A New Efficient Algorithm for Solving the Simple Temporal Problem

Lin XU and Berthe Y. CHOUEIRY
Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln
{lxu|choueiry}@cse.unl.edu

1 Abstract

In this paper we propose a new efficient algorithm, the $\triangle$STP-solver, for computing the minimal network of the Simple Temporal Problem (STP). This algorithm achieves high performance by exploiting a topological property of the constraint graph (i.e., triangulation) and a semantic property of the constraints (i.e., convexity) in light of the results reported by Bliek and Sam-Haroud [1], which were presented for general CSPs and have not yet been applied to temporal networks. Importantly, we design the constraint propagation in $\triangle$STP-solver to operate on triangles instead of operating on edges and implicitly guarantee the decomposition of the constraint graph according to its articulation points. We also provide extensive empirical evaluations of all known algorithms for solving the STP on sets of randomly generated problems. Our experiments demonstrate significant improvements of $\triangle$STP-solver, in terms of number of constraint checks and CPU time, over previously reported algorithms such as the Floyd-Warshall algorithm (F-W) [5; 8], Directed-Path Consistency (DPC) [8], and Partial Path-Consistency (PPC) [1].

2 Introduction

Many critical applications in planning and scheduling rely on an efficient handling of temporal information represented as a Simple Temporal Problem (STP) [6; 8; 3]. The efficiency of the constraint propagation in such a network is particularly crucial in autonomous space applications as demonstrated by the Deep Space 1 Remote Agent experiment [12]. Further, an efficient STP solver is a crucial component for solving the Temporal Constraint Satisfaction Problem (TCSP) because the search process designed by Dechter et al. [8] for solving the TCSP requires solving an STP at *each* node expansion. Thus, the performance of the overall process depends heavily on the performance of solving an STP. In this paper, we propose a new algorithm, $\triangle$STP-solver, for solving the STP and demonstrate empirically that it constitutes a dramatic improvement over previously used algorithms.

We achieve this by first combining the results developed by Bliek and Sam-Haroud [1] for general Constraint Satisfaction Problems (CSPs) with a *new* strategy for constraint propagation, which restricts the propagation effort to the triangles of the triangulated constraint network instead of its edges. Then, we apply the resulting mechanism to solve the STP. The triangulation of the graph and the convexity of the constraints in the STP guarantee that $\triangle$STP-solver is complete and sound for proving the consistency of the STP and for finding the minimal (and decomposable) network. This paper is structured as follows. Section 3 recalls the main properties of a CSP and shows how we use them in our study. Section 4 discusses the algorithms for solving the STP and explains the advantages of the $\triangle$STP-solver. Section 5 describes our experiments and results, and summarizes our observations. Section 6 concludes this paper.

3 Background

A Constraint Satisfaction Problem (CSP) is defined as follows. Given a set of variables, each

with a set of possible values defining its domain, and a set of constraints that restrict the combinations of values that the variables can be assigned at the same time, the task is to assign a value to each variable such that all constraints are simultaneously satisfied. Path consistency, as we discuss below, is an important property of a CSP. Recently Bliek and Sam-Haroud [1] proposed the Partial Path Consistency (PPC) algorithm, which determines whether or not a network is path consistent. Since PPC operates on the triangulated constraint graph [1], it realizes significant computational savings over previously known algorithms, especially for sparse networks. In this paper, we first improve the propagation mechanism of the PPC algorithm by making it operate on triangles instead of individual edges. We then use the improved version to solve the STP. In the next section, we recall the main properties of a CSP and discuss them in light of the STP.

3.1 Main CSP properties

The general properties of constraint graphs and the main algorithms for achieving them are outlined below.

• *Path consistency:* This property ensures that given two values for any two variables that satisfy the constraint between these variables, we can find values for variables in any path of any length (possibly infinite) that satisfy the constraints *along* the path [11]. In general CSPs, path-consistency algorithms PC (e.g., PC-1 [11] and PC-2 [10]) are used to enforce path consistency by tightening the binary constraints. (They also tighten the domains, thus enforcing strong path-consistency.) Montanari established that these algorithms, which consider only paths of length two, on a *complete* graph[2] guarantee a path-consistent network [11]. The Directional Path-Consistency (DPC) algorithm, which achieves path consistency *along a given ordering* d of the variables in the search process, was proposed by Dechter [7] as an efficient approximation of PC; it guarantees path consistency only in the direction that matters, which is that of search. Recently Bliek and Sam-Haroud [1] proposed the Partial Path Consistency (PPC) algorithm, which de-

termines whether or not a network is path consistent without necessarily producing a tight network as with PC. Since PPC operates on the edges of the triangulated graph (fewer than those of the complete graph), it realizes significant computational savings, especially in sparse networks.

• *Minimality:* Minimality, the central problem in CSPs, is a property stronger than path consistency. It guarantees that all the binary constraints are as explicit (i.e., tight) as possible [11].

• *Decomposability:* Decomposability is stronger than minimality and guarantees that a solution to the CSP can be found backtrack-free. This is a highly desirable property and guarantees the tractability of the CSP.

• *Consistency:* In contrast to the above, the consistency property guarantees only the existence of a solution. Note that decomposability is a sufficient condition for consistency.

• *Decomposition into biconnected components:* The decomposition of the constraint graph into its biconnected components according to its articulation points[3] is a known technique for enhancing the performance of solving a CSP in general. It provides an upper bound, in the size of the largest biconnected component, to the search effort [9]. We establish that the new solver we introduce, $\triangle$STP, implicitly decomposes the constraint graph into its biconnected components without using articulation point. This important observation justifies its high performance.

3.2 Properties of the STP

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; see Figure 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

---

[1]A graph is triangulated if every cycle of length strictly greater than 3 possesses a chord.

[2]If the graph is not complete, it is made so by adding universal constraints between non-adjacent edges.

[3]An articulation point of a graph is a vertex whose removal disconnects the graph. A graph with an articulation point is separable, otherwise it is biconnected.

constraints of bounded differences between i and j, see Figure 1 (right). We assume that the intervals in
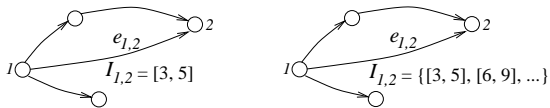
Figure 1. *Left:* STP. *Right:* TCSP.

a label are ordered in a canonical way. In this paper we focus on STPs, but we are integrating our results into an algorithm for solving TCSPs. Below, we show how we exploit the properties of Section 3.1 in the context of the STP.

• *Triangulation of network and convexity constraints.* In addition to proposing PPC, Bliek and Sam-Haroud also 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 the same labeling for the edges common to both graphs. *This important feature of the PPC algorithm has never been exploited before in the context of STPs*, in which the constraints– linear inequalities–are indeed convex. Our $\triangle$STP-solver exploits this result and yields significant savings of the computational efforts over previously available techniques for establishing path consistency of the STP.

• *Distribution of composition over intersection.* The two operators on binary constraints for establishing path consistency are constraint composition $\otimes$ and constraint intersection $\oplus$. Montanari showed that when constraint composition is distributive over constraint intersection, PC guarantees not only path consistency but also minimality and decomposability [11]. In the case of the STP, constraint composition is interval addition, and constraint intersection is interval intersection, which verify the distributivity as noted by Dechter et al. [8]. Therefore we can deduce that the PPC algorithm and the $\triangle$STP-solver, guarantee the minimality and decomposability of the STP. DPC does not guarantee the path-consistency, minimality or decomposability of the constraint network, however, and this is an important feature, it can be used to determine the consistency of the STP.

• *Decomposition into biconnected components.* In the special case of the TCSP, and a fortiori the STP, Dechter et al. [8] showed that each biconnected

component can be solved independently. If all the components are found to be consistent, then the entire network is consistent. If any of the components is not consistent, then the overall temporal network is not consistent. The minimal network of the original problem is obtained by the union of the minimal networks of the individual biconnected components. When the constraint graph is sparse, this property is particularly attractive. This allows us to process the components in parallel, by independent agents. Thus, decomposition into biconnected components is particularly attractive in the case of STPs, especially for large problems with sparse graphs. We show that this decomposition is implicit and automatic in our $\triangle$STP-solver.

4 STP algorithms

Here we discuss four different algorithms to solve STPs. The first two solvers, F-W and DPC, have been extensively studied. However, their performance in combination with a decomposition strategy according to articulation points has never been compared before. The third STP solver we study is PPC, which has never before been used on temporal reasoning problems. Finally, we introduce our new solver, $\triangle$STP.

4.1 F-W & DPC **with articulation points**

The Floyd-Warshall (F-W) algorithm for computing all-pairs shortest-paths is a special case of the PC algorithm. F-W is applied to the distance graph of an STP to compute its minimal network in $\Theta(n^3)$. As discussed in Section 3, DPC is a single pass algorithm and weaker than PC. It does not necessarily yield a path consistent, minimal, or decomposable network, but it determines if the STP is consistent. DPC can be more efficient than F-W; instead of $\Theta(n^3)$, DPC can determine the consistency of STP in $O(nW^*(d)^2)$, where $W^*(d)$ is the maximum number of parents that a node has in the induced graph along the ordering d, which can be substantially smaller than n.

We modify the F-W and DPC algorithms to exploit the existence of articulation points in the temporal network. First, we identify the biconnected components [5], then we execute a particular STP solver on each component, independently. This

yields two algorithms, `F-W+AP` and `DPC+AP`, respectively. It is easy to show that `F-W+AP` and `DPC+AP` never check more constraints than `F-W` and `DPC`. In fact, for a sparse network, our experiments show that they check substantially less. We also show empirically that, even in the absence of articulation points, `F-W+AP` and `DPC+AP` almost never require more CPU time than the original algorithms; when they do, the difference is insignificant due to the overhead for finding the articulation points.

4.2 `PPC` algorithm for STPs

`PPC` was introduced for general CSPs by Bliek and Sam-Haroud [1] who showed that the path-consistency property can be determined in constraint graphs by triangulating them instead of completing them. They showed a significant improvement in performance in comparison to `PC` in sparse networks. They also established that, for convex constraints, both `PPC` and `PC` compute the same labeling for the edges common to both graphs. Since the constraints in the STP (constraints of bounded difference) are convex, we apply for the first time `PPC` to solve a continuous domain problem and compute the minimal network of the STP.

As specified in Figure 2, the `PPC` algorithm starts by triangulating the constraint graph G, then iterates over a queue Q_E of all edges, including those edges added to the temporal graph by the triangulation process. It pops an *arbitrary* edge $e_{i,j}$ from the queue, recovers all triangles $\langle i, j, k \rangle$ in which $e_{i,j}$ participates, and updates its label $I_{i,j}$ by composing the intervals $I_{i,k}$ and $I_{k,j}$ and intersecting the result of this composition with the interval $I_{i,j}$. We slightly modify the original algorithm to allow it to update all three edges at once and to terminate when the queue is empty or inconsistency is found. The distributivity property of interval addition over interval intersection guarantees that running `PPC` on an STP results in the tightest possible labeling (i.e., minimal) of the existing edges.

4.3 $\triangle$STP algorithm

The goal of `PPC` is to make the labels of the edges of the triangulated constraint graph as tight as possible. When the label of an edge in a triangle

```
PPC (P):
Begin
consistency ← True
G ← Triangulate (P)
Q_E ← edges in G
While Q_E ∧ consistency Do
    e_{i,j} ← Dequeue(Q_E)
    Forall k such that ⟨i, j, k⟩ is a subgraph of G Do
        I'_{i,j} ← I_{i,j} ⊕ (I_{i,k} ⊗ I_{k,j})
        When I'_{i,j} ≠ I_{i,j} Then I_{i,j} ← I'_{i,j} and Enqueue(e_{i,j}, Q_E)
        I'_{i,k} ← I_{i,k} ⊕ (I_{i,j} ⊗ I_{j,k})
        When I'_{i,k} ≠ I_{i,k} Then I_{i,j} ← I'_{i,j} and Enqueue(e_{i,k}, Q_E)
        I'_{j,k} ← I_{j,k} ⊕ (I_{j,i} ⊗ I_{i,k})
        When I'_{j,k} ≠ I_{j,k} Then I_{j,k} ← I'_{j,k} and Enqueue(e_{j,k}, Q_E)
        When I_{i,j}, I_{i,k} or I_{j,k} is empty Then consistency ← False
Return consistency
End
```

Figure 2. *The `PPC` algorithm, slightly improved to consider simultaneously all three edges in a triangle.*

is not as tight as it could be, given the labels of the other edges in the triangle, the label is tightened accordingly. This process may require tightening the other edges in the triangle as shown in Figure 3. In
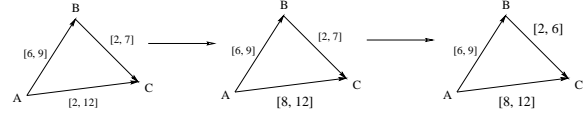

Figure 3. *An example of updating edges.* The label of edge BC then that of AC are updated.

this example we can see that it is worth considering all three edges of a given triangle simultaneously and updating them sequentially. This observation is the basis of our first improvement to `PPC`, and is already integrated in the algorithm of Figure 2.

When the label of an edge in a given triangle is updated, `PPC` triggers constraint propagation over *all* the triangles in which *any* of the edges of the original triangle participate. This is clearly an overkill since only the triangles in which the updated edges participate need to considered. This observation was the motivation for our new algorithm.

While all existing methods consider the temporal network as composed of edges, our new algorithm considers the STP as composed of triangles (see Figure 4). The graph of the temporal network

4
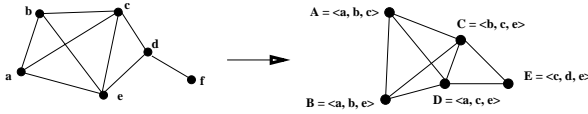
Figure 4. *The temporal graph as a graph of tri-angles.*

is replaced by a graph of triangles. Each triangle is represented by a node, and two nodes are connected if and only if the triangles they represent have a common edge. Thus $\triangle$STP appears as an AC3-like algorithm [10] on this graph of triangles. If an edge of the original constraint graph is not a part of any triangle, it is omitted from the graph of triangles. Indeed, an edge that does not appear in any triangle has no effect on the constraint propagation in the STP and thus can be safely omitted from the graph of triangles. Consequently:

Proposition 4.1. *A tree-structured STP is decomposable and consistent, and its edge labels are minimal.*

We call our new algorithm $\triangle$STP, although it is applicable to general CSPs and would more correctly be called $\triangle$PPC. The new algorithm is shown in Figure 5. First, we triangulate the temporal net-

Figure 5. *The $\triangle$STP algorithm.*

work, using for example the algorithm devised in [13], which may result in new edges. We add these edges to the original constraint graph as universal constraints setting their label to $(-\infty, \infty)$. Then we put all the triangles into a queue, Q_T, of size $O(|E|\ degree(G)))^4$. We check every triangle in the queue. If a given triangle $\langle i, j, k \rangle$ is not minimal, then we update one or more of its edges. We then retrieve all the adjacent triangles that contain any of the updated edges and add them to Q_T if they are not already there. Finally, we remove $\langle i, j, k \rangle$ from the queue, and repeat this process until Q_T is empty or inconsistency is found.

4.4 Features of $\triangle$STP

We summarize the features of $\triangle$STP as follows:
- $\triangle$STP *has the same pruning power as* F-W *with less effort.* $\triangle$STP achieves minimality on the triangulated graph, without requiring the completion of the graph, which is necessary for F-W. This yields dramatic gains in the computational effort.
- $\triangle$STP *automatically decomposes 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 CSP. Our experiments of Figure 7 and 8 and Table 2 and 3 show how such a strategy can improve the performance of the F-W algorithm, even when the articulation points must be explicitly identified. Because constraints propagate through triangles, PPC and consequently $\triangle$STP implicitly exploit the decomposition into biconnected components. Consider a triangulated temporal network composed of two sets of nodes $X = \{p, x_1, x_2, \ldots, x_m\}$ and $Y = \{p, y_1, y_2, \ldots, y_n\}$, and p is the articulation point. Suppose that edges exist only between nodes in either X or Y. Since no edges connect these two sets, there obviously are no triangles that connect them. All triangles are either in set X or in set Y. As shown in Figure 4, two triangles in the graph of triangles can only be connected by a common edge. Therefore, no triangle in set X is connected to a triangle in set Y. When PPC and consequently $\triangle$STP propagate constraints through neighboring triangles, no updates in set X may affect triangles in set Y. As a result, PPC and $\triangle$STP implicitly guarantee that ar-

---

[4]Note that $1 \leq degree(G)) \leq n$

5

ticulation points in the graph (if any), are expoited, as if the network was decomposed into its biconnected components without actually decomposing it.

• $\triangle$STP *is cheaper than* PPC. $\triangle$STP and PPC use the same idea of Bliek and Sam-Haroud [1]; however, $\triangle$STP is more careful about how updates are propagated and thus exploits triangulation of the graph more effectively than PPC. Although propagation of PPC occurs through triangles, PPC does not have a mechanism to record which triangles really need to be checked. This inability causes some unnecessary constraint checks and a waste of CPU time.

• *Our improvement in solving the STP directly benefits the task of solving the TCSP.* TCSP is **NP**-hard and is solved with backtrack search. Every node expansion in the search tree needs to solve an STP. Thus a good STP solver is crucial for solving the TCSP. We are currently demonstrating this idea and showing how the decomposition into independent components is particularly useful in this context.

5   Empirical evaluations

We implemented the following six algorithms in Common Lisp. Floyd-Warshall (F-W), Directed-Path Consistency (DPC), and in combination with a mechanism for detecting and exploiting articulation points, F-W+AP and DPC+AP, Partial Path Consistency (PPC), and our new triangle-based solver ($\triangle$STP). We used three generators of random STPs: GenSTP-1, SPRAND, and GenSTP-2. GenSTP-1 is our own generator. We designed it to guarantee that graphs are connected and that at least 80% of the generated instances are consistent. SPRAND is one class of STPs generated by the public domain library SPLIB, [4]. All the problems we generate with SPRAND have a cycle connecting all the nodes (i.e., a structural constraint). This guarantees strong connectivity and the absence of any articulation points. Finally, GenSTP-2 is a generator given to us by Ioannis Tsamardinos and was used in [14]. GenSTP-2 does not enforce the existence of a structural constraint. The density of the temporal network is defined as $Density = \frac{|E| - |E_{min}|}{|E_{max}| - |E_{min}|}$. Table 1 summarizes the characteristics of the problems tested, including the size of the instances and the number of samples generated for each measurement point. The results, measured in terms of the number of constraint checks and CPU time, were averaged over the number of instances and showed a precision of 5%. The detailed data of the above experiments on the instances generated by GenSTP-1 and SPRAND are shown in Table 2 and 3. The CPU time measurements are made in msec, with a clock resolution of 10 msec.

5.1   Experiments conducted

Using the 50-node problems generated by GenSTP-1, we conducted the following experiments:

• *Managing the queue in* $\triangle$STP. The manner in which triangles are inserted in the queue affects the performance of $\triangle$STP. We tested three heuristics for adding the triangles to the queue: at the front of queue ($\triangle$STP-front), at the end of queue ($\triangle$STP-back), and random insertion into the queue ($\triangle$STP-random). All three strategies resulted in the same output (i.e., the same label of the edges). The results in terms of constraint checks are presented in Figure 6. The results show that $\triangle$STP-back consistently performs the least number of constraint checks. This can be informally in-
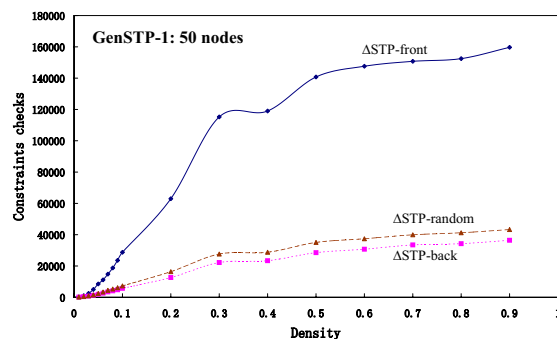
Figure 6. *Constraint Checks for* $\triangle$STP-front, $\triangle$STP-back *and* $\triangle$STP-random.

terpreted as follows. It is more effective to propagate the constraints as early as possible across the network, in a 'sweeping' manner. Interestingly, we noticed that quiescence was consistently reached in 7 or fewer iterations. We use $\triangle$STP-back in the rest of our study.

• *Computing the minimal network.* F-W, F-W+AP,

6

Table 1. *Parameters of problems generated.*

Generator	Problem size					Samples per point	Results
	#Nodes	Density		#Edges			
		Range	Step	Range	Step		
GenSTP-1	50, 100	[0.01, 0.1]	0.01			100	Table 2 and
	50, 100	[0.2, 0.9]	0.1			100	Figure 6, 7, 8
SPRAND	50	[200, 2000]	200			100	Table 3
	100	[400, 1400]	200			100	
	100	[1600, 2800]	400			100	
	257	0.016		768		5	Figure 9
	513	0.008		1536		5	
GenSTP-2	256	0.016		$3 \times 256 = 768$		5	Figure 9
	512	0.008		$3 \times 512 = 1536$		5	
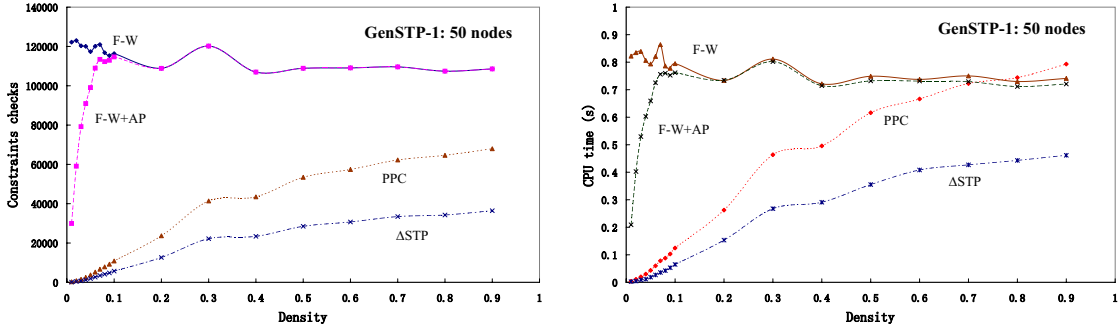
Figure 7. *Constraint Checks (left) and CPU time (right) for* F-W, F-W+AP, PPC, *and* △STP.

PPC and △STP (but not DPC) result in the labels of the common edges, the minimal labels. Figure 7 shows that △STP clearly and significantly dominates all others, for all values of density.

• *Saving on the constraint checks.* DPC does not necessarily yield the minimal network, but it can determine whether or not the network is consistent in significantly fewer constraint checks than F-W. Figure 8 shows that △STP, which is more powerful in terms of pruning power and yields the minimal network, dominates DPC-like strategies when density is less than 50%.

• *Effect of problem size.* In order to compare the performance of these different solvers on larger problems, we tested them on larger problems generated by SPRAND and GenSTP-2. Figure 9 and 10 show the ratio of the number of constraint checks and that of the CPU time needed for all six strategies in reference to the values needed for F-W.

5.2 Observations

From the above experiments, we draw the following observations:

• *Using articulation points.* Dechter et al. [8] showed that decomposing the temporal network into its biconnected components is particularly effective in enhancing the performance of search. It is worth recalling that this decomposition does not affect the quality of the solution: the same edge labels are found with and without decomposition. Figure 7 and 8 show that only F-W realizes significant savings when the density is low. In contrast, decompostion into biconnected components does not benefit the DPC solver to the same extent. This can be explained by the fact that the cost of DPC is bounded by $O(nW^*(d)^2)$, where $W^*(d)$ is the maximum number of parents that a node has in the induced graph. Decomposition does not significantly change the induced width $W^*(d)$; the total cost of solving the subproblems is not sig-

Table 2. *Experimental results for STP solvers on random STP generated by* `GenSTP-1`.

Density	F-W CC	F-W CPU (s)	F-W+AP CC	F-W+AP CPU (s)	DPC CC	DPC CPU (s)	DPC+AP CC	DPC+AP CPU (s)	PPC CC	PPC CPU (s)	△STP CC	△STP CPU (s)
Random STP generated by `GenSTP-1` with 50 nodes												
0.01	122200.5	0.822	29924.05	0.2091	1777.03	0.1168	744.44	0.0307	273.97	0.0039	125.75	0.0025
0.02	123001.5	0.8347	59091.93	0.4026	3572.7	0.1304	2364.62	0.0683	837.9	0.0109	409.64	0.0045
0.03	120339.99	0.8389	79195.61	0.5297	4769.95	0.1376	3833.36	0.0945	1532.55	0.02	761.71	0.0091
0.04	120044.01	0.8063	90934.63	0.6029	6411.11	0.1547	5525.58	0.1176	2529.68	0.03	1270.41	0.0115
0.05	117382.5	0.7935	99076.94	0.6591	8106.14	0.161	7510.24	0.1394	3766.13	0.0433	1910.97	0.0188
0.06	120075.49	0.8209	108975.06	0.7251	10204.46	0.1804	9746.2	0.1679	5207.57	0.0599	2622.19	0.0269
0.07	120940.51	0.8637	113426.05	0.756	11487.391	0.189	11175.431	0.1818	6679.19	0.0782	3445.79	0.0358
0.08	116800	0.7862	112267.63	0.7598	11715.94	0.1894	11447.12	0.181	7861.92	0.0879	4109	0.0424
0.09	115321.5	0.7778	112951.92	0.7525	13024.311	0.1976	12915.95	0.1986	9240.66	0.1031	4800.74	0.0531
0.1	116336.5	0.7947	114676.23	0.7617	14072.08	0.2115	13975.311	0.207	10857.08	0.1247	5705.62	0.0649
0.2	108926.5	0.7335	108852.99	0.7342	21203.27	0.2717	21203.262	0.2705	23677.2	0.2624	12631.6	0.1533
0.3	120195.99	0.8113	120195.99	0.8019	28912.988	0.347	28912.988	0.3442	41404.09	0.4637	22206.16	0.2676
0.4	106959.5	0.7213	106959.5	0.7147	27121.85	0.3313	27121.85	0.3252	43483.79	0.4958	23388.791	0.291
0.5	108896.5	0.7487	108896.5	0.732	29731.49	0.3506	29731.49	0.3514	53446.668	0.6162	28504.24	0.3553
0.6	109074.99	0.7376	109074.99	0.7314	31533.85	0.3732	31533.85	0.3692	57422.24	0.6662	30716.22	0.4083
0.7	109592	0.7502	109592	0.7294	32002.16	0.3795	32002.16	0.3725	62265.727	0.7224	33464.38	0.4269
0.8	107428.51	0.7298	107428.51	0.7116	32391.83	0.3816	32391.83	0.3719	64625.727	0.7439	34257.42	0.443
0.9	108566.5	0.741	108566.5	0.7207	33249.992	0.3925	33249.992	0.3796	67977.31	0.7931	36429.34	0.4616
Random STP generated by `GenSTP-1` with 100 nodes												
0.01	976155.06	8.3611	486223.66	4.088	21574.19	1.0156	14401.68	0.5275	4424.22	0.0586	2225.99	0.0108
0.02	955417	8.2284	737264.25	6.2037	45044.293	1.3432	39022.73	0.9329	14764.17	0.2035	7803.66	0.0772
0.03	944883	7.9927	855073.25	7.142	71363.34	1.3655	67750.06	1.2528	31849.158	0.3818	16698.209	0.1795
0.04	920881.06	7.8254	879589.9	7.3463	89384.945	1.4859	87387.805	1.4347	49463.91	0.5777	26350.969	0.3076
0.05	931483.06	7.8308	918906.56	7.71	115620.83	1.7429	114994.93	1.7121	72491.46	0.8411	38301.637	0.472
0.06	886372.94	7.5324	879934.7	7.3403	116526.336	1.6933	116144.984	1.6616	85443.125	1.0262	45141.34	0.5847
0.07	916842	7.7882	914465.9	7.6159	145073.03	1.9288	144846.11	1.9396	113607.77	1.3013	61303.09	0.8185
0.08	924955.94	7.907	924361.94	7.7039	148479.61	1.9404	148393.72	1.9335	129904.16	1.4633	70892.98	0.9267
0.09	935953	7.9439	935805.6	7.7978	167192.17	2.1092	167192.17	2.1225	161399.25	1.8614	86110.63	1.1857
0.1	895177	7.7186	894583	7.4615	165887.34	2.086	165803.48	2.0561	165634.69	1.9312	90790.92	1.2733
0.2	883597	7.5387	883597	7.3604	218225.31	2.4666	218225.31	2.4527	320976.06	3.7723	175113.86	2.6166
0.3	860400	7.4074	860400	7.1667	232372.25	2.5446	232372.25	2.5384	396075.3	4.7658	219178.31	3.3071
0.4	833850	7.1203	833850	6.9958	240254.4	2.6094	240254.4	2.5553	446748.47	5.4984	247012.77	3.805
0.5	879490.06	7.554	879490.06	7.3287	262964.03	2.8133	262964.03	2.7976	520176.78	6.4435	287163	4.4565
0.6	891914.06	7.6565	891914.06	7.4904	276184.53	2.9108	276184.53	2.8815	564749.56	6.734	309157.75	4.7986
0.7	866636	7.4485	866636	7.3051	267027.4	2.8092	267027.4	2.8233	554381.6	6.5875	303306.12	4.7356
0.8	847892	7.3271	847892	7.2994	258738.61	2.733	258738.61	2.6769	552344.1	6.4986	299997.22	4.8764
0.9	854969	7.3954	854969	7.3383	266861.47	2.8032	266861.47	2.7704	568128.25	6.7406	309514.87	4.9663

nificantly smaller than that of solving the original problem. When density is high, the network cannot be decomposed, and `F-W+AP` and `DPC+AP` perform almost the same as `F-W` and `DPC`, respectively. The problems generated by `SPRAND` cannot be decomposed because of the existence of a cycle that connects all nodes (i.e., structural constraint). Indeed, Table 3 shows the same number of constraint checks for the algorithms with and without articulation points. However, the required effort for finding these articulation points is negligible, as CPU times are the same within the resolution of the clock.

• *Improvements due to* `PPC`: Given the constraint semantics, `PPC` is guaranteed to yield the same labels as `F-W` and `F-W+AP` on their common edges. Since `PPC` operates on the triangulated graph, it performs significantly better for low density values than `F-W`, which operates on the complete graph, and even `F-W+AP`, which exploits the existence of articulation points. When the constraint density increases, the number of triangles in the graph also increases and so does the cost of `PPC`. However, the number of constraint checks and, to some extent, the CPU time for `PPC` remain less than those for `F-W` and `F-W+AP`, which quickly reach a stable

value, $\Theta(n^3)$. For the larger problems generated by `SPRAND` and `GenSTP-2`), Figure 9 and 10 show the `PPC` outperforms `DPC` and `DPC+AP`, which in turn outperform `F-W` and `F-W+AP`. Note, however, that `DPC` and `DPC+AP` do not yield the tightest network. A comparison of Figure 9 and 10 shows that the performance of `PPC` is better on problems generated by `GenSTP-2` than on those generated by `SPRAND`. This is due to the existence of a cycle connecting all the nodes in problems generated by `SPRAND`, which prevents decompositions and causes the triangulation process to add relatively more edges.

• *Improvements due to* △STP. As a refinement of `PPC`, △STP exploits the benefits of triangulation to a greater degree than `PPC` does. Experimental results show that △STP has always better performance than `PPC` in all experiments we performed (Figure 7 and Table 2 and 3). For high density values, △STP can show a worse performance than `DPC` (Figure 8). However, this slight degradation is misleading since it does not account for the output of these two algorithms. Indeed, △STP guarantees the minimal network and `DPC` does not. Hence, the performance of the former remains superior. The

8

Table 3. *Experimental results for STP solvers on random STP generated by* SPRAND.

Number of Edges	F-W		F-W+AP		DPC		DPC+AP		PPC		△STP	
	CC	CPU (s)	CC	CPU (s)	CC	CPU (s)	CC	CPU (s)	CC	CPU (s)	CC	CPU (s)
Random STP generated by SPRAND with 50 nodes												
200	125000	0.8255	125000	0.8301	21824.031	0.2798	21824.031	0.2847	20247.77	0.236	12111.471	0.1595
400	125000	0.8492	125000	0.8301	30981.5	0.3677	30981.5	0.3732	42313.25	0.4893	25902.35	0.347
600	125000	0.8441	125000	0.8244	34524.73	0.4044	34524.73	0.4035	56231.418	0.6606	34142.043	0.4656
800	125000	0.8467	125000	0.8274	36254.89	0.4255	36254.89	0.4176	64894.547	0.7594	39436.86	0.5334
1000	125000	0.8457	125000	0.8281	37302.24	0.4369	37302.24	0.4318	69790.15	0.825	42623.07	0.5697
1200	125000	0.8521	125000	0.8242	38020.63	0.4473	38020.63	0.4382	73899.914	0.8671	44889.09	0.5796
1400	125000	0.8501	125000	0.8243	38502.508	0.4556	38502.508	0.4442	76743	0.9067	46354.59	0.608
1600	125000	0.8513	125000	0.8331	38902.95	0.4647	38902.95	0.4458	79116.336	0.927	47597.69	0.6287
1800	125000	0.8553	125000	0.8343	39166.152	0.4694	39166.152	0.4532	80540.03	0.9526	48321.05	0.6306
2000	125000	0.8621	125000	0.8363	39381.36	0.4577	39381.36	0.4519	81024.4	0.9536	48789.93	0.6291
Random STP generated by SPRAND with 100 nodes												
400	1000000	8.5076	1000000	8.3707	167877.39	2.1703	167877.39	2.1947	144819.36	1.7659	85055.414	1.4427
600	1000000	8.5019	1000000	8.3572	218599.22	2.5686	218599.22	2.5723	241016.73	2.8585	146966.83	2.5927
800	1000000	8.5177	1000000	8.3523	245378.12	2.775	245378.12	2.7759	318725.3	3.7333	198716.12	3.441
1000	1000000	8.5218	1000000	8.3476	263177.97	2.9213	263177.97	2.9205	380805.94	4.4388	236103.58	4.1202
1200	1000000	8.6507	1000000	8.3242	275036.7	3.0351	275036.7	3.0053	434212.72	5.0349	268235.28	4.6083
1400	1000000	8.6643	1000000	8.3216	283548.44	3.0986	283548.44	3.0367	474789.12	5.4202	292905.87	4.886
1600	1000000	8.7028	1000000	8.3169	289520.4	3.1461	289520.4	3.082	512087.9	6.1406	313113.25	5.4773
2000	1000000	8.7978	1000000	8.3284	298104.53	3.2074	298104.53	3.148	565111.94	6.9515	343748.66	6.0293
2400	1000000	8.5296	1000000	8.3569	303608.8	3.2493	303608.8	3.2088	599295	6.7189	365377.84	5.9462
2800	1000000	8.8195	1000000	8.341	307894.12	3.2842	307894.12	3.2199	631238.44	7.3478	382691	6.1807
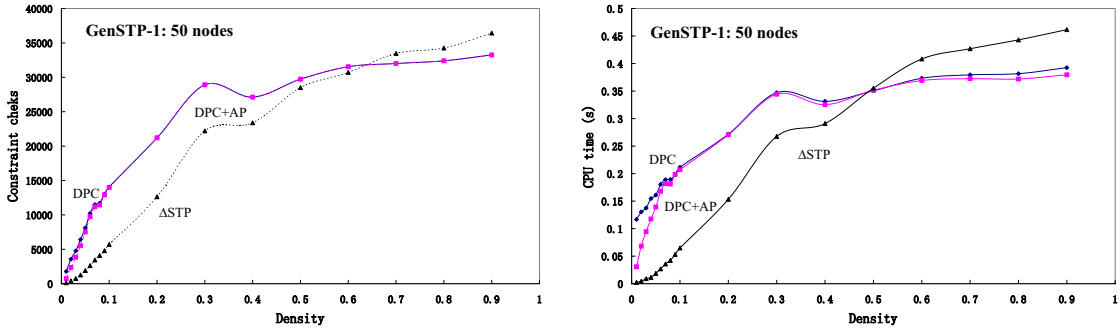
Figure 8. *Constraint Checks (left) and CPU time (right) for* DPC, DPC+AP, *and* △STP.

experiments on large problems, shown in Figure 9 and 10, demonstrate that △STP is the absolute winner over all algorithms. A comparison of Figure 9 and 10 shows that △STP, like PPC, is sensitive to the structure of the temporal graph (i.e., the existence of a cycle). It is more effective on problems generated with GenSTP-2 than on those generated with SPRAND.

5.3 Significance of our results

In practice, most *real-world* applications exhibit typically STPs with large size and low density [2]. The performance of an STP solver in these situations becomes extremely important. △STP is perfect for this kind of job. Its outstanding performance under low density is particularly advantageous and makes it the best algorithm developed to date. Further, when solving a TCSP with search, the STP examined at each node in the search tree is a subgraph

of the original TCSP and thus has a lower density than the TCSP. This supports the importance of an efficient STP solver for low density networks. We expect the combination of △STP with a TCSP solver to improve dramatically the performance of current TCSP solvers.

6 Conclusion and future work

We introduced △STP, a new efficient algorithm for solving the STP. Our algorithm advantageously exploits previous results reported in the literature and binds them via a new strategy for constraint propagation based on triangles. We demonstrated that this algorithm outperforms all previous ones in terms of pruning power and performance. We are currently integrating our new STP solver with a TCSP solver to improve the performance of the latter. More importantly, △STP solver provides us
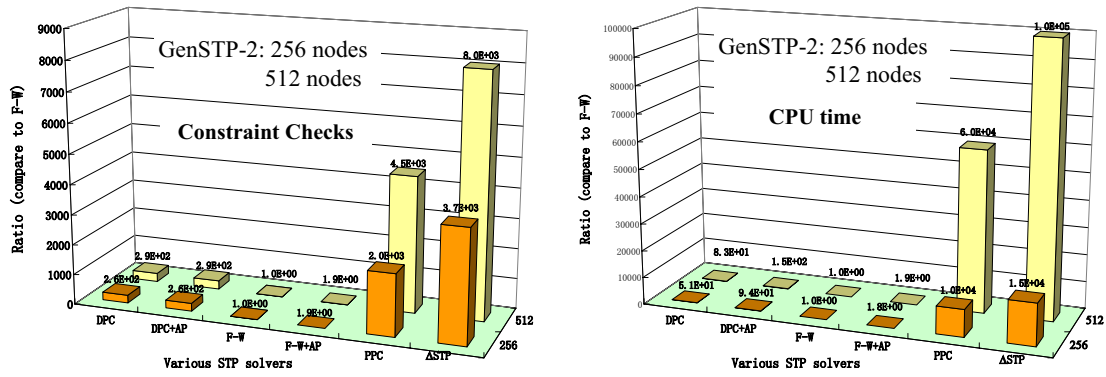
Figure 9. *Constraint Checks (left) and CPU time (right) for STP solvers*, problems generated by `GenSTP-2`.
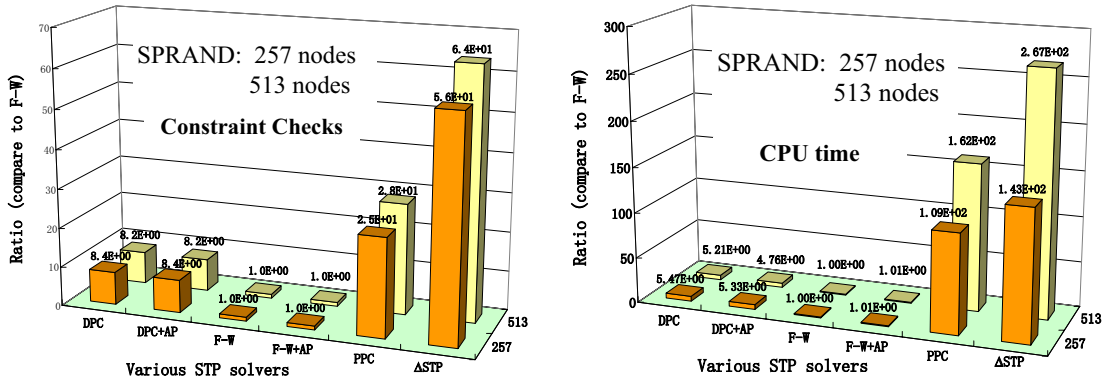

Figure 10. *Constraint Checks (left) and CPU time (right) for STP solvers*, problems generated by `SPRAND`.

with a new perspective on temporal problems as composed by a set of triangles, where two triangles are connected if and only if they have one common edge. Constraint propagation can be carried out according to this new graph of triangles. We are exploiting this idea to improve the search performance of the TCSP solver.

References

[1] Christian Bliek and Djamilla Sam-Haroud. Path Consistency for Triangulated Constraint Graphs. In *Proc. of the 16*[th] *IJCAI*, pages 456–461, Stockholm, Sweden, 1999.

[2] Mark Boddy. Personal communication, 2002.

[3] Amedeo Cesta, Angelo Oddi, and Stephen Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, April 2002.

[4] Boris V. Cherkasskyn, Andrew V. Goldberg, and Tomasz Radzik. Shortest Paths Algorithms: Theory and Experimental Evaluation. *Mathematical Programming*, 73:129–174, 1996.

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

[6] Thomas Dean and Drew McDermott. Temporal Data Base Management. *Artificial Intelligence*, 32:1–55, 1987.

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

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

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

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

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

[12] Nicolas Muscettola, Paul Morris, and Ioannis Tsamardinos. Reformulating Temporal Plans for Efficient Execution. In *Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 444–452, Trento Italy, 1998.

[13] U. Kjærulff. Triagulation of Graphs - Algorithms Giving Small Total State Space. Research Report R-90-09, Aalborg University, Denmark, 1990.

[14] Ioannis Tsamardinos. Reformulating Temporal Plans for Efficient Excution. Master's thesis, Intelligent Systems Program, University of Pittsburgh, 1998.