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

Follow this and additional works at: http://digitalcommons.unl.edu/csearticles

Part of the Computer Sciences Commons
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, as 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-
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 $[a, b]$, and denotes a constraint of bounded difference $a \leq (t_j - t_i) \leq b$.

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 $I_{i,j} = \{l_{i,j}^1, l_{i,j}^2, \ldots, l_{i,j}^h\}$ 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 pairwise-disjoint, and that they are sorted in an increasing order of their end points. The superscript $k$ of interval $l_{i,j}^k$ 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 so that 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 $\text{Domain}(e_{i,j})$, is its label, $I_{i,j} = \{l_{i,j}^1, l_{i,j}^2, \ldots, l_{i,j}^h\}$. 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) $(e_{i,j}, l_{i,j}^k)$ 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 $O(n^3k^{|E|})$. 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 $\text{AC}$, to these ternary constraints (see Fig. 3). The complexity of $\text{AC}$ is $O(\text{degree}(G) \cdot |E| \cdot k^3) = O(n|E|k^3)$, where $\text{degree}(G)$ denotes the largest number of edges incident to any vertex in $G$. Again, the $\text{AC}$ 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 $\text{AC}$ 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 $\text{CC}$.

![Fig. 3. Left: Meta-CSP with a global constraint. Right: with ternary constraints.](image)

3. The filtering algorithm

We approximate the generalized arc-consistency of the meta-CSP by replacing the unique global constraint with a ternary constraint $\Delta[e_{i,j}, e_{i,k}, e_{j,k}]$ 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 $l_{i,k} = l_{i,k}^{l} \cap l_{i,k}^{r}$ of the intervals $l_{ik}^{l} = [a, b]$ and $l_{ik}^{r} = [c, d]$ labeling the respective $e_{i,k}^{l}$ and $e_{i,k}^{r}$ 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 $\text{AC}$ 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 $\Delta[e_{i,j}, e_{i,k}, e_{j,k}]$. Given three variable-value pairs $(e_{i,j}, t_{ij})$, $(e_{i,k}, t_{ik})$, and $(e_{j,k}, t_{jk})$ of the meta-CSP, we say that the labeled triangle $\Delta[(e_{i,j}, t_{ij}), (e_{i,k}, t_{ik}), (e_{j,k}, t_{jk})]$ is a consistent triangle if and only if $(l_{ij} \circ l_{jk}) \cap l_{ik} \neq \emptyset$. Figure 4 shows a consistent triangle $\Delta[(e_{i,j}, [3, 5]), (e_{i,k}, [4, 9]), (e_{j,k}, [2, 6])]$. 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 $\Delta[e_{i,j}, e_{i,k}, e_{j,k}]$ is $\triangle AC$ relative to the meta-CSP variable $e_{i,j}$ if and only if for every interval $l_{ij}^{t}$ in $\text{Domain}(e_{i,j})$ there exists an interval $l_{ik}^{t}$ in $\text{Domain}(e_{i,k})$ and an interval $l_{jk}^{t}$ in $\text{Domain}(e_{j,k})$ such that $(l_{ij}^{t} \circ l_{jk}^{t}) \cap l_{ik}^{t} \neq \emptyset$.

![Fig. 4. A consistent triangle.](image)
2. The ternary constraint $\triangle(e_{ij}, e_{ik}, e_{jk})$ is $\triangledown AC$ if and only if it is $\triangle AC$ relative to the variables $e_{ij}, e_{ik},$ and $e_{jk}$.

3. Finally, the meta-CSP is $\triangledown AC$ if and only if all its ternary constraints are $\triangledown AC$.

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 $O(\text{degree}(G) \cdot |E|) = O(n|E|)$, and the size of each constraint is at most $k^3$.

3.2. $\triangledown AC$ algorithm

The $\triangledown AC$ algorithm, shown in Fig. 7, removes the intervals in the domain of an $e_{ij}$ that do not have a support in any triangle in which $e_{ij}$ 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 \texttt{First-support} of Fig. 5 and \texttt{Initialize-support} of Fig. 6. The \texttt{Push} and \texttt{Delete} operations we use are destructive stack operations.

$\triangledown AC$ operates by looking at every combination of a vvp $(e_{ij}, l_{ij})$ and the triangles $ijk$ in which it appears, denoted $\langle (e_{ij}, l_{ij}), ijk \rangle$. The support of $\langle (e_{ij}, l_{ij}), ijk \rangle$ is the first element in the domains of $e_{ik}$ and $e_{jk}$ 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. $\triangledown AC$

tries to find the next acceptable support. The process is repeated until all vvps have a valid support in every relevant triangle.

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

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

$\texttt{Initialize-support}$ 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. $\triangledown AC$, shown in Fig. 7, iterates over the vvps that have been deleted and retracts them from supporting entries in \texttt{Supported-by}.

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

3.3. Discussion

Note that we do \textit{not} add any edges to the network of the TCSP to triangulate it or to make it a complete graph. \textit{Such an effort would be useless for the
\textit{sake of achieving $\Delta AC$}. 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 $\Delta [e_{ij}, e_{ij}^*, e_{ij}^*]$ is necessarily $\Delta AC$, and none of the intervals in the labels of $e_{ik}$, $e_{im}$, or $e_{km}$ can be removed. In particular, the label of $e_{jm}$ remains $[(-\infty, \infty)]$.

$\Delta AC$ 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], $\Delta AC$ 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 $e_{jm}$ 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 $\Delta AC$ and can be achieved by path-consistency, which is a stronger property than $\Delta AC$.

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 Schwab 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. Schwab 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, $|E| \leq n(n-1)/2$.
– 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 $\triangle AC$ 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 $\triangle AC$, 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 $\triangle AC$, 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

<table>
<thead>
<tr>
<th>Problems tested (100 samples per point)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$ $k$ Density $</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>TCSP I 8 1, 5 0.02 0.1 0.02 [7, 9]</td>
</tr>
<tr>
<td>8 1, 5 0.2 0.9 0.10 [11, 26]</td>
</tr>
<tr>
<td>TCSP II 20 1, 5 0.02 0.1 0.02 [22, 36]</td>
</tr>
<tr>
<td>20 1, 5 0.2 0.9 0.10 [53, 173]</td>
</tr>
</tbody>
</table>

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

Fig. 10. Reduction of problem size of TCSP II.
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 \( \Delta AC \), 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 \( \Delta AC \) increases with the density of the TCSP. It also shows that \( \Delta AC \) 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 \( \Delta AC \) 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 \( \Delta AC \), 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.
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 backtracking 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


