . \tag{5.6}$$

The constraint along C_{k-j} is tighter than the constraint along C_{k-j+1}, and thus C_{k-j} is invalid. Between the time that i_{k-j+1} is processed until the time i_{k-j} is processed, DPC further tightens the constraints along C_{k-j}. Thus, the cycle remains invalid while i_{k-j} is being processed. $\square$

According to Lemma 5.12, when node i_3 is about to be processed, there

exists an invalid cycle C_3, consisting of nodes i_1, i_2 and i_3. Let $T_{i_1,i_3} = \{[a, b]\}$, $T_{i_3,i_2} = \{[c, d]\}$ and $T_{i_2,i_1} = \{[e, f]\}$. At Step 3 the constraint T_{i_1,i_2} is updated such that

$$T_{i_1,i_2} = \{[\max(-f, a + c), \min(-e, b + d)]\} \,. \tag{5.7}$$

Since C_3 is invalid, $0 \notin [a + c + e, b + d + f]$. If $a + c + e > 0$, then $a + c > -e$, and $T_{i_1,i_2} = \emptyset$. Otherwise, $b + d + f < 0$, or $b + d < -f$, and thus $T_{i_1,i_2} = \emptyset$. Hence, at Step 5 the algorithm must halt. $\square$

It is well known that for general CSPs, directional path consistency can be achieved more efficiently than full path consistency [16]; instead of $O(n^3)$, DPC runs in $O(nW^*(d)^2)$ time, where $W^*(d)$ is the maximum number of parents that a node possesses in the induced graph. To assess the savings in the context of temporal problems, recall that each relaxation step involves $O(R^2)$ arithmetic operations, thus yielding a worst-case bound of $O(nW^*(d)^2R^2)$ operations. Another upper bound emerges from the fact that with every node processed the number of intervals recorded may increase by a factor not greater than k, thus giving a total of at most $O(k^n)$ intervals and arithmetic operations in any relaxation step. Hence, the upper bound is $O(nW^*(d)^2k^n)$.

For STPs, each relaxation step involves a constant number of arithmetic operations, and thus consistency for STPs can be determined in $O(nW^*(d)^2)$, in contrast with $O(n^3)$ needed for full path consistency. $W^*(d)$ could be substantially lower than n, and can be found in time $O(|V| + |E|)$ prior to the actual processing [5, 9, 45].

Note that directional path consistency is generally speaking weaker than full path consistency and, hence, might lead to a higher number of dead ends for backtrack. However, the use of directional path consistency yields more dramatic savings if it is embedded within backtracking as the consistency checking routine, "Consistent-STP" (Fig. 5). Instead of checking consistency by the $O(n^3)$ Floyd–Warshall algorithm, we can reduce the search effort of backtrack by a factor of roughly $(n/W^*(d))^2$ using DPC. In the next section we characterize a class of problems that gain fuller benefit from the efficiency of directional path consistency.

6. Network-based algorithms

So far we have presented techniques for processing networks of a general structure. The topology of the constraint graph did not play any role in the choice of the solution technique. However, considering the topological features of the constraint network may guide us, as they do in nontemporal CSPs, in

selecting efficient solution methods, having lower worst-case complexity than naive backtracking.

We first consider the task of finding a single solution to TCSPs. The infinite domains associated with temporal problems prevent us from searching exhaustively through the space of possible scenarios. Instead, we must seek ways of constructing a solution in a guided manner. If the network is decomposable (such as in the case of STPs), a solution can be assembled incrementally, without backtracking, under any ordering we choose. If the network is not decomposable, the feasibility of achieving a backtrack-free solution relies on the topology of the constraint graph. Freuder [20] and Dechter and Pearl [16] have identified sufficient conditions for a network to yield a backtrack-free solution, invoking the notion of higher-order consistency. To demonstrate, we will focus on a class of networks that admit a particularly efficient method when applied to temporal problems. This class is called *series–parallel networks*, and is equivalent to the *regular width-2* networks of [16].

Definition 6.1. A network is said to be *series–parallel* with respect to a pair of nodes, i and j, if it can be reduced to the edge (i, j) by repeated application of the following *reduction* operation: select a node of degree 2 or less, remove it from the network, and connect its neighbors (unless they are connected already). If the network is series–parallel with respect to *any* pair of nodes, it is called a series–parallel network.

Testing whether a network is series–parallel requires $O(|V|)$ time and, as a by-product, the testing algorithm produces an ordering d for which $W^*(d) = 2$, that corresponds to an admissible sequence of reduction operations [4, 50]. It can be shown [16] that enforcing directional path consistency, in an ordering opposite to d, renders such networks backtrack-free, and computes the minimal constraint between the first two nodes in d. If the network is inconsistent, some constraint will become empty, otherwise, a consistent solution can be constructed in a backtrack-free fashion. Since $W^*(d) = 2$, DPC runs in $O(nK)$ time, where K is the maximum number of intervals labeling any edge in the induced graph. The solution construction phase requires an additional $O(nK)$ arithmetic operations.

Montanari [36] showed that full path consistency computes the minimal constraint on every pair of nodes, relative to which the network is series–parallel. In this respect, running full path consistency can be viewed as running DPC along several orderings in parallel, giving any pair of nodes a chance of being the first.

Illustration. Consider the network of Example 1.1. The network is obviously series–parallel, admitting any sequence of reduction operations. Applying DPC in the ordering $d = (0, 1, 2, 3, 4)$ results in the network shown in Fig. 10.
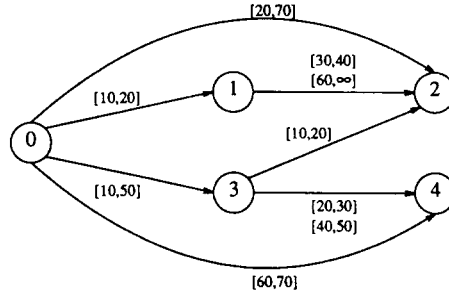
Fig. 10. A directional path consistent network of Example 1.1.

Since no constraint becomes empty, the network is consistent, and a solution can be constructed backtrack-free along d. Moreover, since the network is series–parallel with respect to any pair of nodes, full path consistency computes the full minimal network (see Section 5).

A generalization of directional path consistency, called *adaptive consistency* [16, 17], can render any network backtrack-free by recording higher-order constraints on the neighbors of the nodes processed. This method, although it exhibits a low worst-case complexity in general CSPs, turns out to be ineffective in temporal problems, primarily due to difficulties in storing and processing higher-order interval constraints.

Another approach which exploits the structure of the constraint graph, involves decomposition into *nonseparable components*. We shall show that this can facilitate finding both a consistent solution and the minimal network.

Definition 6.2 (Even [18]). A connected graph $G = (V, E)$ is said to have a *separation vertex* v (sometimes also called an *articulation* point) if there exist vertices a and b, $a \neq v$ and $b \neq v$, such that all the paths connecting a and b pass through v. In this case we also say that v separates a from b. A graph which has a separation vertex is called *separable*, and one which has none is called *nonseparable*. Let $V' \subseteq V$. The induced subgraph $G' = (V', E')$ is called a *nonseparable component* if G' is nonseparable and if for every larger V'', $V' \subset V'' \subseteq V$, the induced subgraph $G'' = (V'', E'')$ is separable.

Definition 6.3 (Even [18]). Let $C_1, \ldots, C_m$ be the nonseparable components of the connected graph $G = (V, E)$, and let $s_1, \ldots, s_p$ be its separating vertices. The *superstructure* of G, $\bar{G} = (\bar{V}, \bar{E})$, is defined as follows:

$$\bar{V} = \{s_1, \ldots, s_p\} \cup \{C_1, \ldots, C_m\} \,, \tag{6.1}$$

$$\bar{E} = \{(s_i, C_j) \mid s_i \text{ is a vertex of } C_j \text{ in } G\} \,. \tag{6.2}$$

It is well known that the superstructure is a tree. The nonseparable components and their superstructure can be found in time $O(|E|)$ (see [18]).

Definition 6.4. Let $G = (V, E)$ be a constraint graph of a TCSP, T, and let $C = (V', E')$ be a nonseparable component of G. The *minimal network of component* C, M_C, is the minimal network of the TCSP defined by C.

Theorem 6.5. *Let M be the minimal network of a consistent TCSP, T, and let M_C be the minimal network of a nonseparable component, $C = (V', E')$, in the constraint graph, $G = (V, E)$, of T. Then, for all $i, j \in V'$, $M_{ij} = (M_C)_{ij}$.*

Proof. Clearly, $M_{ij} \subseteq (M_C)_{ij}$. To prove $(M_C)_{ij} \subseteq M_{ij}$, we show that every value in $(M_C)_{ij}$ also appears in M_{ij}. Let $v \in (M_C)_{ij}$. There exists a labeling, L_1, of C, having minimal network M_{L_1}, in which $v \in (M_{L_1})_{ij}$. Since T is consistent, the TCSP defined by $G - C = (V, E - E')$ is also consistent, and thus there exists a consistent labeling, L_2, of $G - C$. Consider the labeling L whose restrictions to C and $G - C$ are L_1 and L_2, respectively. Let T_L be the STP corresponding to L. T_L is consistent; otherwise, according to Lemma 5.10 it contains an invalid cycle. This cycle must be entirely contained in either C or $G - C$, contradicting the consistency of either L_1 or L_2. Let M_L be the minimal network of T_L. The distance graph of T_L shows that $(M_L)_{ij} = (M_{L_1})_{ij}$, because the shortest paths lengths within C are not affected by the edges of $G - C$. Hence, $v \in (M_L)_{ij}$, thus $v \in M_{ij}$. □

Theorem 6.5 suggests an efficient algorithm for determining consistency and computing the minimal network of a general network. We first find the nonseparable components, $C_1, \ldots, C_m$, and then solve each one of them independently. If all the components are found to be consistent, then the entire network is consistent, and the minimal networks of the individual components coincide with the overall minimal network. If we use backtracking to solve each of the components, then the worst time complexity of this method is $O(nr^3 k^c)$, where r and c denote the largest number of nodes and the largest number of edges in any component, respectively; and k, as before, denotes the maximum number of intervals labeling any edge in the graph. When the topology of any component admits a special, more efficient algorithm, it can be applied directly to that component without affecting the solution of the rest of the problem.

We still must find the minimal constraints on pairs that reside in two different components. This will be determined by Theorem 6.6, after demonstrating how a solution can be constructed to a given TCSP T. We start by finding a solution to the nonseparable component C_0 that contains node 0. All the separation vertices that are connected to C_0 in the superstructure, $\bar{G}$, are instantiated. Then we choose an instantiated separation vertex i, and find a

solution to any nonseparable component C_i that is connected to i in $\bar{G}$, and whose vertices have not been instantiated yet. We continue in this fashion until all the variables are instantiated. Since $\bar{G}$ is a tree, we are guaranteed that once a partial solution of some component has been established, it does not need to be revised.

Theorem 6.6. *Let* $G = (V, E)$ *be the constraint graph of a given TCSP. Let* i *and* j *be two nodes that reside in different nonseparable components of* G, *namely,* $i \in C_i$ *and* $j \in C_j$. *Let* P *be the unique path*

$$P: \quad C_i = C_{i_1}, i_1, C_{i_2}, i_2, \ldots, i_k, C_{i_{k+1}} = C_j, \tag{6.3}$$

that connects C_i *and* C_j *in the superstructure of* G. *Then,*

$$M_{ij} = M_{i,i_1} \otimes M_{i_1,i_2} \cdots \otimes M_{i_{k-1},i_k} \otimes M_{i_k,j}. \tag{6.4}$$

Proof. It suffices to show that

$$M_{i,i_1} \otimes M_{i_1,i_2} \cdots \otimes M_{i_{k-1},i_k} \otimes M_{i_k,j} \subseteq M_{ij}. \tag{6.5}$$

Let

$$v \in M_{i,i_1} \otimes M_{i_1,i_2} \cdots \otimes M_{i_{k-1},i_k} \otimes M_{i_k,j}. \tag{6.6}$$

By definition of the composition operation, there exist values $v_0, \ldots, v_k$, such that $v_0 \in M_{i,i_1}$, $v_j \in M_{i_j,i_{j+1}}$ for $j = 1, \ldots, k-1$, $v_k \in M_{i_k,j}$, and

$$\sum_{j=0}^{k} v_j = v. \tag{6.7}$$

By the minimality of the individual minimal networks, we can construct a solution $X = (x_1, \ldots, x_n)$, that satisfies

$$x_{i_1} - x_i = v_0, \tag{6.8}$$

$$x_{i_{j+1}} - x_{i_j} = v_j, \tag{6.9}$$

for $j = 1, \ldots, k-1$, and

$$x_j - x_{i_k} = v_k. \tag{6.10}$$

Hence,

$$x_j - x_i = v, \tag{6.11}$$

and thus $v \in M_{ij}$. $\square$

The cost of computing a minimal constraint, M_{ij}, using the above method, is $O(k^{cd})$; where c is the size of the largest component that resides along the path connecting C_i and C_j, and d is the length of that path. An alternative upper bound is given by $O(k^e)$. Thus, a full recovery of the minimal network costs $O[n^2 k^{\min(cn, e)}]$.

Illustration. Consider the network of Example 5.4 (Fig. 7). There are two nonseparable components: $C_1 = \{0, 1\}$ and $C_2 = \{1, 2, 3\}$. Component C_1 is a tree and thus already minimal. To compute the minimal network of C_2, we can either apply path consistency (note that C_2 is series–parallel with respect to any pair of nodes) or solve separately the two possible labelings. If M is the minimal network then, by Theorem 6.5:

$$M_{01} = Z_{01}, \qquad M_{12} = Z_{12}, \qquad M_{13} = Z_{13}, \qquad M_{23} = Z_{23}, \qquad (6.12)$$

where Z_{ij} are constraints taken from the minimal networks of the components. The rest of the network can be computed using (6.4):

$$M_{02} = M_{01} \otimes M_{12}, \qquad (6.13)$$

$$M_{03} = M_{01} \otimes M_{13}. \qquad (6.14)$$

Recall that in this example path consistency does compute the minimal network (see Section 5). This phenomenon can be explained by Theorems 6.5 and 6.6. We already noted that path consistency computes the minimal networks of both components. We now show that in general, this should suffice for computing the minimal constraints on edges that go across components. When path consistency terminates, the computed constraints, T_{ij}, satisfy:

$$T_{02} \subseteq T_{01} \otimes T_{12} \qquad (6.15)$$

and

$$T_{03} \subseteq T_{01} \otimes T_{13}. \qquad (6.16)$$

Together with (6.12)–(6.14), we get

$$T_{02} \subseteq M_{02}, \qquad (6.17)$$

$$T_{03} \subseteq M_{03}. \qquad (6.18)$$

Since M is minimal, $T_{02} = M_{02}$ and $T_{03} = M_{03}$; namely, path consistency computes the full minimal network.

Finally, we note that another network-based approach for solving general CSPs, the *cycle-cutset method* [14], cannot be employed beneficially in temporal problems. The reason is that the backtracking used in the solution of TCSPs instantiates arcs, rather than variables, and many such instantiations are needed to decompose the original network.

7. Relations to other formalisms

In this section we relate the TCSP model to two other models of temporal reasoning—Allen's interval algebra and Vilain and Kautz's point algebra. We show how the constraints in these representation schemes can be encoded within the TCSP model. To facilitate such encoding, we allow the interval representation of our constraints to include open and semi-open intervals, with the obvious effect on the definitions of the union and intersection operations. Similarly, an interval that results from a composition operation may be open on one side or on both sides, depending on the operands. For example,

$$\{[1, 2], (6, 8)\} \otimes \{[0, 3), (12, 15]\}$$
$$= \{[1, 5), (6, 11), (13, 17], (18, 23)\} . \tag{7.1}$$

It is easy to verify that all our theorems still hold with this extended provision.

Any constraint network in Vilain and Kautz's point algebra [49] is a special case of a TCSP, lacking metric information. It can be viewed as a CSP involving a set of variables, $X_1, \ldots, X_n$, and binary constraints of the form $X_i \, R \, X_j$, where

$$R \in \{<, \leqslant, >, \geqslant, =, \neq\} . \tag{7.2}$$

The translation into TCSP is straightforward. Constraints of the form $X_j < X_i$ and $X_j \leqslant X_i$ are expressed by the interval representations $T_{ij} = \{(-\infty, 0)\}$ and $T_{ij} = \{(-\infty, 0]\}$, respectively. The constraint $X_i = X_j$ translates into $T_{ij} = \{[0]\}$. The only relation that needs to be represented by a disjunction is $X_i \neq X_j$, translated into $T_{ij} = \{(-\infty, 0), (0, \infty)\}$.

Vilain and Kautz have addressed the tasks of determining consistency and computing the minimal network for problems expressed in the point algebra. They suggested the use of path consistency for computing the minimal network, which turned out to be insufficient [47]. Van Beek [47] addressed a subset of the point algebra, called PA, which excludes $\neq$. He showed that constraint networks in PA may be solved in time $O(n^3)$ by applying path consistency. This follows immediately from the TCSP representation, since every constraint network in PA is equivalent to an STP with edges labeled by

intervals from

$$\{(-\infty, 0), (-\infty, 0], [0], [0, \infty), (0, \infty)\} . \qquad (7.3)$$

Thus, when the constraints are taken from (7.3), path consistency for TCSP coincides with path consistency for PA. Moreover, algorithms devised for solving STPs' tasks reduce to equivalent, often simpler algorithms for solving the same tasks in PA. For example, directional path consistency can determine consistency in PA in $O(nW^*(d)^2)$ operations, which amounts to linear time when $W^*(d)$ is bounded.

The full point algebra, including the inequality constraint $\neq$, translates into TCSPs with disjunctions, for which our general methods can be applied, and the special structure of the constraints exploited. In [47] it is shown that enforcing 4-consistency suffices for computing the minimal network in the point algebra. This result takes special advantage of the nonmetric nature of the relations in (7.2). More recently, it has been found that path consistency is sufficient for determining consistency in the full point algebra [35]. This establishes an $O(n^3)$ complexity for general networks in Vilain and Kautz's point algebra.[5] A more efficient method has been reported recently by Van Beek [48], requiring $O(n^2)$ time (see also [35]).

In contrast, Allen's interval algebra [2] cannot be translated into binary TCSPs. It can be viewed as a CSP involving a set of variables, $X_1, \ldots, X_n$, whose domains are pairs of time points, representing the beginning and ending times of temporal events. The allowed relationships between pairs of variables are taken from the set

$$\{before, meets, overlaps, during, starts, finishes\} , \qquad (7.4)$$

their inverses, and the equality relation—a total of 13 relations. The translation into TCSP introduces nonbinary constraints. For example, the constraint

$$A \ (before \vee after) \ B , \qquad (7.5)$$

where intervals A and B are given by $A = [X_1, X_2]$ and $B = [X_3, X_4]$, cannot be encoded by a binary TCSP constraint [49]; it requires the 4-ary constraint

$$(X_2 < X_3) \vee (X_4 < X_1) . \qquad (7.6)$$

Problems involving higher-order constraints can be expressed as disjunctions of STPs, and solutions can be assembled by taking the union of the individual STP solutions. Although the number of such subproblems may be large, advantage can be taken of the simple procedures available for solving each

---

[5] We have recently learned that this had also been established by Ladkin and Maddux, The algebra of constraint satisfaction problems and temporal reasoning, Tech. Rept., Kestrel Institute, Palo Alto, CA (1988).

STP. It seems likely, however, that unless metric constraints are specified, the representation suggested in [2] can be handled more conveniently.

8. Summary and conclusions

This paper provides a formal framework for dealing with temporal constraints, encouraging the transference of algorithms and theoretical results developed for general constraint satisfaction problems. We distinguish between simple temporal problems (STPs) and general temporal problems, the former admitting at most one interval constraint on any pair of time points. We show that the STP can be solved in polynomial time, using the well-known Floyd–Warshall's all-pairs-shortest-paths algorithm. For general TCSPs, we present a decomposition scheme that provides answers to the reasoning tasks considered, but its computational efficiency, in the worst case, might be limited. The decomposition scheme might be improved by traditional constraint satisfaction techniques, such as backjumping, learning, various ordering schemes and preprocessing techniques. We studied the applicability of path consistency algorithms as preprocessing of temporal problems, demonstrated their termination and bounded their complexity; they seem to offer a practical compromise in very complex problems. In particular, the more efficient directional path consistency was shown to retain the essential properties of full path consistency in determining consistency for STPs and in enhancing backtrack search of general TCSPs. Among the specialized network-based algorithms, only the one employing decomposition into nonseparable components was found applicable to TCSPs. It offers a method for computing the minimal network in time exponential in the largest nonseparable component.

We see the main application of our framework to be in temporal reasoning tasks involving metric information, namely, expressions involving absolute time differences (e.g. "John came home an hour after Mary"). In this respect, the expressiveness of our language supercedes that of Allen's interval algebra. However, it can be considered weaker than the interval algebra, being limited to problems involving constraints on pairs of time points. Our framework subsumes Vilain and Kautz's point algebra as a special case, and provides an arsenal of techniques and intuitions for solving problems in this domain. The natural extension of this work is to explore TCSPs with higher-order expressions (e.g., "John drives to work at least 30 minutes more than Fred does"; $X_2 - X_1 + 30 \leq X_4 - X_3$), with the aim of exploiting the unique algebraic features provided by the linearity and continuity of temporal constraints. An extension of this work, integrating our quantitative analysis with the qualitative framework of Allen's algebra can be found in a recent article.[6]

---

[6] I. Meiri, Combining qualitative and quantitative constraints in temporal reasoning, in: *Proceedings AAAI-91*, Anaheim CA (1991).

Appendix A. Proof of Corollary 3.5

Corollary 3.5. *Given a consistent STP, T, the equivalent STP, M, defined by*

$$\forall i, j , \quad M_{ij} = \{[-d_{ji}, d_{ij}]\} , \tag{A.1}$$

is the minimal network representation of T.

Proof. We will show that M is the minimal network by showing that it cannot be tightened any more; in other words, starting with the assignment $X_0 = 0$, for any $d \in [-d_{ji}, d_{ij}]$ there exists a solution $X = (x_0, \ldots, x_n)$ in which $x_j - x_i = d$.
    Case 1.

$$d \le d_{0j} - d_{0i} . \tag{A.2}$$

According to Corollary 3.4, $X_i = d_{0i}$ is a feasible value. Clearly,

$$d_{0i} + d \ge d_{0i} - d_{ji} , \tag{A.3}$$

and since

$$d_{ji} \le d_{j0} + d_{0i} , \tag{A.4}$$

we get

$$d_{0i} + d \ge -d_{j0} . \tag{A.5}$$

Together with (A.2) we have

$$-d_{j0} \le d_{0i} + d \le d_{0j} . \tag{A.6}$$

Therefore, the assignment $X_j = d_{0i} + d$ satisfies the unary domain constraints on variable X_j, and

$$\{X_0 = 0, X_i = d_{0i}, X_j = d_{0i} + d\} \tag{A.7}$$

satisfies the constraints applicable to $\{X_0, X_i, X_j\}$. By Theorem 3.3 this partial assignment can be extended to a solution.
    Case 2.

$$d \ge d_{0j} - d_{0i} . \tag{A.8}$$

According to Corollary 3.4, $X_j = d_{0j}$ is a feasible value. Clearly,

$$d_{0j} - d \ge d_{0j} - d_{ij} , \tag{A.9}$$

and since

$$d_{ij} \leq d_{i0} + d_{0j} ,$$ (A.10)

we get

$$d_{0j} - d \geq -d_{i0} .$$ (A.11)

Together with (A.8) we have

$$-d_{i0} \leq d_{0j} - d \leq d_{0i} .$$ (A.12)

Therefore, the assignment $X_i = d_{0j} - d$ satisfies the unary domain constraints on variable X_i, and

$$\{X_0 = 0, X_i = d_{0j} - d, X_j = d_{0j}\}$$ (A.13)

satisfies the constraints applicable to $\{X_0, X_i, X_j\}$. By Theorem 3.3, this partial assignment can be extended to a solution. $\square$

Appendix B. Proof of Lemma 5.10

Lemma 5.10. *A given STP, T, is consistent if and only if all the cycles in its constraint graph are valid.*

Proof. If the network is consistent, then all the cycles are valid, since if there was an invalid cycle C, $i_0, \ldots , i_k = i_0$, then the path constraint along C would yield

$$X_{i_0} - X_{i_0} \neq 0 ,$$ (B.1)

reflecting inconsistency.

Conversely, assume that all the cycles are valid. We will show that the network is consistent. According to Theorem 3.1 we only need to show that there is no negative cycle in the corresponding distance graph. Suppose there was such a negative cycle, C, consisting of nodes $i_0, \ldots , i_k = i_0$, and edge weights $a_{0,1}, a_{1,2}, \ldots , a_{k-1,k} = a_{k-1,0}$. Since C is negative, we have

$$\sum_{j=1}^{k} a_{i_{j-1},i_j} < 0 .$$ (B.2)

Moreover, from (3.1)–(3.3) we obtain

$$-a_{i_j,i_{j-1}} \leq a_{i_{j-1},i_j}$$ (B.3)

for $j = 1, \ldots , k$. Thus, combining (B.2) and (B.3) yields

$$0 \not\in \left[\sum_{j=1}^{k} -a_{i_j, i_{j-1}}, \sum_{j=1}^{k} a_{i_{j-1}, i_j} \right].$$

(B.4)

At the same time, applying the composition along C gives

$$T_{i_0, i_1} \otimes \cdots \otimes T_{i_{k-1}, i_k} = \left[\sum_{j=1}^{k} -a_{i_j, i_{j-1}}, \sum_{j=1}^{k} a_{i_{j-1}, i_j} \right],$$

(B.5)

thus rendering C invalid—a contradiction. □

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] J.F. Allen, Maintaining knowledge about temporal intervals, *Commun. ACM* **26** (11) (1983) 832–843.

[3] J.F. Allen, Towards a general theory of action and time, *Artif. Intell.* **23** (2) (1984) 123–154.

[4] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability—a survey, *BIT* **25** (1985) 2–23.

[5] S. Arnborg, D.G. Corneil and A. Proskurowski, Complexity of finding embeddings in a *k*-tree, *SIAM J. Algebraic Discrete Methods* **8** (2) (1987) 177–184.

[6] B. Aspvall and Y. Shiloach, A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality, *SIAM J. Comput.* **9** (4) (1980) 827–845.

[7] R.C. Backhouse and B.A. Carré, Regular algebra applied to path-finding problems, *J. Inst. Math. Appl.* **15** (1975) 161–186.

[8] C.E. Bell and A. Tate, Use of a longest path algorithm to manage temporal information and restrict search in an automated planner, Working Paper Series No. 85-34, Artificial Intelligence Applications Institute, University of Edinburgh, Scotland (1985).

[9] U. Bertelé and F. Brioschi, *Nonserial Dynamic Programming* (Academic Press, New York, 1972).

[10] G.B. Dantzig, *Linear Programming and Extensions* (Princeton University Press, Princeton, NJ, 1962).

[11] E. Davis, Constraint propagation with interval labels, *Artif. Intell.* **32** (3) (1987) 281–331.

[12] E. Davis, Private communication (1989).

[13] T.L. Dean and D.V. McDermott, Temporal data base management, *Artif. Intell.* **32** (1987) 1–55.

[14] R. Dechter, Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition, *Artif. Intell.* **41** (3) (1990) 273–312.

[15] R. Dechter and I. Meiri, Experimental evaluation of preprocessing techniques in constraint satisfaction problems, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 271–277.

[16] R. Dechter and J. Pearl, Network-based heuristics for constraint satisfaction problems, *Artif. Intell.* **34** (1) (1987) 1–38.

[17] R. Dechter and J. Pearl, Tree clustering for constraint networks, *Artif. Intell.* **38** (3) (1989) 353–366.

[18] S. Even, *Graph Algorithms* (Computer Science Press, Rockville, MD, 1979).

[19] E.C. Freuder, Synthesizing constraint expressions, *Commun. ACM* **21** (11) (1978) 958–965.

[20] E.C. Freuder, A sufficient condition of backtrack-free search, *J. ACM* **29** (1) (1982) 24–32.

[21] J. Gaschnig, Performance measurement and analysis of certain search algorithms, Tech. Rept. CMU-CS-79-124, Carnegie-Mellon University, Pittsburgh, PA (1979).

[22] S. Hanks and D.V. McDermott, Default reasoning, nonmonotonic logics, and the frame problem, in: *Proceedings AAAI-86*, Philadelphia, PA (1986) 328–333.

[23] R.M. Haralick and G.L. Elliott, Increasing tree search efficiency for constraint satisfaction problems, *Artif. Intell.* **14** (1980) 263–313.

[24] K. Kahn and G.A. Gorry, Mechanizing temporal knowledge, *Artif. Intell.* **9** (1977) 87–108.

[25] L.G. Khachiyan, A polynomial algorithm in linear programming, *Soviet Math. Dokl.* **20** (1979) 191–194.

[26] P.B. Ladkin, Metric constraint satisfaction with intervals, Tech. Rept. TR-89-038, International Computer Science Institute, Berkeley, CA (1989).

[27] P.B. Ladkin and R.D. Maddux, On binary constraint networks, Tech. Rept., Kestrel Institute, Palo Alto, CA (1989).

[28] D.J. Lehmann, Algebraic structures for transitive closure, *Theor. Comput. Sci.* **4** (1977) 59–76.

[29] C.E. Leiserson and J.B. Saxe, A mixed-integer linear programming problem which is efficiently solvable, in: *Proceedings 21st Annual Allerton Conference on Communications, Control, and Computing* (1983) 204–213.

[30] Y.Z. Liao and C.K. Wong, An algorithm to compact a VLSI symbolic layout with mixed constraints, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* **2** (2) (1983) 62–69.

[31] A.K. Mackworth, Consistency in networks of relations, *Artif. Intell.* **8** (1) (1977) 99–118.

[32] A.K. Mackworth and E.C. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Artif. Intell.* **25** (1) (1985) 65–74.

[33] J. Malik and T.O. Binford, Reasoning in time and space, in: *Proceedings IJCAI-83*, Karlsruhe, FRG (1983) 343–345.

[34] D.V. McDermott, A temporal logic for reasoning about processes and plans, *Cogn. Sci.* **6** (1982) 101–155.

[35] I. Meiri, Faster constraint satisfaction algorithms for temporal reasoning, Tech. Rept. R-151, UCLA Cognitive Systems Laboratory, Los Angeles, CA (1990).

[36] U. Montanari, Networks of constraints: fundamental properties and applications to picture processing, *Inf. Sci.* **7** (1974) 95–132.

[37] B. Nudel, Consistent-labeling problems and their algorithms: expected-complexities and theory-based heuristics, *Artif. Intell.* **21** (1983) 135–178.

[38] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity* (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[39] D.S. Parker, Partial order programming, Tech. Rept. CSD-870067, UCLA, Los Angeles, CA (1987).

[40] P.W. Purdom, Search rearrangement backtracking and polynomial average time, *Artif. Intell.* **21** (1983) 117–133.

[41] Y. Shoham, *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence* (MIT Press, Cambridge, MA, 1988).

[42] R. Shostak, Deciding linear inequalities by computing loop residues, *J. ACM* **28** (4) (1981) 769–779.

[43] R.E. Tarjan, A unified approach to path problems, *J. ACM* **28** (3) (1981) 577–593.

[44] R.E. Tarjan, Fast algorithms for solving path problems, *J. ACM* **28** (3) (1981) 594–614.

[45] R.E. Tarjan and M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs, *SIAM J. Comput.* **13** (3) (1984) 566–579.

[46] R.E. Valdés-Pérez, Spatio-temporal reasoning and linear inequalities, Artificial Intelligence Laboratory, AIM-875, MIT, Cambridge, MA (1986).

[47] P. Van Beek, Approximation algorithms for temporal reasoning, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 1291–1296.

[48] P. Van Beek, Reasoning about qualitative temporal information, in: *Proceedings AAAI-90*, Boston, MA (1990) 728–734.

[49] M. Vilain and H. Kautz, Constraint propagation algorithms for temporal reasoning, in: *Proceedings AAAI-86*, Philadelphia, PA (1986) 377–382.

[50] J.A. Wald and C.J. Colbourn, Steiner trees, partial 2-trees, and minimum IFI networks, *Networks* **13** (1983) 159–167.