7-13-2005

An Efficient Consistency Algorithm for the Temporal Constraint Satisfaction Problem

Berthe Y. Choueiry
University of Nebraska-Lincoln, choueiry@cse.unl.edu

Lin Xu
University of Nebraska-Lincoln

AI Communications 17 (2004) 213–221
IOS Press

An efficient consistency algorithm for the Temporal Constraint Satisfaction Problem

Berthe Y. Choueiry * and Lin Xu
Constraint Systems Laboratory, Department of Computer Science and Engineering, University of Nebraska-Lincoln, USA
E-mail: {choueiry,lxu}@cse.unl.edu

Abstract. Dechter et al. [5] proposed solving the Temporal Constraint Satisfaction Problem (TCSP) by modeling it as a meta-CSP, which is a finite CSP with a unique global constraint. The size of this global constraint is exponential in the number of time points in the original TCSP, and generalized-arc consistency is equivalent to finding the minimal network of the TCSP, which is NP-hard. We introduce △AC, an efficient consistency algorithm for filtering the meta-CSP. This algorithm significantly reduces the domains of the variables of the meta-CSP without guaranteeing arc-consistency. We use △AC as a preprocessing step to solving the meta-CSP. We show experimentally that it dramatically reduces the size of a meta-CSP and significantly enhances the performance of search for finding the minimal network of the corresponding TCSP.

Keywords: Constraint temporal networks, consistency algorithm

1. Introduction

In this paper we study constraint propagation in networks of metric temporal constraints, which are an essential tool for building systems that reason about time and actions. These networks model events and their relationships (as distances between events), and provide the means to specify the temporal elements of an episode with a temporal extent. Examples of such an episode are as diverse as a story, a discourse, a manufacturing process, the measurements executed by the Hubble space telescope, the activities of a robot, or the scheduling of a summer vacation. The ability to efficiently process temporal networks is a pre-requisite for enabling computers to support human users in decision making and to automate the planning and execution of complex engineering tasks. This paper describes an efficient consistency algorithm for the meta-CSP modeling the Temporal Constraint Satisfaction Problem (TCSP) [5].

A major research effort in the Constraint Processing community is the development of efficient filtering algorithms. These algorithms propagate the constraints in a problem in order to reduce its size and enhance the performance of the algorithms used for solving it. Although particularly simple at the conceptual level, the basic mechanism for ensuring arc-consistency (AC) [7,8,12] has witnessed several refinements [1,2,6,9] and remains the subject of intensive research [3,15]. The unusual attention to a mechanism executable in polynomial time is justified by the fact that this simple mechanism is at the heart of many procedures for solving CSPs.

To the best of our knowledge, the only work reported in the literature on applying consistency algorithms to the meta-CSP is a study by Schwalb and Dechter [11], which we discuss in Section 3.3. Shortly stated, the above study changes the end points of the temporal intervals. In contrast, our approach considers each interval as an atomic value, which is either kept or removed, but whose extent is never modified.

In this paper we argue that arc-consistency of the meta-CSP is NP-hard. We define the property of △arc-consistency of the meta-CSP and propose an efficient algorithm, △AC, for achieving it. This algorithm, which guarantees △arc-consistency of the meta-CSP but, not its arc-consistency, drastically reduces the size of the meta-CSP and enhances the performance of the search process used for solving it. While the basic idea behind our filtering algorithm is simple, the value of our contribution lies in the design of polynomial-time and space data-structures, reminiscent of AC-4 [9] and AC-2001 [3] for general CSPs, that make the algo-

*Corresponding author. B.Y. Choueiry, 256 Avery Hall, Lincoln, NE, 68588-0115, USA.

rithm particularly efficient and perhaps even optimal for achieving the property of △arc-consistency. Note that optimality still needs to be formally established.

This paper is structured as follows. Section 2 introduces our notation, the task we address and its complexity. Section 3 introduces the concept of the △AC-consistency of the meta-CSP and the algorithm for achieving it. Section 4 describes our experiments and observations. Finally, Section 5 concludes this paper.

2. Background

We first define formally the temporal constraint problems addressed.

2.1. STP and TCSP

A Simple Temporal Problem (STP) is defined by a graph $G = (V, E, I)$, where V is a set of vertices t_i representing time points, E is a set of directed edges $e_{i,j}$ representing constraints between two time points t_i and t_j, and I is a set of constraint labels for the edges (see Fig. 1). A constraint label $I_{i,j}$ of edge $e_{i,j}$ is a *unique* interval $[a, b]$, a and $b \in \mathbb{R}$, and denotes a constraint of bounded difference $a \leqslant (t_j - t_i) \leqslant b$. We assume that there is at most one constraint between any two vertices t_i and t_j and that the constraint $e_{i,j}$ labeled $[a, b]$ can also be referred to as the constraint $e_{j,i}$ labeled $[-b, -a]$.

A Temporal Constraint Satisfaction Problem (TCSP) is defined by a similar graph $G = (V, E, I)$, where each edge label $I_{i,j} = \{l_{ij}^1, l_{ij}^2, \ldots, l_{ij}^k\}$ is a *set* of disjoint intervals denoting a disjunction of constraints of bounded differences between t_i and t_j (see Fig. 2). We assume that the intervals in a label are given in a canonical form in which all intervals are pair-wise disjoint,

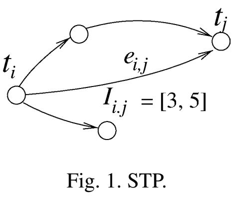

Fig. 1. STP.

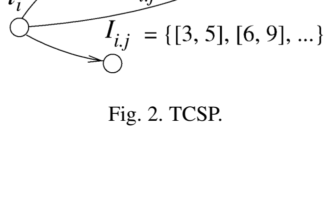

Fig. 2. TCSP.

and that they are sorted in an increasing order of their end points. The superscript k of interval l_{ij}^k denotes the position of the interval in the domain. This ordering scheme is important for the specification of our algorithm.

Solving a temporal constraint network corresponds to assigning a value to each time point all the constraints are simultaneously satisfied. Finding the equivalent minimal network can be accomplished by removing from the edge labels the values that do not participate in any solution. Solving an STP and finding its minimal network can be done in polynomial time. For example, the Floyd-Warshall algorithm for computing all pairs shortest paths computes the minimal network in $\Theta(n^3)$, where n is the number of nodes, or time points, in the network. Solving the TCSP is NP-complete and finding its minimal network is NP-hard [5].

2.2. The meta-CSP

Dechter et al. [5] described a backtrack search procedure for determining the consistency of the TCSP. To this end, the TCSP is expressed as a "meta" Constraint Satisfaction Problem (CSP), or meta-CSP. The variables of the meta-CSP are the edges $e_{i,j}$ of G. Their number $|E|$ depends on the density of the temporal graph and may reach $n(n-1)/2$, where n is the number of nodes in the TCSP. The domain of a variable $e_{i,j}$, denoted $\texttt{Domain}(e_{i,j})$, is its label, $I_{i,j} = \{l_{ij}^1, l_{ij}^2, \ldots, l_{ij}^k\}$. The size of the meta-CSP, defined as the product of the domain sizes of its variables $\Pi_{e_{i,j} \in E}|I_{i,j}|$, is $k^{|E|}$. A variable-value pair is a tuple of a variable and a value from its domain. The only constraint in the meta-CSP is a global constraint that requires the variable-value pairs (vvps) $\{(e_{i,j}, l_{ij}^h)\}$ for all the variables $e_{i,j} \in G$ to form a consistent STP. The size of this constraint (i.e., number of possible tuples) is $k^{|E|}$; it can reach $k^{n(n-1)/2}$ and is exponential in the number of time points in the TCSP. Solving the meta-CSP corresponds to assigning one interval to each edge from its label such that the resulting temporal network forms a consistent STP. The backtrack search proposed by Dechter et al. [5] for solving the meta-CSP requires checking the consistency of an STP at every node in the search, each of which is $O(n^3)$. Its complexity is thus $O(n^3 k^{|E|})$. The consistency of the TCSP can be determined by finding a solution to the meta-CSP; and finding the minimal network of the TCSP can be achieved by finding all the minimal STP networks that are solutions of the meta-CSP [5].

2.3. Consistency of the meta-CSP

The only constraint in the meta-CSP is a global constraint. The application of generalized arc-consistency to this constraint requires finding all its tuples [10]. Finding the constraint definition is hence equivalent to finding all the solutions of the meta-CSP, which is NP-hard [5]. Thus, running a generalized arc-consistency algorithm on the meta-CSP is prohibitively expensive.

Proposition 2.1. *Generalized arc-consistency on the meta-CSP is NP-hard.*

We propose to reduce the size of the meta-CSP by considering a ternary constraint between every three nodes of the meta-CSP forming a triangle in the graph of the TCSP, and applying an efficient generalized arc-consistency algorithms, which we call $\triangle$AC, to these ternary constraints (see Fig. 3). The complexity of $\triangle$AC is $O(\mathit{degree}(G) \cdot |E| \cdot k^3) = O(n|E|k^3)$, where *degree*$(G)$ denotes the largest number of edges incident to any vertex in G. Again, the $\triangle$AC algorithm achieves $\triangle$arc-consistency of the meta-CSP, but does not guarantee that the resulting meta-CSP is arc-consistent. However, it provides an efficient way to reduce its size.

In order to demonstrate the effectiveness of our approach, we test and report the performance of $\triangle$AC as a preprocessing step to search, showing a dramatic reduction in the size of the meta-CSP. We also report the performance improvement of the backtrack search for solving the meta-CSP with and without this preprocessing in terms of CPU time and number of constraint checks CC.

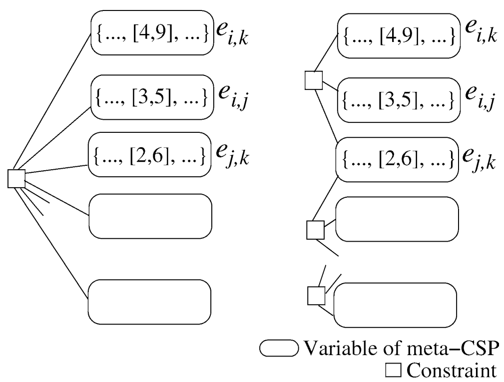

Fig. 3. Left: Meta-CSP with a global constraint. Right: with ternary constraints.

3. The filtering algorithm

We approximate the generalized arc-consistency of the meta-CSP by replacing the unique global constraint with a ternary constraint $\triangle[e_{i,j}, e_{i,k}, e_{j,k}]$ among every variable $e_{i,j}$, $e_{i,k}$, and $e_{j,k}$ of the meta-CSP that forms an *existing* triangle in the temporal network G. Below, we define the $\triangle$arc-consistency property as the generalized arc-consistency of this constraint and describe the $\triangle$AC algorithm to achieve it.

3.1. $\triangle$arc-consistency

An STP can be solved by computing the transitive closure under composition and intersection of the intervals of its edges. The transitive closure of an STP results in a complete temporal graph.

The *composition* $l_{ik} = l_{ij} \circ l_{jk}$ of the intervals $l_{ij} = [a, b]$ and $l_{jk} = [c, d]$ labeling the respective edges $e_{i,j}$ and $e_{j,k}$ is a new interval $l_{ik} = [a + c, b + d]$ labeling the edge $e_{i,k}$.

The *intersection* $l_{i,k} = l'_{i,k} \cap l''_{i,k}$ of the intervals $l'_{ik} = [a, b]$ and $l''_{ik} = [c, d]$ labeling the respective $e'_{i,k}$ and $e''_{i,k}$ is a new interval $l_{ik} = [\text{maximum}(a, c), \text{minimum}(b, d)]$ labeling the edge $e_{i,k}$.

We use the above two operations to define the property of $\triangle$AC of a meta-CSP. For each triangle ijk connecting the distinct time points t_i, t_j, and t_k in the original temporal network, we define a ternary constraint in the meta-CSP $\triangle[e_{i,j}, e_{i,k}, e_{j,k}]$. Given three variable-value pairs $(e_{i,j}, l_{ij})$, $(e_{i,k}, l_{ik})$, and $(e_{j,k}, l_{jk})$ of the meta-CSP, we say that the *labeled triangle* $\triangle[(e_{i,j}, l_{ij}), (e_{i,k}, l_{ik}), (e_{j,k}, l_{jk})]$ is a consistent triangle if and only if $(l_{ij} \circ l_{jk}) \cap l_{ik} \neq \emptyset$. Figure 4 shows a consistent triangle $\triangle[(e_{i,j}, [3, 5]), (e_{i,k}, [4, 9]), (e_{j,k}, [2, 6])]$. We also say that each variable-value pair in the triangle is supported by the two other variable-value pairs. We introduce the following three definitions:

1. The ternary constraint $\triangle[e_{i,j}, e_{i,k}, e_{j,k}]$ is $\triangle$AC relative to the meta-CSP variable $e_{i,j}$ if and only if for every interval $l^x_{ij} \in \mathtt{Domain}(e_{i,j})$ there exists an interval $l^y_{ik} \in \mathtt{Domain}(e_{i,k})$ and an interval $l^z_{kj} \in \mathtt{Domain}(e_{k,j})$ such that $(l^y_{ik} \circ l^z_{kj}) \cap l^x_{ij} \neq \emptyset$.

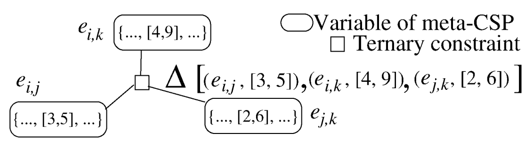

Fig. 4. A consistent triangle.

```
First-support(⟨(e_{i,j}, l_{ij}), ijk⟩)
t_{ijk} ← Supported-by(⟨(e_{i,j}, l_{i,j}), ijk⟩)
If t_{ijk} = nil
  Then r ← 1, s ← 0
  Else let t_{ijk} be of
         the form △[(e_{i,j}, l_{ij}), (e_{i,k}, l^r_{ik}), (e_{j,k}, l^s_{jk})]
       r ← position of l^r_{ik} in |Domain(e_{i,k})|
       s ← position of l^s_{jk} in |Domain(e_{j,k})|
For m from (s + 1) to |Domain(e_{j,k})|
  When (l^r_{ik} ∘ l^m_{jk}) ∩ l_{ij}
    Return △[(e_{i,j}, l_{ij}), (e_{i,k}, l^r_{ik}), (e_{j,k}, l^m_{jk})]
When r ≠ |Domain(e_{i,k})|
  For n from (r + 1) to |Domain(e_{i,k})|
    For t from 1 to |Domain(e_{j,k})|
      When (l^n_{ik} ∘ l^t_{jk}) ∩ l_{ij}
        Return △[(e_{i,j}, l_{ij}), (e_{i,k}, l^n_{ik}), (e_{j,k}, l^t_{jk})]
Return nil
```

Fig. 5. `First-support`.

2. The ternary constraint $\triangle[e_{i,j}, e_{i,k}, e_{j,k}]$ is $\triangle$AC if and only if it is $\triangle$AC relative to the variables $e_{i,j}$, $e_{i,k}$, and $e_{j,k}$.
3. Finally, the meta-CSP is $\triangle$AC if and only if all its ternary constraints are $\triangle$AC.

We identify all the existing triangles in the temporal network and replace each of them by a ternary triangle constraint. The number of these new constraints is in $O(\mathit{degree}(G) \cdot |E|) = O(n|E|)$, and the size of each constraint is at most k^3.

3.2. $\triangle$AC algorithm

The $\triangle$AC algorithm, shown in Fig. 7, removes the intervals in the domain of an $e_{i,j}$ that do not have a support in any triangle in which $e_{i,j}$ appears in the temporal graph. It implements mechanisms for consistency checking that are reminiscent of AC-4 [9] and AC-2001 [3] in that it tries to optimize the effort for consistency checking. It uses the procedures `First-support` of Fig. 5 and `Initialize-support` of Fig. 6. The `Push` and `Delete` operations we use are destructive stack operations.

$\triangle$AC operates by looking at every combination of a vvp $(e_{i,j}, l_{ij})$ and the triangles ijk in which it appears, denoted $\langle(e_{i,j}, l_{ij}), ijk\rangle$. The support of $\langle(e_{i,j}, l_{ij}), ijk\rangle$ is the first element in the domains of $e_{i,k}$ and $e_{j,k}$ that yields a consistent triangle. (Note that domains are and variables are ordered canonically.) Intervals in the domain of a variable that are not supported in any triangle are removed from the domain. When an interval is removed, some vvps may lose their support. $\triangle$AC tries to find the next acceptable support. The process is repeated until all vvps have a valid support in every relevant triangle.

```
Initialize-support(G)
Support-by, Supports: two empty hash-tables
Q ← {(e_{i,j}, l_{ij})}, set of all vvps in the meta-CSP
Q' ← nil
Consistency ← true
While nonempty(Q) ∧ Consistency do
  (e_{i,j}, l_{ij}) ← Pop(Q)
  Forall k such that ijk is a subgraph of G do
    t_{ijk} ← △[(e_{i,j}, l_{ij}), (e_{i,k}, l_{ik}), (e_{j,k}, l_{jk})]
            ← First-support(⟨(e_{i,j}, l_{ij}), ijk⟩)
    If t_{ijk}
      Then Supported-by[(e_{i,j}, l_{ij}), ijk] ← t_{ijk}
           Push(⟨e_{i,j}, l_{ij}, ijk⟩, Supports[(e_{i,k}, l_{ik})])
           Push(⟨e_{i,j}, l_{ij}, ijk⟩, Supports[(e_{j,k}, l_{jk})])
      Else Domain(e_{i,j}) ← Domain(e_{i,j}) \ {l_{ij}}
           Push((e_{i,j}, l_{ij}), Q')
           When Domain(e_{i,j}) = ∅
             Then Consistency ← false
Return Q', Supported-by, Supports, Consistency
```

Fig. 6. `Initialize-support`.

We use a hash-table `Supported-by` to keep track of the support of each vvp $(e_{i,j}, l_{ij})$ in a triangle ijk. A key in this hash-table is a tuple $\langle(e_{i,j}, l_{ij}), ijk\rangle$; its value is a consistent triangle $\triangle[(e_{i,j}, l_{ij}), (e_{i,k}, l_{ik}), (e_{j,k}, l_{jk})]$. The size of `Supported-by` is $O(|E|k\,\mathit{degree}(G))$.

We also use a hash-table `Supports` to keep track of what a given vvp supports in `Supported-by`. The key is a vvp $(e_{i,j}, l_{ij})$, and the value is a list of the keys of `Supported-by` that this vvp supports. By construction, `Supports` has $O(|E|k)$ keys and a total of $O(|E|k\,\mathit{degree}(G))$ elements.

`Initialize-support` initializes hash-tables. It returns these data structures, the list Q' of vvps deleted from the domains of the meta-CSP at the initialization step, and a boolean variable indicating whether the meta-CSP was already found inconsistent. $\triangle$AC, shown in Fig. 7, iterates over the vvps that have been deleted and retracts them from supporting entries in `Supported-by`.

We can prove that $\triangle$AC terminates, does not remove any consistent intervals (i.e., is sound), and is in $O(\mathit{degree}(G)|E|k^3) = O(n|E|k^3)$.

3.3. Discussion

Note that we do *not* add any edges to the network of the TCSP to triangulate it or to make it a complete graph. *Such an effort would be useless for the*

$\triangle$**AC**(G)
Q
`Supported-by`
`Supports`
Consistency $\leftarrow$ `Initialize-support`(G)
While nonempty(Q) $\wedge$ Consistency **do**
  $(e_{i,k}, l_{ik}) \leftarrow$ `Pop`(Q)
  Forall $\langle e_{i,j}, l_{ij}, ijk\rangle \in$ `Supports`$[(e_{i,k}, l_{ik})]$)
    $t_{ijk} \leftarrow \triangle[(e_{i,j}, l_{ij}), (e_{i,k}, l_{ik}), (e_{j,k}, l_{jk})]$
      $\leftarrow$ `Supported-by`$(\langle(e_{i,j}, l_{ij}), ijk\rangle)$
    `Delete`$(\langle(e_{i,j}, l_{ij}), ijk\rangle,$ `Supports`$[(e_{i,k}, l_{ik})])$
    `Delete`$(\langle(e_{i,j}, l_{ij}), ijk\rangle,$ `Supports`$[(e_{j,k}, l_{jk})])$
    $t'_{ijk} \leftarrow \triangle[(e_{i,j}, l_{ij}), (e_{i,k}, l'_{ik}), (e_{j,k}, l'_{jk})]$
      $\leftarrow$ `First-support`$(\langle(e_{i,j}, l_{ij}), ijk\rangle)$
    If t'_{ijk}
     Then `Supported-by`$[(e_{i,j}, l_{ij}), ijk] \leftarrow t'_{ijk}$
       `Push`$(\langle e_{i,j}, l_{ij}, ijk\rangle,$ `Supports`$[(e_{i,k}, l'_{ik})])$
       `Push`$(\langle e_{i,j}, l_{ij}, ijk\rangle,$ `Supports`$[(e_{j,k}, l'_{jk})])$
     Else `Domain`$(e_{i,j}) \leftarrow$ `Domain`$(e_{i,j}) \setminus \{l_{ij}\}$
       `Push`$((e_{i,j}, l_{ij}), Q)$
       When `Domain`$(e_{i,j}) = \emptyset$
        Then Consistency $\leftarrow false$
Return $\{$`Domain`$(e_{i,j})\}$

Fig. 7. $\triangle$AC.

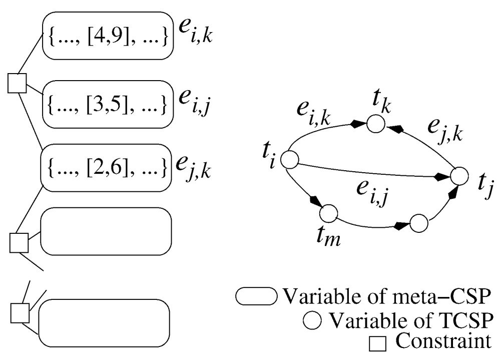

Fig. 8. Left: Meta-CSP with ternary constraints. Right: TCSP.

sake of achieving $\triangle$*AC*. Indeed, two time points that are not connected in the graph of the TCSP (e.g., t_m and t_j in Fig. 8) are considered to be linked by a universal constraint, which is an edge labeled by $\{(-\infty, \infty)\}$. Finally $(-\infty, \infty)$ is an absorbing element for the composition operation, and its intersection with any other interval is never empty. Consequently, the ternary constraint $\triangle[e_{i,j}, e_{i,m}, e_{j,m}]$ is necessarily $\triangle$AC, and none of the intervals in the labels of $e_{i,k}$, $e_{i,m}$, or $e_{k,m}$ can be removed. In particular, the label of $e_{j,m}$ remains $\{(-\infty, \infty)\}$.

$\triangle$AC is a generalized arc-consistency algorithm for the ternary constraints of the meta-CSP (Fig. 8, left). According to the definition of path-consistency in quantitative temporal networks given by Dechter in [4], $\triangle$AC can also be viewed as a *weak path-consistency* algorithm for the original TCSP (Fig. 8, right). It is weak in that does not modify the intervals labeling the edges of the TCSP. In the above example, this is illustrated by the fact that the interval in the label of $e_{j,m}$ is not tightened.

Of course, the existence of an edge between t_i and t_j and between t_i and t_m restrains the labeling of the edge between t_j and t_m. However, inferring this resulting constraint is beyond $\triangle$AC and can be achieved by path-consistency, which is a stronger property than $\triangle$AC.

The only consistency algorithms for temporal networks that we are aware of are `NPC-1` and `NPC-2` (numerical PC algorithms) of Dechter [4] and `ULT` (Upper-Lower Tightening algorithm) and `LPC` (Loose Path-Consistency) of Schwalb and Dechter [11]. These algorithms aim to achieve path consistency of the TCSP and may modify the intervals in the label of an edge. Given the disjunctive intervals of the labels of the TCSP, `NPC-1` and `NPC-2` may cause a fragmentation problem, which increases the number of intervals per label in the TCSP and increases the size of the resulting meta-CSP. Schwalb and Dechter introduced `ULT` and `LPC` to avoid this fragmentation problem [11]. These algorithms approximate full path-consistency of the TCSP.

Our approach differs from `NPC-1`, `NPC-2`, `ULT`, and `LPC` in that we do *not modify* any interval labeling an edge in the TCSP: we either keep the interval or remove it. We consider each interval as an atomic value in the domain of a variable of the meta-CSP. Our goal is to remove inconsistent individual intervals from the labels, not to tighten these intervals. Tightening the intervals may not terminate in the general case and may be prohibitively expensive in the integral case.

4. Experimental results

We conducted empirical evaluations on randomly generated TCSPs. Below we describe our random generator, the characteristics of the experiments we conducted, and our observations.

4.1. Random generator

We designed a generator of random TCSP instances that guarantees that the temporal network is connected and that a specified percentage of the generated instances is solvable. Our generator is designed as follows. It takes as input:

– The number of nodes n in the temporal network, which is the number of time points in the TCSP.
– The density d of temporal graph G. This determines the number of edges $|E|$ in the temporal graph. Naturally, $|E| \leqslant n(n-1)/2$.
– The maximal number of intervals in the label of an edge k. The actual number of intervals per label is chosen randomly in a uniform manner between 1 and k.
– The range of the nodes selected from $R = [1, r]$, with $r \in \mathbb{N}$.
– The percentage p_c of solvable problems.

We generate a random TCSP example according to the steps below:

1. We select values in R to correspond to positions of time points in this interval. We enforce that the first node of the graph has position 1, and the last node in the graph has position r. Then we select randomly $(n-2)$ distinct points within the given interval R, excluding the extremities of R.
2. We use the $n(n-1)/2$ combinations of two time points t_i and t_j (with $t_i < t_j$) generated above to generate a list L of edges $e_{i,j}$. Then we build the list E of edges by edges randomly selecting $|E|$ edges from L.
3. Measuring the distance $\delta = (t_j - t_i)$ for each edge $e_{i,j}$ in E, we label $e_{i,j}$ with a random number of intervals in $[1, k]$ while ensuring that there is at least one interval $[a, b]$ such that $\delta \in [a, b]$. This ensures that the resulting TCSP has a solution.
4. With probability $(1 - p_c)$, we swap the labels of two random edges in the graph.
5. Finally, we test the graph for connectivity and discard the unconnected graphs.

4.2. *Experiments conducted*

We tested △AC on the randomly generated connected problems of Table 1. Our generator guarantees that at least 80% of these problems have at least one solution. We average the results over 100 samples.

In order to demonstrate the filtering power of △AC, the comparison of the average size of the meta-CSP before and after filtering is shown in Fig. 9 for TCSP I and Fig. 10 for TCSP II. The numerical values reported in Table 2 and Table 3.

In order to demonstrate the advantages of △AC, we report the cost of finding all the solutions of the meta-CSP with and without this preprocessing. To solve the meta-CSP we use the basic chronological backtrack search described in [5]. This search process requires solving an STP at each node expansion. To this end, we use the Directional Path-Consistency algorithm DPC of Dechter [4]. In our experiments, DPC is significantly more efficient than the Floyd-Warshall algorithm in determining the consistency of the STP (although it does not necessarily yield the minimal STP) [13].

Figure 9 also shows the number of the solutions of the meta-CSP for TCSP I. We do not show this number in Fig. 10 because the meta-CSP corresponding to TCSP II is large and finding *all* its solutions is prohibitively expensive.

Table 1

Problems tested (100 samples per point)

	n	k	Density		$\|E\|$
			Range	Step	Range
TCSP I	8	[1, 5]	[0.02, 0.1]	0.02	[7, 9]
	8	[1, 5]	[0.2, 0.9]	0.10	[11, 26]
TCSP II	20	[1, 5]	[0.02, 0.1]	0.02	[22, 36]
	20	[1, 5]	[0.2, 0.9]	0.10	[53, 173]

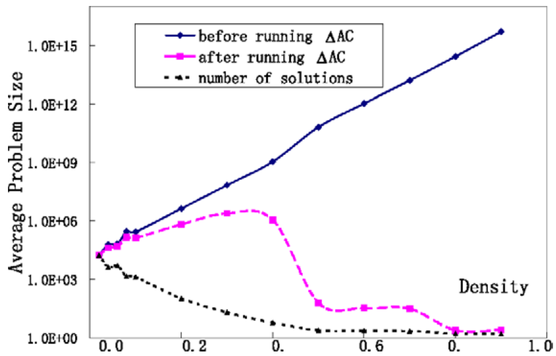

Fig. 9. Reduction of problem size of TCSP I.

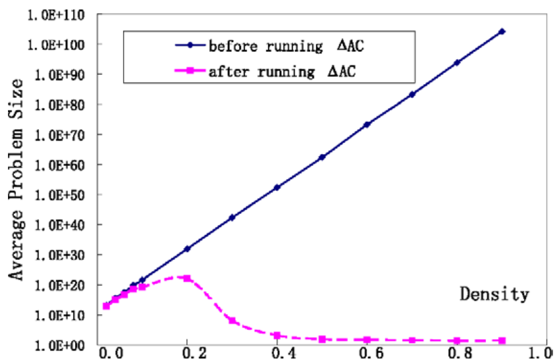

Fig. 10. Reduction of problem size of TCSP II.

Table 2

Performance of $\triangle$AC on TCSP I

Graph density	Number of variables in meta-CSP	Size of meta-CSP		Number of solutions	Cost of search without $\triangle$AC		Cost of search with $\triangle$AC		Cost of $\triangle$AC	
		Original	Filtered		CPU [s]	CC	CPU [s]	CC	CPU [s]	CC
0.02	7	16702	16701.67	16701.67	13.60	518463.7	13.62	518463.7	5.00E-04	0
0.04	8	58448	40831.72	4176.91	21.60	843112.7	17.86	712777.8	0.0011	55.53
0.06	8	64780	48399.24	4837.69	25.03	965354.3	22.02	868557.0	0.0012	50.98
0.08	9	282427	142638.28	1437.01	24.23	1008288.4	18.14	782634.6	0.0022	122.70
0.1	9	271254	132758.27	1331.86	26.08	1103695.6	17.83	793677.7	0.0017	134.14
0.2	11	4257366	653949.00	105.88	23.95	1105540.5	6.43	335393.7	0.0033	324.44
0.3	13	6.81E+07	2424326.70	20.02	16.32	866010.3	2.10	117963.1	0.0050	575.80
0.4	15	1.10E+09	1117395.50	5.97	22.13	1320010.5	0.49	29187.1	0.0075	880.23
0.5	18	6.64E+10	62.07	2.40	26.11	1630835.2	0.07	3654.7	0.0115	1383.80
0.6	20	1.06E+12	33.21	2.35	29.25	1932359.2	0.07	3821.0	0.0150	1711.11
0.7	22	1.61E+13	31.16	2.19	34.87	2297002.5	0.08	3607.9	0.0192	2059.18
0.8	24	2.74E+14	2.41	1.66	57.13	3946315.0	0.07	3226.7	0.0217	2393.20
0.9	26	5.23E+15	2.48	1.60	74.39	5128653.0	0.08	3851.7	0.0262	2839.48

Table 3

Performance of $\triangle$AC on TCSP II

Graph density	Number of variable in meta-CSP	Size of meta-CSP		Cost of $\triangle$AC	
		Original	Filtered	CPU [s]	CC
0.02	22	1.51E+13	9.31E+12	4.10E-03	86
0.04	26	4.16E+15	1.05E+15	0.006	253
0.06	29	2.97E+17	5.66E+16	0.008	362
0.08	33	7.27E+19	3.94E+18	0.011	558
0.1	36	4.45E+21	1.72E+19	0.014	811
0.2	53	7.86E+31	1.11E+22	0.036	2581
0.3	70	2.00E+42	1.48E+08	0.072	5268
0.4	87	2.23E+52	1545.05	0.114	8047
0.5	105	2.62E+62	79.69	0.168	11324
0.6	122	1.96E+73	60.20	0.254	15446
0.7	139	1.90E+83	37.11	0.332	20522
0.8	156	6.46E+93	23.55	0.433	26050
0.9	173	1.88E+104	24.60	0.554	33139

The results of solving the meta-CSP in terms of CPU time and constraint checks CC for TCSP I are shown in Figs 11 and 12, and the numerical values are reported in Table 2. In this table, we also report the cost of running $\triangle$AC, although it is already included in the cost of search in order to demonstrate that the overhead due to filtering is practically negligible.

4.3. Observations

The comparison of Figs 9 and 10 shows that the pruning power of $\triangle$AC increases with the density of the TCSP. It also shows that $\triangle$AC dramatically reduces the size of the meta-CSP especially when density is high. Further, Fig. 9 shows that the size of meta-CSP obtained after filtering by $\triangle$AC is close to the number of solutions for high-density networks. Both behaviors are typical of consistency filtering techniques used as a preprocessing step to search.

Figures 11 and 12 show the cost of finding all the solutions of the meta-CSP, with and without preprocessing with $\triangle$AC, in terms of constraint checks and CPU time, respectively. The figures show that preprocessing does not negatively affect the cost of search under low density and is tremendously effective in reducing the total cost under high density. Indeed, the cost of search is almost negligible when density is high.

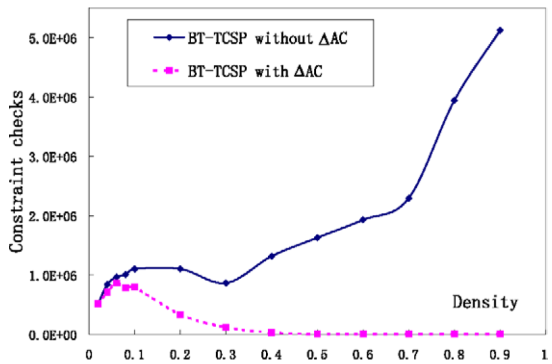

Fig. 11. Constraint checks for finding all solutions of the meta-CSP corresponding to TCSP I.

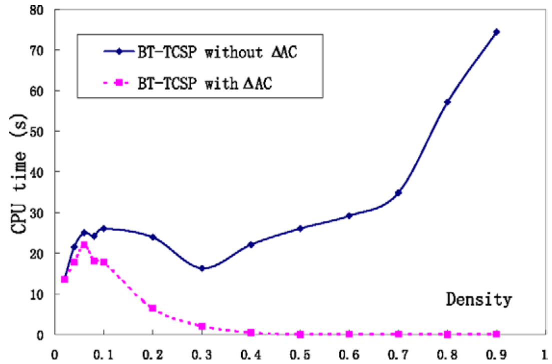

Fig. 12. CPU time for finding all solutions of the meta-CSP corresponding TCSP I.

In contrast, search without preprocessing with △AC is prohibitively expensive when density is high.

When density is low, the temporal graph has few edges; hence the meta-CSP has relatively few variables and its size is small. When density increases, the number of edges in the temporal graph, and hence the number of variables in the meta-CSP, increases exponentially. However, this increases the number of triangles in the temporal graph and enhances the filtering power of △AC, which removes most intervals. In all cases, the experiments strongly support using △AC when solving a TCSP.

Moreover, the use of △AC seems to uncover the existence of a phase transition around a density value of 0.09. The existence of a phase transition in solving the TCSP was already noted in [11,14] but deserves more thorough investigation.

5. Conclusions

From the experimental results reported in the previous section, we draw the following conclusions:

1. △AC can dramatically reduce the size of the meta-CSP, especially when density is high. Hence it helps to improve the performance of the backtrack search to solve the meta-CSP.
2. The cost of △AC is negligible compared with the cost of the search for solving the meta-CSP.

This establishes that the △AC is a cheap and effective consistency algorithm and should become part of any standard preprocessing technique for solving the TCSP.

One interesting direction for future research is to integrate △AC in a look-ahead strategy for solving the TCSP. Another research direction is to use △AC to improve the performance of the `ULT` and `LPC` algorithms of Schwalb and Dechter [11] since the two approaches are orthogonal and, to the best of our knowledge, the only efficient consistency filtering techniques for TCSPs reported in the literature.

Acknowledgements

This work is supported by a NASA-Nebraska grant and the CAREER Award #0133568 from the National Science Foundation. We are indebted to the anonymous reviewers for their comments, which helped us improve our presentation.

References

[1] C. Bessière, Arc-consistency and arc-consistency again, *Artificial Intelligence* **65**(1) (1994), 179–190.

[2] C. Bessière, E.C. Freuder and J.-C. Régin, Using constraint metaknowledge to reduce arc consistency computation, *Artificial Intelligence* **107**(1) (1999), 125–148.

[3] C. Bessière and J.-C. Régin, Refining the basic constraint propagation algorithm, in: *Proc. of the 17th IJCAI*, 2001, pp. 309–315.

[4] R. Dechter, *Constraint Processing*, Morgan Kaufmann, 2003.

[5] R. Dechter, I. Meiri and J. Pearl, Temporal constraint networks, *Artificial Intelligence* **49**(1–3) (1991), 61–95.

[6] Y. Deville and P. Van Hentenryck, An efficient arc consistency algorithm for a class of CSP problems, in: *Proc. of the 12th IJCAI*, 1991, pp. 325–330.

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

[8] A.K. Mackworth and E.C. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Artificial Intelligence* **25**(1) (1984), 65–74.

[9] R. Mohr and T.C. Henderson, Arc and path consistency revisited, *Artificial Intelligence* **28**(2) (1986), 225–233.

[10] R. Mohr and G. Masini, Good old discrete relaxation, in: *European Conference on Artificial Intelligence (ECAI-88)*, 1988, pp. 651–656.

[11] E. Schwalb and R. Dechter, Processing disjunctions in temporal constraint networks, *Artificial Intelligence* **93**(1–2) (1997), 29–61.

[12] D. Waltz, Understanding line drawings with scenes with shadows, in: *The Psychology of Computer Vision*, P.H. Winston, ed., McGraw-Hill, Inc., 1975, pp. 19–91.

[13] L. Xu and B.Y. Choueiry, A new efficient algorithm for solving the simple temporal problem, in: *10th International Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic (TIME-ICTL 03)*, IEEE Computer Society Press, 2003, pp. 212–222.

[14] L. Xu and B.Y. Choueiry, Improving backtrack search for solving the TCSP, in: *Principles and Practice of Constraint Programming (CP 03)*, LNCS 2833, Springer, 2003, pp. 754–768.

[15] Y. Zhang and R.H.C. Yap, Making AC-3 an optimal algorithm, in: *Proc. of the 17th IJCAI*, 2001, pp. 316–321.