Improving Backtrack Search for Solving the TCSP

Lin Xu and Berthe Y. Choueiry

Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln University of Nebraska-Lincoln, Lincoln NE
{lxu | choueiry}@cse.unl.edu

1 Abstract

In this paper, we address the task of finding the minimal network of a Temporal Constraint Satisfaction Problem (TCSP). We report the integration of three approaches to improve the performance of the exponential-time backtrack search (BT-TCSP) proposed by Dechter et al. [6] for this purpose. The first approach consists of using a new efficient algorithm ($\triangle$STP) [21] for solving the Simple Temporal Problem (STP), an operation that must be executed at each node expansion during BT-TCSP. The second approach improves BT-TCSP itself by exploiting the topology of the temporal network. This is accomplished in three ways: finding and exploiting articulation points (AP), checking the graph for new cycles (NewCyc), and using a new heuristic for edge ordering (EdgeOrd). The third approach is a filtering algorithm, $\triangle$AC, which is used as a preprocessing step to BT-TCSP, and which significantly reduces the size of the TCSP [22]. In addition to introducing two new techniques, NewCyc and EdgeOrd, this paper discusses an extensive evaluation of the merits of the above three approaches. Our experiments on randomly generated problems demonstrate significant improvements in the number of nodes visited, constraint checks, and CPU time.

2 Background and Motivation

A Simple Temporal Problem (STP) is defined by a graph $G = (V, E, I)$ where V is a set of vertices i representing time points p_i; E is a set of directed edges $e_{i,j}$ representing constraints between two time points p_i and p_j; and I is a set of constraint labels for the edges, see Fig. 1 (left). A constraint label $I_{i,j}$ of edge $e_{i,j}$ is an interval $[a, b]$, $a, b \in \mathbb{R}$,

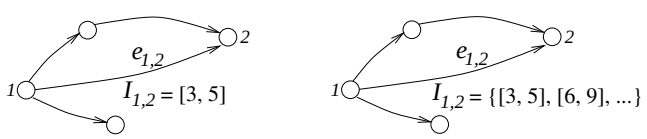


Fig. 1. *Left:* STP. *Right:* TCSP.

and denotes a constraint of bounded difference $a \leq (p_j - p_i) \leq b$. Note that $I_{i,j} = [a, b]$ $\Leftrightarrow I_{j,i} = [-b, -a]$. 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 i and j, see Fig. 1 (right). We assume that the intervals in a label are disjoint and ordered in a canonical way. The following is a typical example:

> *Tom has class at 8:00 a.m. He can either make breakfast for himself (10-15 minutes), or get something to eat from a local store (less than 5 minutes). After breakfast (5-10 minutes), he goes to school either by car (20-30 minutes) or by bus (at least 45 minutes). Today, Tom gets up between 7:30 a.m. and 7:40 a.m.*

We wish to answer queries such as: "Can Tom arrive at school in time for class?", "Is it possible for Tom to take the bus?", "If Tom wanted to save money by making breakfast for himself and taking the bus, when should he get up?", and so on. This temporal problem can be represented as a temporal graph.

Let p_0 be a reference time-point (e.g., 6:00 am), p_1 the time point Tom gets up, p_2 the time point he starts his breakfast, p_3 the time point he finishes it, and p_4 the time point he arrives at the school. Fig. 2 shows the temporal graph of this TCSP.

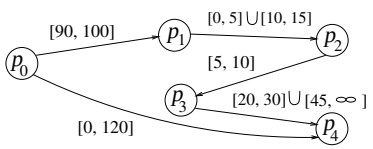


Fig. 2. *A TCSP example.*

Dechter [5] described a backtrack search procedure (BT-TCSP) for solving a TCSP, which is an NP-hard problem. To this end, the TCSP is expressed as a 'meta' Constraint Satisfaction Problem (meta-CSP). The variables of the meta-CSP are the edges $e_{i,j}$ of G. Their number depends on the density of the temporal graph. The domain of a variable $e_{i,j}$ is its label, $I_{i,j} = \{l_{ij}^{(1)}, l_{ij}^{(2)}, \ldots, l_{ij}^{(k)}\}$. A partial solution is a set $\{(e_{ij}, l_{ij}^{(h)})\}$ of variable-value pairs (vvps) that form a consistent STP, *which is a global constraint*. A complete solution is a consistent STP in which all the edges of G appear. The minimal network of the TCSP is the union of *all* complete solutions. Each node in the tree generated by BT-TCSP is an STP P' that has E' edges, a subset of the edges of the original network ($E' \subseteq E$), each labeled with a unique interval from its domain. When P' is consistent, the node is expanded by adding to P' an edge from $(E - E')$ labeled with an interval from its domain. This yields a new STP that is checked again for consistency. Fig. 3 illustrates the tree corresponding to the example of Fig. 2, where edges are considered in their lexicographical order.

In this paper, we combine the following techniques to improve the performance of BT-TCSP, and demonstrate their effectiveness on randomly generated problems:

1. Every node in the tree is an STP that needs to be solved before the search can proceed. Hence, the performance of a TCSP solver depends critically on that of the STP solver. We compare for the first time the performance of various known STP solvers, including a new one, $\triangle$`STP`, that we proposed in [21]. We show that it

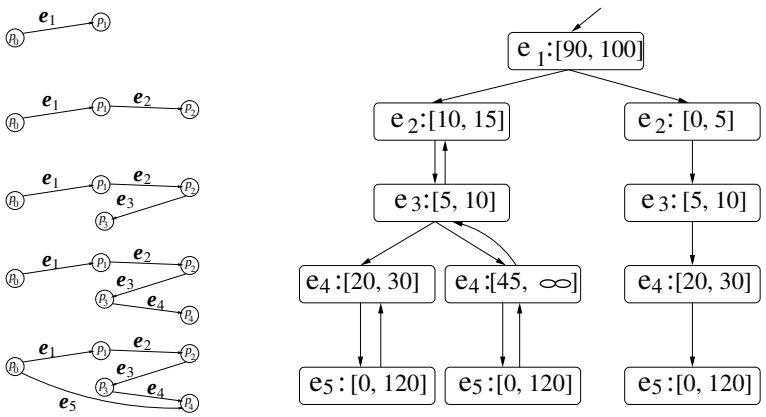


Fig. 3. *The search tree for the example of Fig. 2.*

outperforms all others. Note that the performance of the STP solver does not affect the number of nodes visited in BT-TCSP.

2. One well-known technique to improve the performance of a CSP is to decompose it into sub-problems using its articulation points [9, 11, 6], and to solve the sub-problems independently. We provide for the first time an empirical evaluation of the effectiveness of this technique.
3. Further exploiting the topology of the temporal network, we show how to avoid running an STP-solver by checking for the existence of new cycles (NewCyc) in the network as edges are added along a given path in the tree. In the example of Fig. 3, the first four consistency checks are unnecessary because there are no cycles in the respective networks and the corresponding STPs are always consistent.
4. Another way to improve the performance of BT-TCSP is to find a good variable-ordering heuristic for the search. This corresponds to a sequencing of E, the edges of G, as they are added along a given path in the tree. A good sequence reduces unnecessary backtracking and also the number of constraint checks. We introduce a new ordering heuristic (EdgeOrd) that exploits the adjacency of existing triangles in the graph to determine the ordering of their edges in the tree.
5. We reduce the domains of the variables of the meta-CSP by using the efficient filtering algorithm, $\triangle$AC, described in [22].

The contributions of this paper can be summarized as follows:

1. A new technique for saving constraint checks (NewCyc) and a new ordering heuristic (EdgeOrd).
2. The combination of the above listed techniques (i.e., an STP-solver, AP, NewCyc, EdgeOrd, and $\triangle$AC) to find all the solutions of the TCSP.
3. Empirical evaluation and analysis of the effectiveness of these techniques and their combinations to demonstrate their significance.

This paper is structured as follows. Section 3 reviews the STP-solvers used. Section 4 discusses the three improvements exploiting the topology of the temporal network. Section 5 summarizes a filtering algorithm used as a preprocessing step. Section 6 describes our experiments and observations. Finally, Section 7 concludes this paper.

3 Algorithms for Solving the STP

TCSP is **NP**-hard and is solved with backtrack search. Every node expansion in the search tree needs to check the consistency of an STP. Thus a good STP solver is critical for solving the TCSP. We test the following STP solvers: Directed Path Consistency DPC [7], Partial Path Consistency PPC [2], and Triangle-STP $\triangle$STP [21].

3.1 Solving the STP Using Directional Path Consistency (DPC)

A basic algorithm to solve an STP is the Floyd-Warshall algorithm (F−W), which computes all-pairs shortest-paths in a distance graph [4]. F−W guarantees consistency, minimality, and decomposability and has a complexity of $\Theta(n^3)$. Montanari showed that F−W is a special case of the Path Consistency (PC) algorithm [15]. Dechter et al. propose the Directed-Path Consistency (DPC) algorithm [7]. This algorithm is never more costly than F−W, runs in $O(n^3)$, and can determine the consistency of an STP in $O(n(W^*(d))^2)$, where $W^*(d)$ is the induced width of the graph along a given ordering d. DPC determines the consistency of the STP, but does not necessarily yield the minimal and decomposable network. Since only the consistency of an STP matters during BT-TCSP, we use DPC instead of F−W because of its lower cost.

3.2 Solving the STP Using Partial Path Consistency (PPC)

Bliek and Sam-Haroud introduced Partial Path-Consistency (PPC), an algorithm applicable to general CSPs (and not restricted to temporal networks) [2]. PPC works on a triangulated graph, unlike the PC algorithm which requires a complete graph. Further, Bliek and Sam-Haroud showed that when the constraints are *convex*, the PC algorithm (operating on the complete graph) and the PPC algorithm (operating on the triangulated graph) yield equivalent results: the same labeling for the edges common to both graphs and the minimality and decomposability of the STP. PPC never requires more constraint checks than PC, which is advantageous when the (triangulated) graph is sparse. This is particularly attractive in BT-TCSP, which requires solving an STP at each node.

PPC requires that the graph be triangulated, which may result in new edges being added to the graph. We triangulate the temporal network using the algorithm devised in [17]. We represent the new edges as universal constraints in the original constraint graph and set their label to $(-\infty, \infty)$.

In the tree generated by BT-TCSP, each node represents an STP whose graph adds exactly one edge to the graph of the parent of the node (and must be triangulated to be used by PPC). Assuming a static ordering in the tree, the total number of graphs that appear along any given complete path is exactly equal to the number of edges in the original problem. Further, all nodes at a given level of the search tree have the same graph (only the edge labelings may vary). Thus, under static ordering, the number of possible graphs considered during the BT-TCSP process is exactly equal to the total number of edges in the temporal network.

We test two methods for accessing the triangulations of the STPs given a static variable ordering, Fig. 4. In the first method, *Plan A*, we pre-compute all the STPs needed in search, triangulate them, and store their triangulations for use during search.

```
All-triangulated-subgraphs (G_0, d)
G ← nil
TriSubGs ← nil
E_0 ← all edges in G_0 using ordering d
For e ∈ E_0 do
    Push(e, G)
    Push(Triangulate(G), TriSubGs)
Return Reverse(TriSubGs)
```

```
Induced-subgraphs (G_t, G_i)
E_t ← edges of G_t
N_i ← all nodes ∈ G_i
E_i ← nil
Forall e_{i,j} ∈ E_t
        When i ∈ N_i and j ∈ N_i
          Then Push(e_{i,j}, E_i)
Return E_i
```

Fig. 4. *Left:* List of triangulated subgraphs given an ordering. *Right:* Inducing a subgraph from the triangulated original graph.

In the second method, *Plan B*, we triangulate the entire network only once. We then induce from the triangulated graph the subgraph whose vertices form the STP under consideration. Since the original graph is triangulated, each induced subgraph is also triangulated.

- *Plan A*: Given a variable ordering d, the list of the graphs considered during BT-TCSP is generated as shown in Fig. 4 (left). `Push` adds an item to a list, `Reverse` reverses a list, and **Triangulate** triangulates a graph. We use the ith element of `TriSubGs` list as the triangulated subgraph for the node at the ith level of the tree.
- *Plan B:* Here we compute the triangulated graph only once and induce from it the subgraph needed at every step. Fig. 4 (right) shows the algorithm where G_t is the triangulated graph of the original network and G_i is the subgraph considered at level $1 \leq i \leq |E|$ in the search. Note that this technique may end up considering denser graphs than necessary, which increases the cost of solving the STP.

Our experimental results show that *Plan A* always outperforms *Plan B* in terms of the number of constraint checks and CPU time. Note that neither of these two plans affects the number of backtracks (the number of nodes visited) in BT-TCSP.

3.3 $\triangle$`STP` Algorithm Used with TCSP Algorithm

$\triangle$`STP` algorithm yields the same minimal network as `F-W` and `PPC`. It uses the idea of triangulation and considers the temporal graph as composed of triangles instead of edges. Constraint propagation is 'triangle-based' rather than 'edge-based.' As a finer version of `PPC`, $\triangle$`STP` can find the minimal network with less cost than `F-W` and `PPC`. When density is low, $\triangle$`STP` is even cheaper than `DPC`, which does not guarantee the minimal network. Similar to `PPC`, the pre-requisite condition for $\triangle$`STP` is to first triangulate the temporal graph. We have introduced two plans to obtain triangulated subgraphs in the previous subsection. We will use *Plan A* for its lower cost in practice.

When solving a TCSP with search, the STP examined at each node in the search tree is a subgraph of the original TCSP. Since the STPs we need to check always have lower density than the original TCSP, the outstanding performance of $\triangle$`STP` under low density makes it even more attractive to use for solving the TCSP.

4 Exploiting the Topology of the Constraint Network

We propose three topology-based techniques to enhance the performance of search. While the first technique is applied *prior* to search to decompose the problem into independent components, the last two are intertwined with the search process.

4.1 Decomposition Using Articulation Points

The existence of articulation points in the graph of the temporal network can be used to decompose the network into its biconnected components, which can be solved independently. Finding the articulation points can be done in $O(|E|)$ [4]. This method provides an upper bound to the search effort in the size of the largest biconnected component [11]. It can effectively reduce the number of constraint checks in BT-TCSP and the number of nodes visited in its tree. A solution to the entire network is a combination of any of the solutions of the biconnected components. The total number of solutions is: $S = \prod_{i=1}^{n} s_i$, where s_i is the number of solutions for component i. This conjunctive decomposition of the temporal network [12] allows us to solve the sub-problems in parallel, as in a multi-agent system. Articulation points usually appear only when the density is low or when the TCSP has a special topology. Note that even in the absence of articulation points, we could 'induce' such decompositions by removing some edges of the graph, in a manner similar to the cycle-cutset method of Dechter and Pearl [8]. We have implemented the mechanism for finding and using existing articulation points but not yet explored how to induce their existence.

4.2 New Cycle Check

The inconsistency of an STP is detected by the existence of a negative cycle in its distance graph. When the graph of an STP has no cycles, the STP is necessarily consistent.

Proposition 1. *A tree-structured constraint network is necessarily globally consistent.*

Note that is a stronger result than using the tree-structure of the constraint graph, which requires ensuring 2-consistency [10]. In BT-TCSP, nodes are expanded by adding one edge at a time. When the addition of a new edge does not yield a new cycle in the graph, a consistent STP remains consistent regardless of the labeling chosen for the new edge. We exploit this observation to save unnecessary consistency checks.

Corollary 1. *When the addition of an edge to a globally consistent STP yields no new cycles, the resulting STP is globally consistent.*

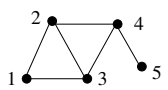


Fig. 5. *Simple constraint graph.*

Consider the example of Fig. 5. Suppose that search adopts the following ordering of the edges: $e_{1,2}$, $e_{2,3}$, $e_{1,3}$, $e_{3,4}$, $e_{2,4}$, and $e_{4,5}$. Fig. 6 shows the configurations of the STPs checked for consistency at each level in the search.

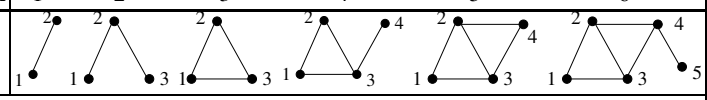
Search level	1	2	3	4	5	6	
STP							
Checking strategy							**Total**
Always	√	√	√	√	√	√	6
NewCyc	×	×	√	×	√	×	2

Fig. 6. *Comparison of STP checks using the new-cycle check heuristic.*

Along a given path, as the tree generated by search is being explored in a depth-first manner, two strategies can be adopted at a given level: (1) Always check the STP for consistency, and (2) check the consistency of the STP only when a new cycle has been added to the network. At levels 1 and 2, no cycles exist in the graph, and the STP is necessarily consistent, Fig. 6. At levels 4 and 6, no new cycles have been added to the graph of levels 3 and 5 respectively, and the corresponding STPs remain necessarily consistent regardless of their labeling. As illustrated above, checking for new cycles saves us unnecessary operations. Further, when the addition of a new edge yields a new cycle, two biconnected components of the previous level are necessarily merged into a new biconnected component at the current level. We need to check *only* the consistency of the newly formed biconnected component, and we can safely ignore the rest of the temporal network. This allows us to localize the effort of consistency checking to the necessary part of the network.

Corollary 2. *When the addition of an edge to a globally consistent STP yields a new cycle, the resulting STP is globally consistent if and only if the newly formed biconnected component is a consistent STP.*

The application of this new heuristic, NewCyc, significantly enhances the performance of solving the meta-CSP with search. To apply it, we need to identify, between two levels of the search tree, (1) that a new cycle has been introduced and (2) the two biconnected components that were merged as a result. This is done by running the $O(|E|)$ algorithm for finding articulation points at each level, checking whether the number of biconnected components was reduced between two levels, and identifying the component to be checked as that containing the new edge.

4.3 Ordering Heuristic for the Meta-CSP

Variable ordering is an effective heuristic for improving the performance of search. In general, it is governed by the 'fail first principle.' The shallower the node pruned in the tree, the larger the pruned subtree, and the larger the cost savings. For the meta-CSP, a node is pruned when it corresponds to an inconsistent STP. Thus, the ordering of the

edges (which are the variables of the meta-CSP) affects how quickly an inconsistent STP is found and also the effectiveness of constraint propagation in the STP.

As stated in Corollary 1, along a given path, no inconsistency may occur between one level and the next unless at least one new cycle is formed in the temporal graph. Consequently, a reasonable ordering heuristic is to first consider those edges that form triangles with edges existing in the STP. This may allow us to uncover inconsistencies as early as possible. It also increases the effectiveness of backtracking, because it is more likely to undo an inconsistency by changing the labeling of an edge in the same triangle as the one that yielded the inconsistency than that of a random edge. Our new edge-ordering heuristic orders the edges of the temporal graph in such a way that the network is expanded triangle by triangle 'around' the existing edges. The algorithm, given in Fig. 7, returns the list of edges in the order to be used by the search. It uses basic operations on lists. `Append` concatenates two lists in the order provided. `Pop` removes and returns the first item in a list. It requires that each edge be associated with

EdgeOrd (G)
$E_0 \leftarrow$ all edges of G
$E \leftarrow$ `nil`
While E_0 **do**
  $e_{i,j} \leftarrow$ Edge of E_0 appearing in the largest number of triangles in E_0
  $E \leftarrow$ `Append` $(E, \{e_{i,j}\})$
  $Q \leftarrow$ `nil`
  While $e_{i,j}$ **do**
    Forall k such that ijk is a subgraph of G **do**
      $Q \leftarrow$ `Append` $(Q, \{e_{i,k}, e_{j,k}\})$, $E \leftarrow$ `Append` $(E, \{e_{i,k}, e_{j,k}\})$
      $E_0 \leftarrow E_0 \setminus \{e_{i,j}, e_{i,k}, e_{j,k}\}$, $e_{i,j} \leftarrow$ `Pop`(Q)
Return E

Fig. 7. *Edge ordering heuristic.*

the number of triangles in which it appears in G, which is bounded by $(n-1)$, where n is the number of nodes in G (i.e., the time points). We obtain these numbers as a by-product of the implementation of the triangulation algorithm.

Based on the topology of the network, we choose the edge that participates in the largest number of triangles and schedule the edges of those triangles for a priority instantiation during the search. Fig. 8 illustrates the first steps of the application of the algorithm starting from edge I. First, the triangles in which edge I participates are explored. From there, we reapply iteratively the same process to each of the edges explored, i.e. edges II, III, and IV, gradually covering all the edges in the biconnected component. The modification of the label of any these edges propagates through these triangles. Thus, inconsistencies and deadends are likely to be more quickly detected during search, and backtrack remains locally contained.

We can show that this process stops when all the edges in the biconnected component have been visited. Then EdgeOrd restarts from an unvisited edge from the original graph and repeats the process until all edges of the original network have been visited.

The function returns a list in which the edges that are in a given biconnected component appear in sequence. As a result, this ordering heuristic implicitly enables search to

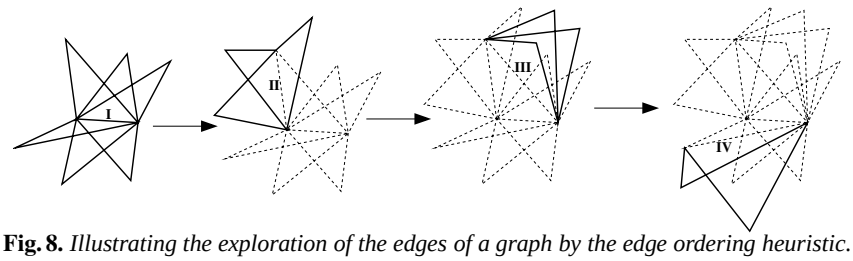


Fig. 8. *Illustrating the exploration of the edges of a graph by the edge ordering heuristic.*

examine the biconnected components of the graph in isolation, and thus decompose the graph automatically. The advantages of this mechanism are:

1. *Localized backtracking*: Neighboring levels in the search tree are likely to correspond to edges that form a triangle and thus are topologically related. When it encounters a deadend, search will backtrack to an edge that is more likely the culprit than another edge taken randomly from the graph.
2. *Automatic decomposition of the graph into its biconnected components*: This ordering heuristic implicitly guarantees that articulation points in the graph (if any), are exploited, as if the network was decomposed into its biconnected components without using the special algorithm necessary for this purpose (see Section 4.1).

We believe, but still need to show, that these features make EdgeOrd a more effective heuristic than a dynamic variable-ordering heuristic based only on domain size.

5 $\triangle$Arc-Consistency

When solving a CSP, it is common to run a domain filtering mechanism (such as arc-consistency, AC) as a preprocessing step to search, and to interleave search with a lookahead strategy (such as forward-checking, FC [13]). Consistency checking may reduce the domain of the variables, thus reducing the size of the CSP and the search effort.

The size of the meta-CSP is exponential in the size of the TCSP. If k is the number of intervals in the label of an edge in the TCSP, $|E|$ is the number of edges, and n is the number of nodes where $|E| \leq \frac{n(n-1)}{2}$, the size of the meta-CSP is in $O(k^{|E|})$. Thus it is important to explore mechanisms to reduce the size of the meta-CSP by removing 'inconsistent' intervals from the edge labels. The only constraint in the meta-CSP is a *global* constraint that requires all variable-value pairs of the meta-CSP to form a consistent STP. Thus, for the meta-CSP, AC is the generalized arc-consistency of this unique constraint, which is NP-hard [22]. In [22], we introduce the concept of $\triangle$Arc-Consistency as an approximation of the generalized arc-consistency of the meta-CSP. We also introduce an efficient algorithm, $\triangle$AC, that implements $\triangle$Arc-Consistency. This algorithm ensures that for every interval $l_{ij}^{(x)}$ in the domain of a meta-CSP variable $e_{i,j}$ there exist an interval $l_{ik}^{(y)}$ in the domain of the meta-CSP variable $e_{i,k}$ and an

interval $l_{kj}^{(z)}$ in the domain of the meta-CSP variable $e_{k,j}$ such that $l_{ij}^{(x)} \cap (l_{ik}^{(y)} \circ l_{kj}^{(z)}) \neq \emptyset$, where $\cap$ is interval intersection and $\circ$ is interval composition [5]. We establish that the complexity of $\triangle$AC is $O(degree(G) \times |E| \times k^3) = O(n|E|k^3)$. The value of $\triangle$AC lies in the data structures it uses, reminiscent of AC-4 [14] and AC-2001 [1], to save significantly the number of constraint checks[1]. We have not yet used $\triangle$AC in a lookahead strategy, but plan to do so in the future.

6 Experimental Results

Fig. 9 shows the TCSP solvers we tested, with and without pre-processing by $\triangle$AC.

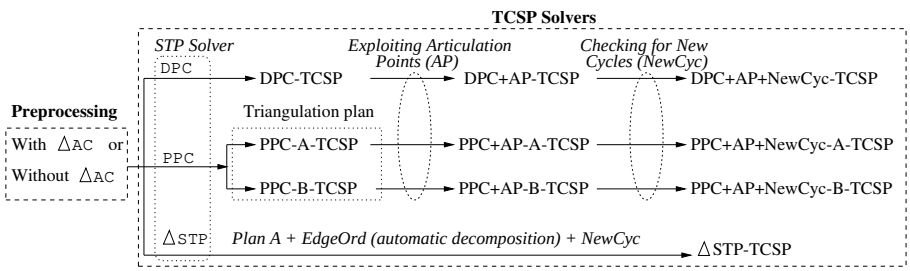


Fig. 9. *TCSP solvers tested.*

The STP solvers we used are DPC, PPC, and $\triangle$STP all as described in Section 3. The network is triangulated only prior to PPC and $\triangle$STP. We combined these STP solvers with the techniques proposed in Section 4 (i.e., AP, NewCyc, and EdgeOrd). Since we have not yet implemented a lookahead strategy, all the TCSP solvers tested use a static variable ordering. By default, and except for $\triangle$STP-TCSP (where we use EdgeOrd), it is a lexicographical ordering of the lexicographically sorted tuples naming the edges by their two endpoints. We compared the performance of the TCSP solvers in terms of the number of nodes visited NV, constraint checks CC, and CPU time. Since all CPU time curves have almost exactly the same shapes as the CC curves, *they are omitted to save space but are all available upon request*. We carried out our tests on randomly generated, (guaranteed) connected problems. Our generator, described in [22], guarantees that at least 80% of these problems have at least one solution. The TCSP instances generated have the following characteristics: $n = 8$, k randomly chosen between 1 and 5, density of the temporal network ($d = \frac{|E|-|E_{min}|}{|E_{max}|-|E_{min}|}$) varies in [0.02, 0.1] with a step of 0.02 and in [0.2, 0.9] with a step of 0.1. The number of variables in the meta-CSP, for which we find *all solutions*, varies from 7 to 26. The size of the meta-CSP varies on average between 1.6×10^5 and 5.2×10^{15}. We averaged the results of over 100 samples. The goal of our experiments was to study the *effects* on the various solvers of the improvements we proposed [2] (i.e., $\triangle$STP, AP, NewCyc, EdgeOrd, $\triangle$AC), and to establish their effectiveness. An extensive comparison of the the performance of the various STP solvers can be found in [21].

[1] We are investigating an improvement that may establish its optimality.

[2] Note that although decomposition according to articulation points is a well-known technique, to the best of our knowledge, it has not been yet assessed experimentally.

Section 6.1 discusses the number of solutions of the problems tested. Naturally, all solvers find the same solutions. Section 6.2 shows the effect of our techniques on the shape of the tree by measuring the number of nodes visited. Section 6.3 shows the effect of our techniques on the various TCSP solvers (i.e., DPC, PPC, and $\triangle$STP) on the number of constraint checks. In Sections 6.2 and 6.3 we also show how filtering the meta-CSP with $\triangle$AC dramatically improves the performance of search. The effect of this preprocessing is clearly visible in comparisons of the scale of the vertical axis of the charts without and after preprocessing. While the benefits of this filtering algorithm are discussed in [22], we confirm here that it is useful in every TCSP solver we tested.

6.1 Solutions to the TCSP

When density is low, there are few constraints, any partial solution is likely to be extended to a global solution, and there are many solutions to the meta-CSP as is seen in Fig. 10. Indeed, under low density, the temporal network (which is guaranteed con-

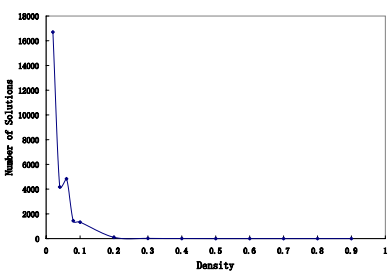


Fig. 10. *The number of solutions of the meta-CSP.*

nected by construction) has almost no cycles. Thus, almost any combination of intervals in the label of the edges is a solution to the meta-CSP (see Proposition 1). The number of solutions quickly drops as density rises. When d=0.9, there are only one or two solutions, one of which is guaranteed by construction.

6.2 Effects on the Size of the Search Tree

The effects of AP and EdgeOrd on the 'shape' of the tree can be assessed by the number of nodes visited NV by search. They are shown in Fig. 11.

Note that the effects of NewCyc on the various STP solvers (i.e., DPC, PPC, and $\triangle$STP) are irrelevant to this measurement. Indeed, they aim at reducing the cost of checking the consistency of the STP at a node in the tree once search has effectively reached the node. The '$*$' in the legend of Fig. 11 indicates that these results hold for all STP solvers tested. Fig. 11 shows that AP reduces significantly NV when density is low. When density is high, almost no articulation point exists, hence AP does not impact NV. The effect of EdgeOrd is quite dramatic across all values for density because it allows BT-TCSP to quickly identify dead-ends, as a good ordering heuristic is supposed to do. Moreover, and thanks to $\triangle$AC, we start to notice the existence of a phase transition that appears around $d = 0.1$ and becomes increasingly visible as we move toward more effective TCSP solvers.

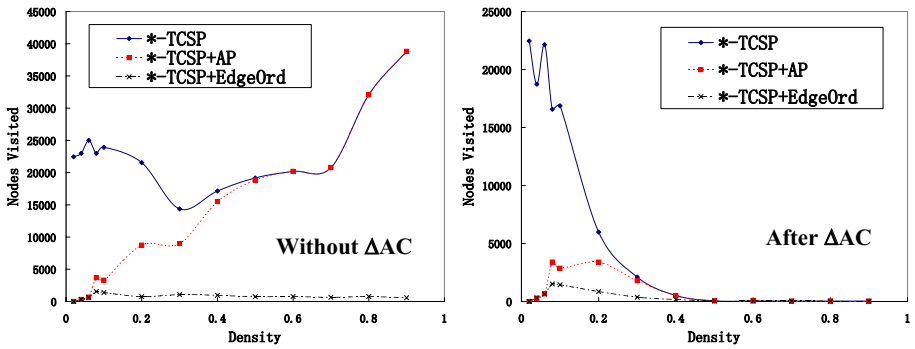


Fig. 11. *Nodes visited by BT-TCSP. Left: without preprocessing. Right: after filtering with △AC.*

6.3 Effects on the Number of Constraints Checks (Same as CPU Time)

Here we discuss the effects of our techniques on the various TCSP solvers: DPC, PPC, and △STP. We show the benefits of AP and NewCyc on DPC (Fig. 12). We show the benefits of AP, NewCyc on PPC for both *Plan A* (Fig. 13) and *Plan B* (Fig. 14) Finally, we show the benefits of EdgeOrd and NewCyc under *Plan A* on △STP (Fig. 15).

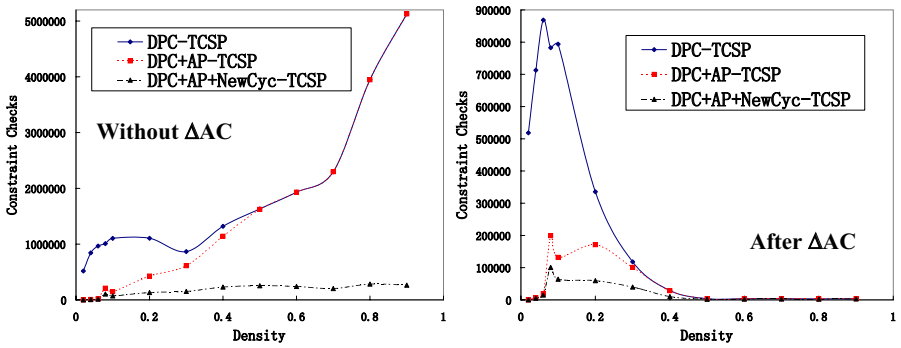


Fig. 12. *Constraint checks for* DPC-TCSP.

Exploiting Articulation Points: For DPC (Fig 12) and PPC (Fig. 13 and 14), AP is again particularly effective for low density graphs but useless for high density ones.

New Cycle Check: NewCyc dramatically reduces CC across all density values (even though it has no effect on the number of nodes visited, as stated in Section 6.2).

Triangulation Plans: The triangulation of an STP during search, required for PPC solver, is carried out according to *Plan A* (Fig. 13) and *Plan B* (Fig. 14) of Section 3.2. By comparing the scale of the vertical axis of these two figures, we conclude that *Plan A* is superior to *Plan B*. This can be explained as follows. *Plan A* triangulates, before search, all the networks that will be checked for consistency during search (there are

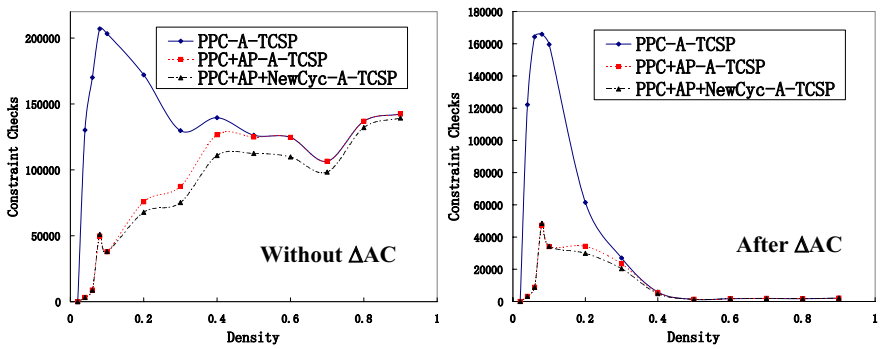


Fig. 13. *Constraint checks for* PPC-TPCS *using Plan A.*

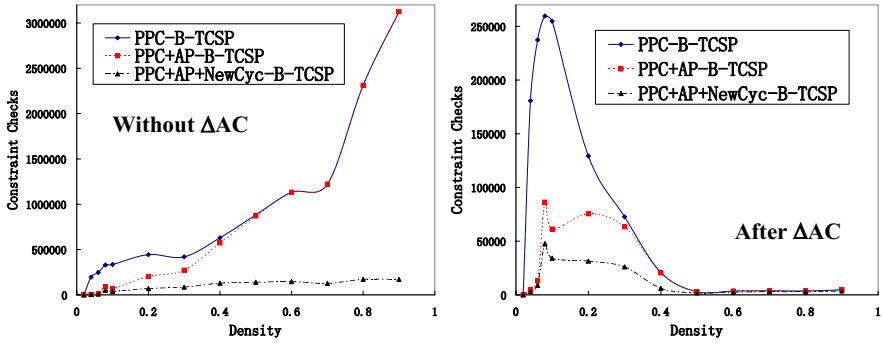


Fig. 14. *Constraint checks for* PPC-TCSP *using Plan B.*

exactly $|E|$ such graphs). *Plan B* finds the triangulation of an STP at a given node during search by inducing a subgraph from the triangulated original STP. Hence, *Plan B* triangulates the network only once, while *Plan A* carries out as many triangulation operations as the number of edges in the network (and levels in the search). However, the induced subgraphs in *Plan B* end up much denser than the ones used by *Plan A*, thus requiring more effort from PPC, the STP solver. Further, the fact that *Plan A* yields no denser graphs than *Plan B* becomes an even more desirable feature when TCSP is dense. This explains the significant differences in behavior between *Plan A* and *Plan B* under high density TCSPs.

The Winning Combination: In [21] we compared the performances of F-W, DPC, PPC, and △STP for solving an STP. We found that DPC, PPC, and △STP consistently outperform F-W, the Floyd-Warshall algorithm. Further, △STP consistently outperforms PPC. Indeed, the former is a finer version of the latter. Importantly, when the density of the temporal graph is below 0.4, △STP (which guarantees minimality) outperforms DPC (which does not). For sensibly high densities, we found DPC to be more effective. Since in the search for solving the meta-CSP we consider subgraphs of the original network, the networks at the different levels of the tree are more likely to be sparse than dense. This shows that even when the TCSP is dense, △STP is a good choice for the

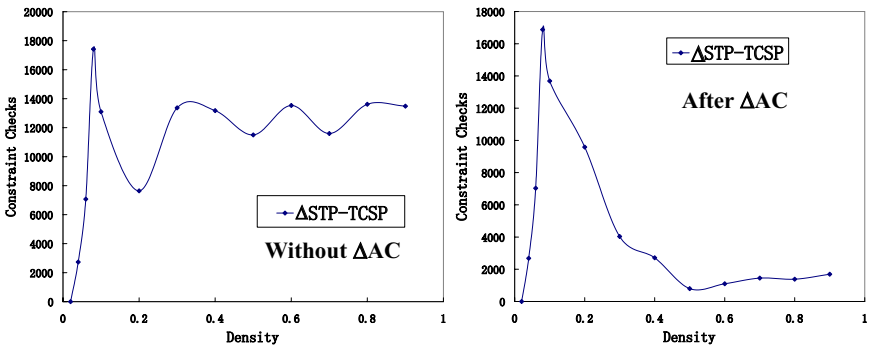


Fig. 15. *Constraint checks for △STP-TCSP.*

STP solver. Hence, among the techniques tested, the best combination one could use to solve a TCSP is the one we called △STP-TCSP (Fig. 9). Indeed △STP outperforms all TCSP solvers including the one based on DPC (compare Fig. 12 and 15).

7 Conclusions

At the beginning of our investigations, the best mechanism known to date for solving the meta-CSP[3] was one based on DPC. We introduced △STP, enhanced it with NewCyc and EdgeOrd, and showed empirically that it results in dramatic improvements. Indeed, in comparison to the original DPC, the best combination of our techniques reduces the number of constraint checks by a factor of 500 (median) and 40,000 (average) and that of CPU by a factor of 320 (median) and 1,200 (average).

Further, we showed that our techniques uncover the existence of a phase-transition-like phenomenon for solving the TCSP as the density of the network varies[4]. This is most visible with △STP-TCSP. This observation calls for more detailed investigations in this direction. As directions for future research, we plan to:

1. Exploit △AC in a lookahead strategy for solving the meta-TCSP. And,
2. Evaluate empirically how to improve BT-TCSP with dynamic bundling [3].

Beyond the TCSP, △STP is directly applicable for solving the disjunctive temporal problem (DTP) with backtrack search [19, 16, 20], but requires triangulating the STP incrementally at each node in the tree. We believe that NewCyc is also applicable as long as the constraint added applies to two points that are not yet constrained in the current path in the tree. These directions require further investigation and evaluation.

[3] Note that we do not include in our comparison algorithms that tighten these intervals in the labels of the edges. Those may not terminate in the general case and are prohibitively expensive in the integral case [18].

[4] Schwalb and Dechter [18] report a similar phenomenon when varying the number of variables and the tightness of the constraints.

Acknowledgments. We are grateful to Eddie Schwalb and Rina Dechter for various pointers to TCSPs and to Deb Derrick for editorial help. This work is supported by a NASA-Nebraska grant, CAREER Award #0133568 from the National Science Foundation, and a gift from Honeywell Laboratories.

References

1. C. Bessière and J.-C. Régin. Refining the Basic Constraint Propagation Algorithm. In *Proc. of the 17 th IJCAI*, pages 309–315, 2001.
2. C. Bliek and D. Sam-Haroud. Path Consistency for Triangulated Constraint Graphs. In *Proc. of the 16 th IJCAI*, pages 456–461, 1999.
3. B.Y. Choueiry and A.M. Davis. Dynamic Bundling: Less Effort for More Solutions. In Koenig and Holte, editors, *5th International Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*, vol 2371 of *LNAI*, pages 64–82. Springer Verlag, 2002.
4. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill Book Co & MIT Press, 2001.
5. R. Dechter. *Constraint Programming*. Morgan Kaufmann, 2003.
6. R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61–95, 1991.
7. R. Dechter and J. Pearl. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, 34:1–38, 1987.
8. R. Dechter and J. Pearl. The Cycle-Cutset Method for improving Search Performance in AI Applications. In *Third IEEE Conference on AI Applications*, pages 224–230, 1987.
9. S. Even. *Graph Algorithm*. Computer Science Press, 1979.
10. E.C. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29 (1):24–32, 1982.
11. E.C. Freuder. A Sufficient Condition for Backtrack-Bounded Search. *JACM*, 32 (4):755–761, 1985.
12. E.C. Freuder and P.D. Hubbe. A Disjunctive Decomposition Control Schema for Constraint Satisfaction. In Vijay Saraswat and Pascal Van Hentenryck, editors, *Principles and Practice of Constraint Programming*, pages 319–335. MIT Press, 1995.
13. R.M. Haralick and G.L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.
14. R. Mohr and T.C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28:225–233, 1986.
15. U. Montanari. Networks of Constraints: Fundamental Properties and Application to Picture Processing. *Information Sciences*, 7:95–132, 1974.
16. A. Oddi and A. Cesta. Incremental Forward Checking for the Disjunctive Temporal Problem. In *Proc. of the 14 th ECAI*, 2000.
17. U. Kjærulff. Triagulation of Graphs - Algorithms Giving Small Total State Space. Research Report R-90-09, Aalborg University, Denmark, 1990.
18. E. Schwalb and R. Dechter. Processing Disjunctions in Temporal Constraint Networks. *Artificial Intelligence*, 93:29–61, 1997.
19. K. Stergiou and M. Koubarakis. Backtracking Algorithms for Disjunctions of Temporal Constraints. *Artificial Intelligence*, 12- (1):81–117, 2000.
20. I. Tsamardinos and M.E. Pollack. Efficient Solution Techniques for Disjunctive Temporal Reasoning Problems. *Artificial Intelligence*, 2003. In press.
21. L. Xu and B.Y. Choueiry. A New Efficient Algorithm for Solving the Simple Temporal Problem. In *Proc. of the 10 th TIME-ICTL*, 2003. IEEE Computer Society Press.
22. L. Xu and B.Y. Choueiry. An Approximation of Generalized Arc-Consistency for TCSPs. In *Working notes of the Workshop on Spatial and Temporal Reasoning (IJCAI 03)*, 2003.