Efficient Techniques for Searching 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, 68588-0115
lxu | choueiry@cse.unl.edu

Abstract

In this paper, we address the task of solving the general 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. [1991] for this purpose. The first approach consists of using a new efficient algorithm (△STP) [Xu and Choueiry, 2003a] 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, △AC, which is used as a preprocessing step to BT-TCSP, and which significantly reduces the size of the TCSP [Xu and Choueiry, 2003b]. 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.

1 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; E is a set of edges $e_{i,j}$ representing constraints between two time points i and j; and I is a set of constraint labels for the edges, Fig. 1 (left). A constraint label $I_{i,j}$ of edge $e_{i,j}$ is a *unique* interval $[a, b]$, $a, b \in \mathbb{R}$, and denotes a constraint of bounded difference $a \leq (j - i) \leq b$. 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, 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 and 7:40..*

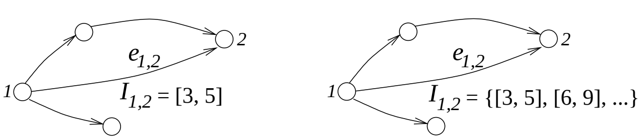


Figure 1: *Left:* STP. *Right:* TCSP.

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.

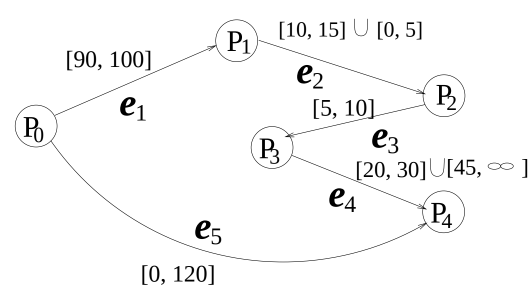


Figure 2: *A TCSP example.*

Dechter [2003] 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’ Con-

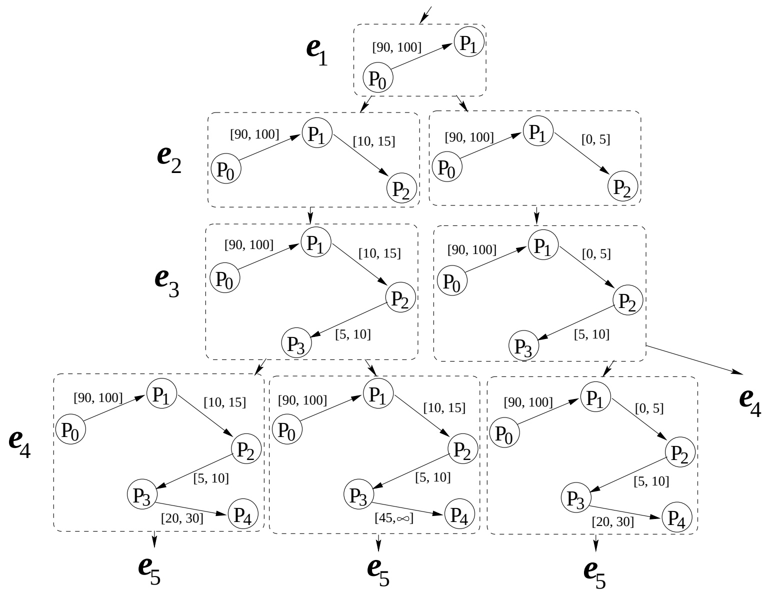


Figure 3: *The search tree for the example of Fig. 2.*

straint 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 the minimal networks of *all* complete solutions, and solving the TCSP requires finding all the solutions of the meta-CSP. 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 [Xu and Choueiry, 2003a]. We show that it 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 [Even, 1979; Freuder, 1985; Dechter *et al.*, 1991], 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 the existence of new cycles (NewCyc) in the network as edges are added along a given path in the tree. For the example shown in 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 detail in a companion paper [Xu and Choueiry, 2003b]

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 solve 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 2 reviews the various STP-solvers we used. Section 3 discusses the three improvements that exploit the topology of the temporal network. Section 4 summarizes a filtering algorithm that is more thoroughly discussed in [Xu and Choueiry, 2003b]. Section 5 describes our experiments and observations. Finally, Section 6 concludes this paper.

2 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 [Dechter and Pearl, 1988], Partial Path Consistency DPC [Bliek and Sam-Haroud, 1999], and Triangle-STP $\triangle$STP [Xu and Choueiry, 2003a].

2.1 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 [Cormen *et al.*, 2001]. F-W guarantees consistency, minimality, and decomposability and has a worst-case complexity of $\Theta(n^3)$. Montanari showed that F-W is a special case of the Path Consistency (PC) algorithm [Montanari, 1974]. Dechter et al. propose the Directed-Path Consistency (DPC) algorithm. This algorithm is never more costly than F-W, runs in $O(n^3)$, and can determine the consistency of an STP in $O(nW^*(d)^2)$, 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. Due to the fact that only the consistency of an STP matters during BT-TCSP, we use DPC instead of F-W because of its lower cost.

2.2 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) [1999]. 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 [Kjaerulff, 1990]. 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 devise two methods for accessing the triangulations of the STPs need in given a static 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. 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 i^{th} element of TriSubGs list as the triangulated subgraph for the node at the i^{th} 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.

2.3 $\triangle$STP algorithm used with TCSP algorithm

$\triangle$STP algorithm can output 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 above 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. Thus the STPs we need to check always have lower density than the original TCSP, Since Thus the outstanding performance of $\triangle$STP under low density makes it even more attractive to use for solving the TCSP.

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

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

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

3 Exploiting the topology of the constraint network

We propose three techniques 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.

3.1 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 $O(|E|)$ [Cormen *et al.*, 2001]. This method provides an upper bound to the search effort in the size of the largest biconnected component [Freuder, 1985]. 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 [Freuder and Hubbe, 1995] 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 [1987]. We have implemented the mechanism for finding and using existing articulation points but not yet explored how to induce their existence.

3.2 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[1].

Proposition 3.1. *A tree-structured constraint network is necessarily globally consistent.*

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.

[1]Note that is a stronger result than using the tree-structure of the constraint graph, which requires ensuring 2-consistency [Freuder, 1982].

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

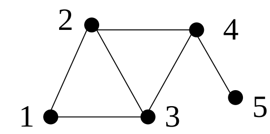


Figure 5: *Simple constraint graph.*

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.

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

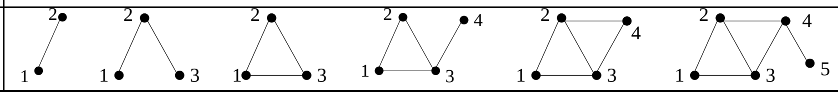
Search level	1	2	3	4	5	6	
STP							
Checking strategy							**Total**
Always	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	6
NewCyc	$\times$	$\times$	$\surd$	$\times$	$\surd$	$\times$	2

Figure 6: *Comparison of STP checks using different the new-cycle check heuristic.*

be checked as that containing the new edge.

3.3 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 the number of triangles in which it appears in G, which is bounded by $(n-1)$, where n in 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 Edge-Ord 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 examine the biconnected components of the graph in isolation, and thus decompose the graph automatically. The advantages of this mechanism are:

1. *Localized backtracking*: This heuristic is based on the topology of the temporal graph. Neighboring levels in the search tree are likely to be physically 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*: The decomposition of the graph into its biconnected components is an effective technique to bind the search effort and enhance the performance of solving a TCSP. 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.

4 $\triangle$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 [Haralick and Elliott, 1980]). The goal of an AC algorithm is to reduce the domain of the variables, thus reducing the size of the CSP and that of the search tree to be explored. Arc-consistency is usually easy to achieve in polynomial time. Quite a few general arc-consistency algorithms exist, such as AC-3 [Mackworth, 1977], AC-4 [Mohr and Henderson, 1986], AC-6 [Bessière, 1994], AC-7 [Bessière *et al.*, 1999], AC-3.1 [Zhang and Yap, 2001], and AC-2001 [Bessière and Régin, 2001].

Removing 'inconsistent' intervals from the edge labels reduces the size of the meta-CSP and directly benefits search. 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 the number of nodes by $|E| \leq \frac{n(n-1)}{2}$, 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. The only constraint in the meta-CSP is a global constraint for which no

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

Figure 7: *Edge ordering heuristic.*

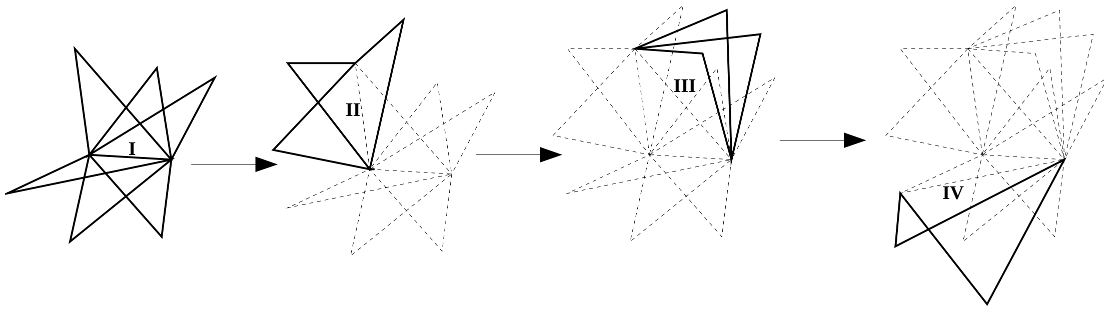


Figure 8: *Illustrating the exploration of the edges of a graph by the edge ordering heuristic.*

efficient consistency algorithm is known. In a companion paper [Xu and Choueiry, 2003b], we introduce the concept of $\triangle$Arc-Consistency as an approximation to the generalized arc-consistency of the meta-CSP. We also introduce an efficient algorithm, $\triangle$AC, that implements $\triangle$Arc-Consistency. We establish that the complexity of $\triangle$AC is $O(degree(G) \times |E| \times k^3) = O(n|E|k^3)$. This algorithm uses simple data structures to save significantly the number of constraint checks[2]. We use $\triangle$AC as a preprocessing step to search in order to reduce the size of the explored tree. We have not yet interleaved any lookahead strategy based on $\triangle$AC with search, but plan to do so in the future.

5 Experimental results

Fig. 9 shows the TCSP solvers we tested, with and without pre-processing by $\triangle$AC[3].

The STP solvers we used are `DPC`, `PPC`, and $\triangle$`STP` of Section 2. We combined them with the techniques proposed in Section 3 (i.e., AP, NewCyc, and EdgeOrd). We compared their performance 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 the companion paper [Xu and Choueiry, 2003b], 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 must 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 [4] (i.e., $\triangle$STP, AP, NewCyc, EdgeOrd, $\triangle$AC), and to establish their effectivness. It is not our goal here to compare the performance of the various STP solvers, which is discussed extensively in [Xu and Choueiry, 2003a].

Section 5.1 discusses the number of solutions of the problems tested. Naturally, all solvers must find the same solutions. Counting the number of solutions was useful to confirm that all solvers were sound and that our implementation was bug-free. Section 5.2 shows the effect of our techniques on the shape of the tree by measuring the number of nodes visited. Section 5.3 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 5.2 and 5.3 we also show how filtering the meta-CSP with $\triangle$AC dramatically improves the performance of search. The effect of this

[2] We are considering an improvement that may establish its optimality.

[3] The companion paper uses only `DPC` [Xu and Choueiry, 2003a].

[4] 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.

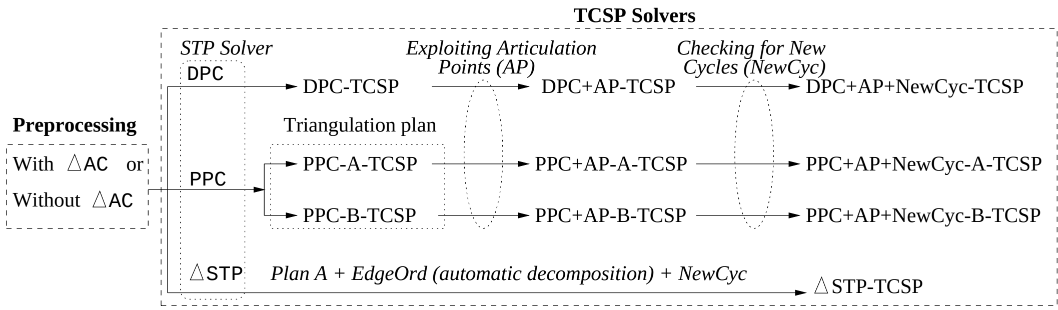


Figure 9: *TCSP solvers tested.*

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 the topic of our companion paper [Xu and Choueiry, 2003b], we confirm here that it is useful in any TCSP solver.

5.1 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 connected 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 3.1). The number of solutions quickly drops density. When d=0.9, there are only one or two solutions, one of which us guaranteed by construction.

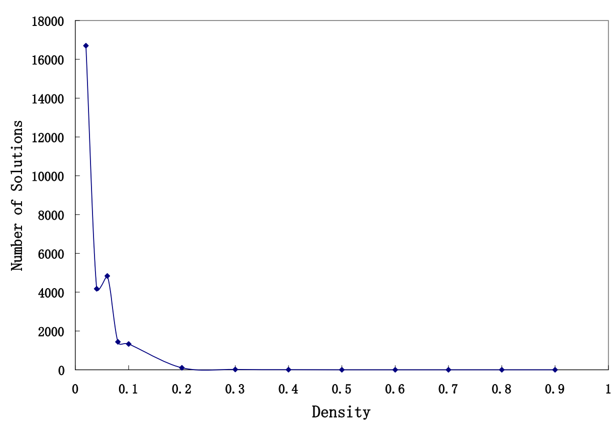


Figure 10: *The number of solutions of the meta-CSP.*

5.2 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 △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 △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.

5.3 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).

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 5.2).

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 2.2. 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 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

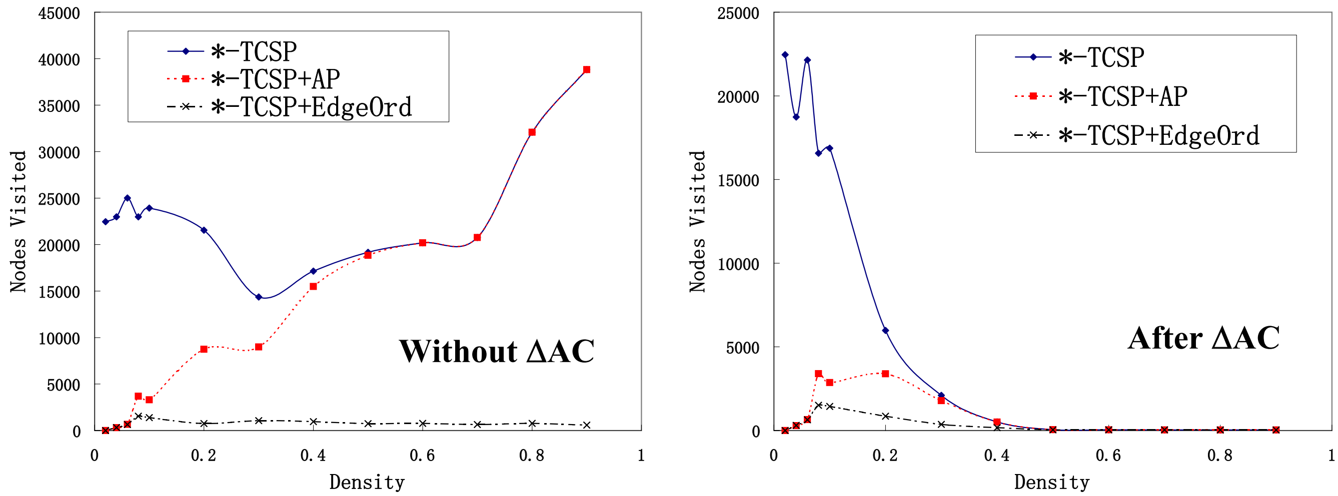


Figure 11: *Nodes visited by BT-TCSP. Left: without preprocessing. Right: after filtering with* $\triangle$*AC.*

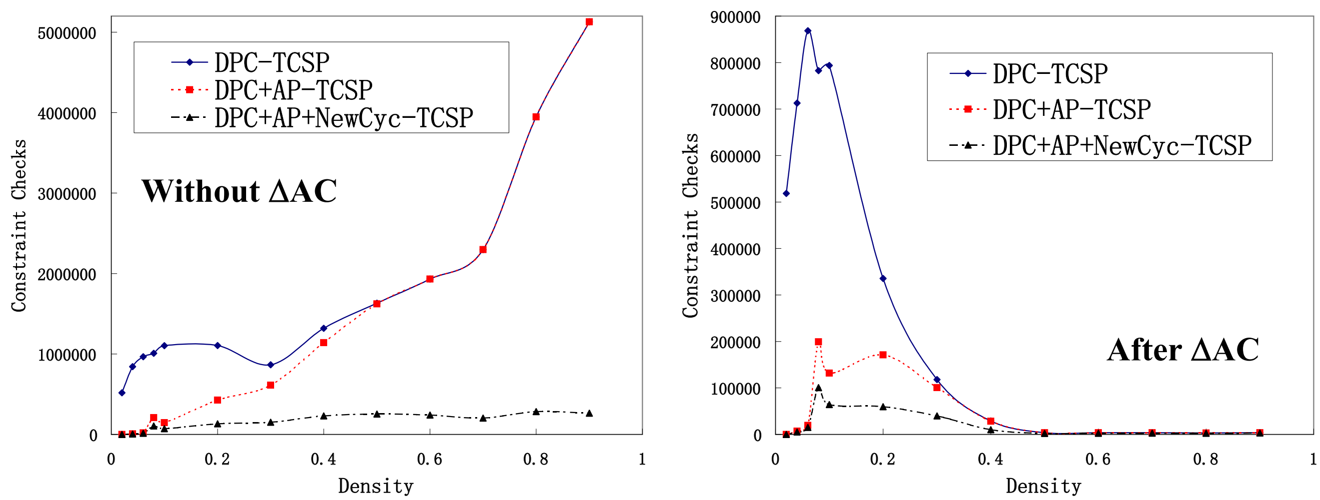


Figure 12: *Constraint checks for* DPC-TCSP.

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 [2003a] we compared the performances of `F-W`, `DPC`, `PPC`, and $\triangle$`STP` for solving an STP. We found that `DPC`, `PPC`, and $\triangle$`STP` consistently outperform `F-W`, the Floyd-Warshall algorithm. Further, $\triangle$`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, $\triangle$`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, $\triangle$`STP` is a good choice for the STP solver. Hence, among the techniques tested, the best combination one could use to solve a TCSP is the one we called $\triangle$STP-TCSP (Fig. 9). Indeed $\triangle$`STP` outperforms all TCSP solvers including the one based on `DPC` (compare Fig. 12 and 15).

6 Conclusions

At the beginning of our investigations, the best mechanism known to date for solving the meta-CSP[5] was one based on `DPC`. We introduced $\triangle$STP, enhanced it with NewCyc and EdgeOrd, and showed empirically that it results in dramatic

[5]Note that we do not include in our comparision 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 [Dechter, 2003].

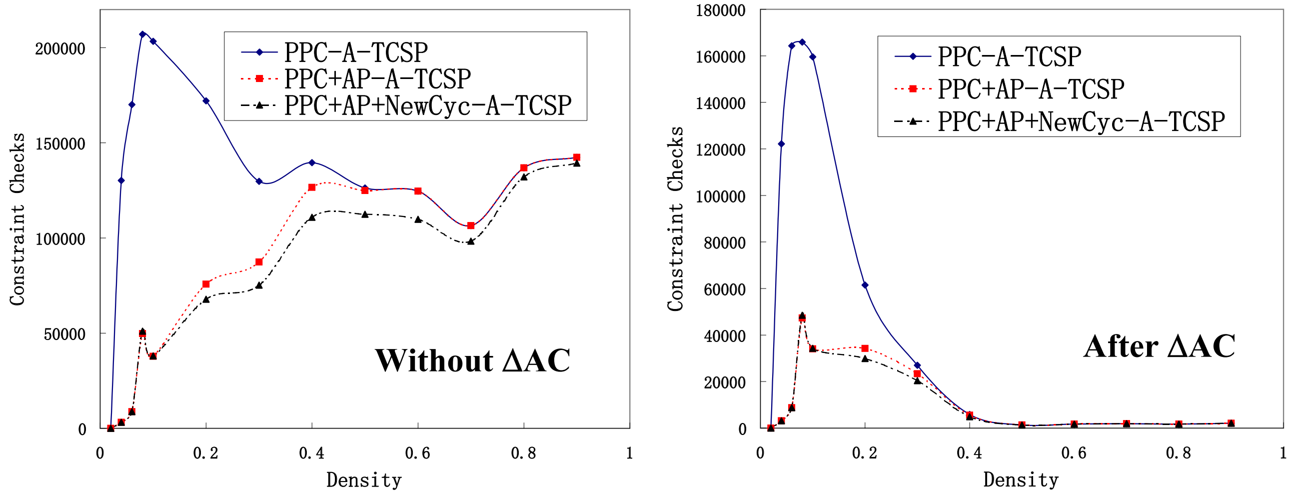


Figure 13: *Constraint checks for* PPC-TPCS *using Plan A.*

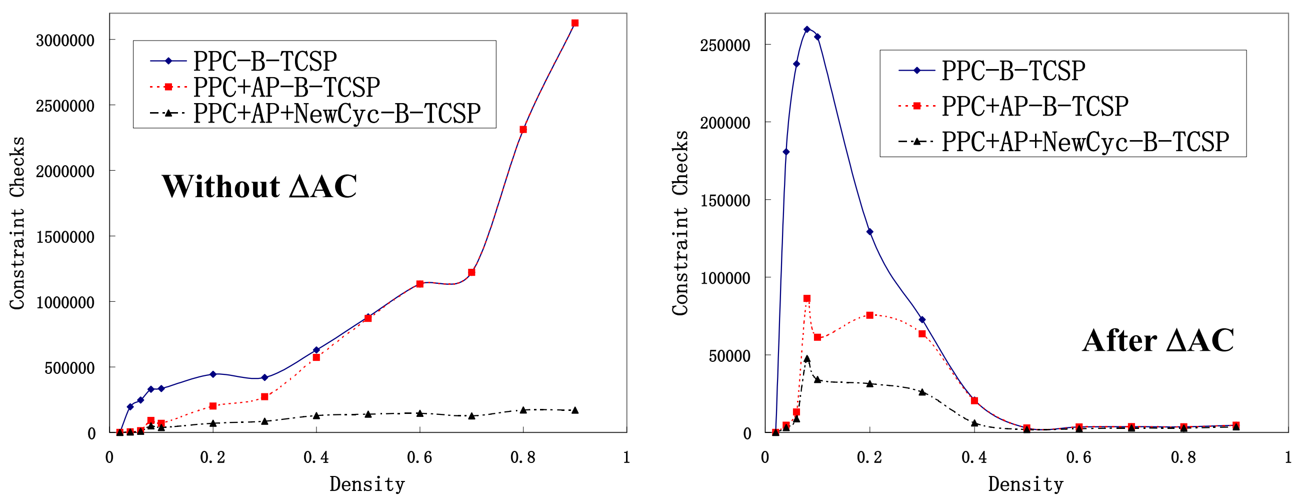


Figure 14: *Constraint checks for* PPC-TCSP *using Plan B.*

improvements. Indeed, in comparision 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, which 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. Investigate how to exploit △AC in a lookahead strategy for solving the meta-TCSP; and,
2. Evaluate empirically how to improve BT-TCSP with dynamic bundling [Choueiry and Davis, 2002], which is particularly attractive in this context since we are looking for all solutions.

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

[Bessi`ere and Régin, 2001] Christian Bessi`ere and Jean-Charles Régin. Refining the Basic Constraint Propagation Algorithm. In *Proc. of the 17 th IJCAI*, pages 309–315, Seatte, WA, 2001.

[Bessi`ere *et al.*, 1999] C. Bessi`ere, E. C. Freuder, and J.-C. R`egin. Using Constraint Metaknowledge to Reduce Arc Consistency Computation. *Artificial Intelligence*, 107 (1):125–148, 1999.

[Bessi`ere, 1994] Christian Bessi`ere. Arc-Consistency and Arc-Consistency Again. *Artificial Intelligence*, 65:179–190, 1994.

[Bliek and Sam-Haroud, 1999] Christian Bliek and Djamilla Sam-Haroud. Path Consistency for Triangulated Constraint Graphs.

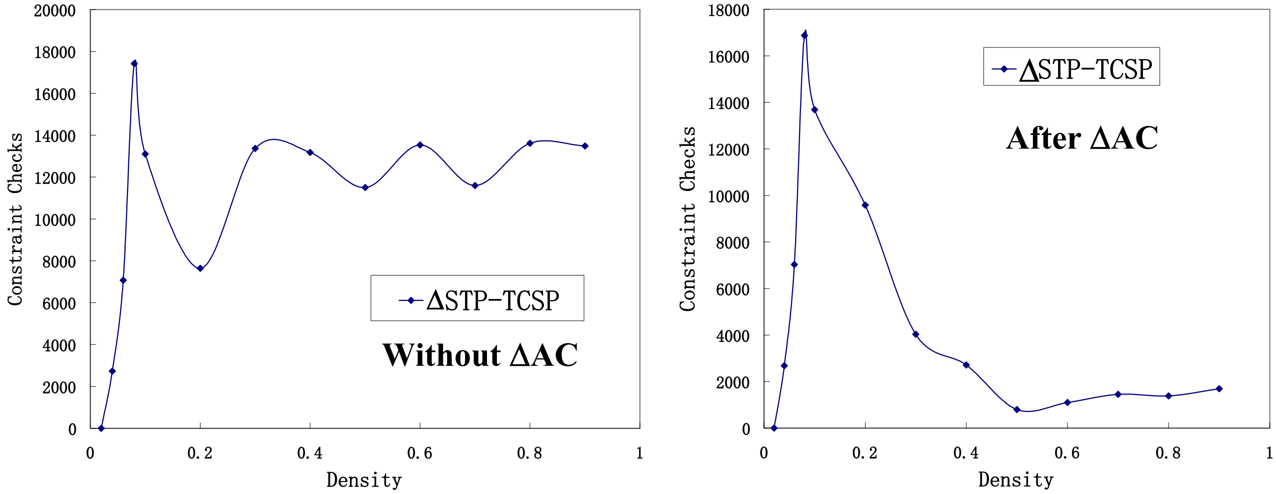


Figure 15: *Constraint checks for △STP-TCSP.*

In *Proc. of the 16 th IJCAI*, pages 456–461, Stockholm, Sweden, 1999.

[Choueiry and Davis, 2002] Berthe Y. Choueiry and Amy M. Davis. Dynamic Bundling: Less Effort for More Solutions. In Sven Koenig and Robert Holte, editors, *5th International Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*, volume 2371 of *Lecture Notes in Artificial Intelligence*, pages 64–82. Springer Verlag, 2002.

[Cormen *et al.*, 2001] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw-Hill Book Co & MIT Press, 2001.

[Dechter and Pearl, 1987] Rina Dechter and Judea Pearl. The Cycle-Cutset Method for improving Search Performance in AI Applications. In *Third IEEE Conference on AI Applications*, pages 224–230, Orlando, FL, 1987.

[Dechter and Pearl, 1988] Rina Dechter and Judea Pearl. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, 34:1–38, 1988.

[Dechter *et al.*, 1991] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61–95, 1991.

[Dechter, 2003] Rina Dechter. Constraint Processing. Manuscript, forthcoming, 2003.

[Even, 1979] Salmon Even. *Graph Algorithm*. Computer Science Press, 1979.

[Freuder and Hubbe, 1995] Eugene C. Freuder and Paul 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, Cambridge, MA, 1995.

[Freuder, 1982] Eugene C. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29 (1):24–32, 1982.

[Freuder, 1985] Eugene C. Freuder. A Sufficient Condition for Backtrack-Bounded Search. *JACM*, 32 (4):755–761, 1985.

[Haralick and Elliott, 1980] Robert M. Haralick and Gordon L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.

[Kjaerulff, 1990] U. Kjaerulff. Triagulation of Graphs - Algorithms Giving Small Total State Space, 1990.

[Mackworth, 1977] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mohr and Henderson, 1986] R. Mohr and T. C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28:225–233, 1986.

[Montanari, 1974] Ugo Montanari. Networks of Constraints: Fundamental Properties and Application to Picture Processing. *Information Sciences*, 7:95–132, 1974.

[Xu and Choueiry, 2003a] Lin Xu and Berthe 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 2003)*, Cairns, Queensland, Australia, 2003. IEEE Computer Society Press.

[Xu and Choueiry, 2003b] Lin Xu and Berthe Y. Choueiry. An Efficient Algorithm for the Filtering TCSP. In *Principles and Practice of Constraint Programming*, 2003. Submitted to Principles and Practice of Constraint Programming.

[Zhang and Yap, 2001] Yualin Zhang and Roland H.C. Yap. Making AC-3 an Optimal Algorithm. In *Proc. of the 17 th IJCAI*, pages 316–321, Seatte, WA, 2001.