P^3C: A New Algorithm for the Simple Temporal Problem

Léon Planken and **Mathijs de Weerdt**
Delft University of Technology
The Netherlands

Roman van der Krogt
Cork Constraint Computation Centre
Ireland

Abstract

The Simple Temporal Problem (STP) is a sub-problem of almost any planning or scheduling problem involving time constraints. An efficient method to solve the STP, called $\triangle$STP (Xu and Choueiry 2003), is based on partial path consistency and starts from a chordal constraint graph. In this paper, we analyse this algorithm and show that there exist instances for which its time complexity is quadratic in the number of triangles in the constraint graph. We propose a new algorithm, P^3C, whose worst-case time complexity is linear in the number of triangles. We show both formally and experimentally that P^3C outperforms $\triangle$STP significantly.

Introduction

The Simple Temporal Problem (STP) was first proposed by Dechter et al. (1991). The input to this problem is a set of (time) variables with constraints that bound the difference between pairs of these variables. The solution then is (an efficient representation of) the set of all possible assignments to these variables that meet these constraints, or the message that such a solution does not exist.

The STP has received widespread attention that still lasts today, with applications in such areas as medical informatics (Anselma et al. 2006), spacecraft control and operations (Fukunaga et al. 1997), and air traffic control (Buzing and Witteveen 2004). Moreover, the STP appears as a pivotal subproblem in more expressive formalisms for temporal reasoning, such as the Temporal Constraint Satisfaction Problem—proposed by Dechter et al. in conjunction with the STP—and the Disjunctive Temporal Problem (Stergiou and Koubarakis 2000). For our purposes, we will focus on the importance of the STP for temporal planning (Nau, Ghallab, and Traverso 2004, Chapter 15).

Traditionally, planning has been concerned with finding out *which* actions need to be executed in order to achieve some specified objectives. Scheduling was confined to finding out *when* the actions need to be executed, and using *which* resources. A typical architecture exploiting this separation is presented in Figure 1: the first step separates the causal information from the temporal information, resulting in a classical planning problem that is solved with a classical planner. This is then combined with the temporal information that was removed during the first step to produce a temporal plan. This approach is for example taken by McVey et al. (1997).[*]

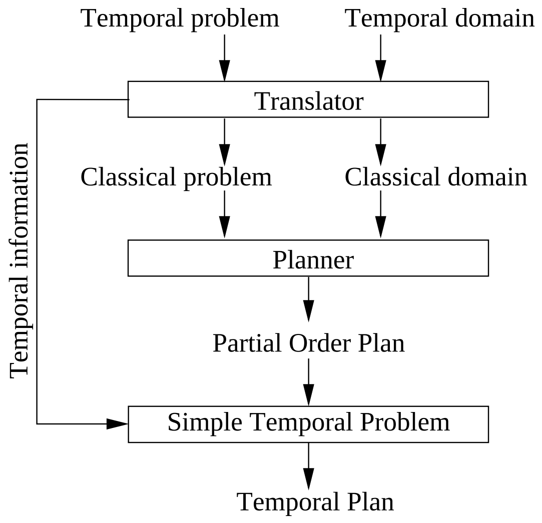

Figure 1: A typical architecture separating planning and scheduling. Adapted from (Halsey, Long, and Fox 2004).

However, in reality, planning and scheduling are intertwined: time constraints affect which actions may be chosen, and also how they should be combined. In such cases the architecture of Figure 1 may not produce valid plans in all occasions. The Crikey planner (Halsey, Long, and Fox 2004) circumvents this problem by identifying which parts are separable, and which are not. In the parts that are non-separable the causal and temporal problems are solved together; the separable parts are treated as separate problems.

Whereas Crikey tries to decouple planning and scheduling as much as possible, other people have taken an iterative approach. Both (Myers and Smith 1999) and (Garrido and Barber 2001) discuss this type of approach in general. A specific example of the approach is Machine^T (Castillo, Fdez-Olivares, and González 2002). Machine^T is a partial-order causal link planner that interleaves causal planning

[*]Application of the architecture sketched here is not limited to scheduling over time; Srivastava, Kambhampati, and Do (2001) propose a similar architecture to schedule over the available resources.

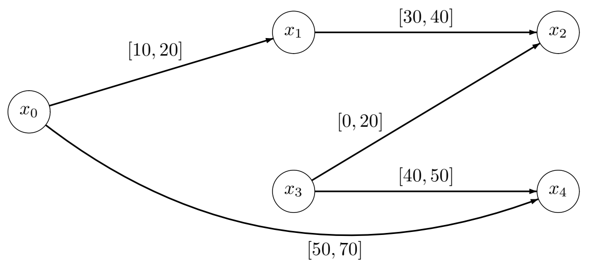

Figure 2: An example STP instance $\mathcal{S}$

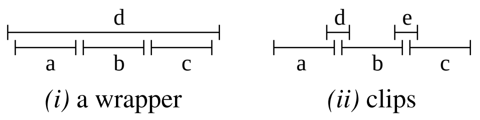

Figure 3: Two ways to encode time constraints. The *wrapper* (i) encodes a time window within which the actions a, b and c have to take place. Alternatively, constraints on the interval allowed between two actions can be encoded using *clips* (ii). Adapted from (Cresswell and Coddington 2003).

with temporal reasoning. The temporal reasoning module works on an STP instance that represents the partial plan of the causal planner. It propagates time constraints, can be used to identify threats by finding actions that possibly overlap in time, and ensures consistency. The same approach is taken by VHPOP (Younes and Simmons 2003).

Especially in the latter iterative approaches, the efficiency with which STPs allow temporal information to be dealt with is an important feature. For large problems, millions of partial plans may be generated, and for each of those the STP has to be updated and checked for consistency. In order not to become a bottle-neck, the STP has to be solved very quickly. The best known algorithm for STPs is called $\triangle$STP (pronounced triangle-STP) (Xu and Choueiry 2003). It requires that the network be represented by a *chordal* graph. Once a chordal graph is obtained (for which efficient heuristics exist), $\triangle$STP may require time quadratic in the number of triangles in the graph, as we will prove below. Analysis of this proof allows us to develop a new algorithm, called P^{3}C, which is *linear* in the number of triangles. This P^{3}C algorithm and the analysis of $\triangle$STP are the main contributions of this paper.

The remainder of the text is organised as follows. First, we formally discuss the Simple Temporal Problem and describe the existing solution techniques, including $\triangle$STP. Then, we analyse $\triangle$STP and propose our new algorithm P^{3}C. After experimentally validating our method, we conclude.

The Simple Temporal Problem

In this section, we briefly introduce the Simple Temporal Problem (STP); for a more exhaustive treatment, we refer the reader to the seminal work by Dechter, Meiri, and Pearl (1991).

An STP instance $\mathcal{S}$ consists of a set $X = \{x_1, \ldots, x_n\}$ of time-point variables representing events, and a set C of m constraints over pairs of time points, bounding the time difference between events. Every constraint $c_{i\to j}$ has a weight $w_{i\to j} \in \mathbb{R}$ corresponding to an upper bound on the time difference, and thus represents an inequality $x_j - x_i \leq w_{i\to j}$. Two constraints $c_{i\to j}$ and $c_{j\to i}$ can be combined into a single constraint $c_{i\leftrightarrow j} : -w_{j\to i} \leq x_j - x_i \leq w_{i\to j}$ or, equivalently, $x_j - x_i \in [-w_{j\to i}, w_{i\to j}]$, giving both upper and lower bounds. An unspecified constraint is equivalent to a constraint with an infinite weight; therefore, if $c_{i\to j}$ exists and $c_{j\to i}$ does not, we have $c_{i\leftrightarrow j} : x_j - x_i \in [-\infty, w_{i\to j}]$.

The following planning problem will be used to illustrate how an STP may arise as a sub-problem from a planning problem. The resulting STP in this case is similar to the one provided in (Dechter, Meiri, and Pearl 1991).

Example. *The example deals with planning for an industrial environment, including rostering and production planning. For the purpose of this example, consider a situation in which large parts of the plan have been computed already. To get a fully working plan, all that remains to be done is to ensure that at any time an operator is assigned to monitor the* **casting** *process. However, as* **casting** *is a relatively safe process, it can be left unattended for at most 20 minutes (between operator shifts). Moreover, the monitoring room is small, and has room for just one person.*

There are two operators available that can be assigned to the **casting** *task: John and Fred. Today, Fred's shift must end between 7:50 and 8:10. However, before he leaves, he has to attend to some paperwork, which takes 40–50 minutes. John is currently assigned to another task that ends between 7:10 and 7:20. As it is located on the far side of the plant, it will take him 30–40 minutes to make his way down to the casting area.*

The planning software generates a partial plan that lets Fred start his shift at the **casting** *process, with John taking over after Fred leaves for his paperwork. The question now is: is this a viable plan, and if so, what are the possible times to let Fred stop, and let John start monitoring the* **casting** *process?*

To see how a Simple Temporal Problem can be used to find the answer to our question, we first have to assign time-point variables to each event in the plan: let x_1 and x_2 represent John leaving his previous activity and arriving at the **casting** process, respectively; and let x_3 and x_4 denote Fred starting his paperwork, and finishing it. A *temporal reference point*, denoted by x_0, is included to enable us to refer to absolute time. For our example, it is convenient to take x_0 to stand for 7:00. If we represent all time intervals in minutes, the graph representation of the STP for this example is given in Figure 2; here, each vertex represents a time-point variable, and each edge represents a constraint. When represented as a graph in this way, the STP is also referred to as a Simple Temporal Network (STN); however, these terms are often used interchangeably.

It should be clear that the problem can arise from a PDDL planning description of the overall problem. Durative actions with variable durations can be used to model the various activities and a "wrapper" or a "clip" (Cresswell and

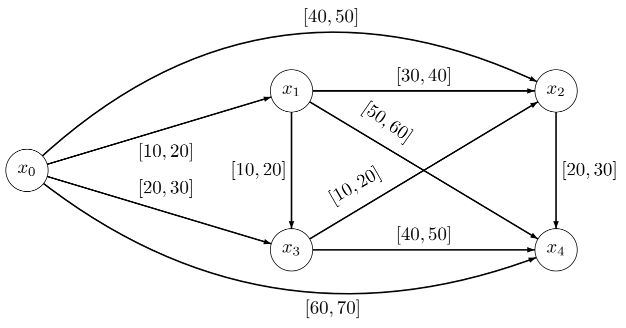

Figure 4: The minimal network $\mathcal{M}$ corresponding to $\mathcal{S}$

Coddington 2003) can be used to ensure that the process does not go without an operator for more than 20 minutes (see Figure 3).

Note that intervals labelling constraints can be used to represent both freedom of choice for the actors (e.g. John's departure from his previous activity) as well as uncertainty induced by the environment (e.g. travelling time). The approach presented in this paper applies to both interpretations; however, uncertainties can only be dealt with at plan execution time, see e.g. (Vidal and Bidot 2001).

A *solution* to the STP instance is an assignment of a real value to each time-point variable such that the differences between each constrained pair of variables fall within the range specified by the constraint. For example, the reader can verify both from the network and from the original plan that $\langle x_0 = 0,\ x_1 = 10,\ x_2 = 40,\ x_3 = 20,\ x_4 = 70\rangle$ is a solution to our example STP. Note that many solutions exist; to capture all of them, we are interested in calculating an equivalent *decomposable* STP instance, from which all solutions can then be extracted in a backtrack-free manner. The traditional way to attain decomposability is by calculating the *minimal network* $\mathcal{M}$. In $\mathcal{M}$, all constraint intervals have been tightened as much as possible without invalidating any solutions from the original instance.

The minimal network of our example is depicted in Figure 4. Note that some previously existing constraints have been tightened; also, since the graph is now complete, new information has become available. In the remainder of this text, we will not concern ourselves with finding individual solutions, but focus on finding decomposable networks such as $\mathcal{M}$ from which such solutions can be efficiently extracted. Such a network of minimal constraints is especially useful when further additions of constraints are expected, such as in the process of constructing a plan, because checking whether a new constraint is consistent with the existing ones can be done in constant time. Subsequently adapting the constraints in a constraint network may cost more time. Developing an efficient incremental algorithm to this end is outside the scope of this paper, but an interesting and important topic for further study.

Previous Solution Techniques

Dechter et al. noted (1991) that the Floyd-Warshall all-pairs-shortest-paths (APSP) algorithm can be used to compute the minimal network $\mathcal{M}$. Floyd-Warshall is simple to implement and runs in $\mathcal{O}\left(n^3\right)$ time, where n is the number of time-point variables in the STP instance. It corresponds to enforcing the property of path consistency (PC) (Montanari 1974), known from constraint satisfaction theory.

Algorithm 1: DPC

```
  Input: An STN instance $\mathcal{S} = \langle V, E\rangle$ and an ordering
         $d = (v_n, v_{n-1}, \ldots, v_1)$
  Output: CONSISTENT or INCONSISTENT
1 for $k \leftarrow n$ to $1$ do
2     forall $i, j < k$ such that $\{i, k\}, \{j, k\} \in E$ do
3         $w_{i\to j} \leftarrow \min(w_{i\to j}, w_{i\to k} + w_{k\to j})$
4         if $w_{i\to j} + w_{j\to i} < 0$ then
5             return INCONSISTENT
6         end
7     end
8 end
9 return CONSISTENT
```

For checking whether an STP instance is consistent, Dechter et al. proposed directed path consistency (DPC), which we include as Algorithm 1. The algorithm iterates over the variables along some ordering d. After iteration k, there exists an edge $\{v_i, v_j\}$, possibly added in line 3, for every pair of nodes that are connected by a path in the subgraph induced by $\{v_k \in V \mid k > \max(i, j)\} \cup \{v_i, v_j\}$; moreover, this edge is labelled by the shortest path in that subgraph. This implies in particular that $c_{1\leftrightarrow 2}$ (if it exists) is minimal. The algorithm runs in time $\mathcal{O}\left(n \cdot (w^*(d))^2\right)$, where $w^*(d)$ is a measure called the *induced width* relative to the ordering d of the vertices in the constraint graph. For $d = (v_n, v_{n-1}, \ldots, v_1)$, we have

$$w^*(d) \ = \ \max_i |\{\{v_i, v_j\} \in E \ \mid \ j < i\}|$$

Since $w^*(d)$ is at most equal to $n-1$ and may be much smaller, DPC is more efficient than Floyd-Warshall; however, it does not in general produce a decomposable STP.

In 1999, Bliek and Sam-Haroud proposed the property of *partial path consistency* (PPC) for constraint satisfaction problems, which is defined on an (undirected) chordal constraint graph instead of the complete constraint graph as is PC. Since chordal graphs play an important role in this paper, we give the definition here.

Definition 1. *Let $G = \langle V, E\rangle$ be an undirected graph. If $(v_1, v_2, \ldots, v_k, v_{k+1} = v_1)$ with $k > 3$ is a cycle, then any edge on two nonadjacent vertices $\{v_i, v_j\}$ with $1 < j - i < k - 1$ is a* chord *of this cycle. G is* chordal *(also ambiguously called "triangulated") if every cycle of length greater than 3 has a chord.*[*]

Chordal graphs generally contain far less edges than complete graphs; this holds especially if the graph was originally sparse. Hence, enforcing PPC is often far cheaper than is enforcing PC. Bliek and Sam-Haroud further proved that, for

[*]The term "triangulated graph" is also used for maximal planar graphs, which do not concern us here.

Algorithm 2: $\triangle$STP

```
Input: A chordal STN $\mathcal{S} = \langle V, E \rangle$
Output: The PPC network of $\mathcal{S}$ or INCONSISTENT
$Q \leftarrow$ all triangles in $\mathcal{S}$
while $Q \neq \emptyset$ do
    choose $T \in Q$
    foreach permutation $(v_i, v_j, v_k)$ of $T$ do
        $w_{i \to k} \leftarrow \min(w_{i \to k}, w_{i \to j} + w_{j \to k})$
        if $w_{i \to k}$ has changed then
            if $w_{i \to k} + w_{k \to i} < 0$ then
                return INCONSISTENT
            end
            $Q \leftarrow Q \cup \{$all triangles $\hat{T}$ in $\mathcal{S} \mid v_i, v_k \in \hat{T}\}$
        end
    end
    $Q \leftarrow Q \setminus T$
end
```

convex problems, PPC produces a decomposable network just like PC does.

Xu and Choueiry (2003) realised that the STP is a convex problem, since each of its constraints is represented by a single interval. Enforcing PPC on an STP instance then corresponds to calculating the minimal labels for each edge in the constraint graph, disregarding directionality. They implemented an efficient version of the PPC algorithm, called $\triangle$STP, which we include as Algorithm 2.

The trade-off for the efficiency gained by enforcing PPC is that less information is produced: whereas PC yields minimal constraints between *each* pair of time-point variables, PPC only labels the edges in the chordal graph with minimal constraints. However, it can easily be ensured that all information of interest is produced by adding new edges to the constraint graph before triangulation. For this reason, enforcing PPC can truly be seen as PC's replacement for solving the STP.

Xu and Choueiry show empirically that the $\triangle$STP outperforms the original PPC algorithm by Bliek and Sam-Haroud, but they do not give a formal analysis of the time complexity of $\triangle$STP. In the next section we give an analysis, showing that there exist problem instances for which the time complexity of $\triangle$STP is quadratic in the number of triangles in the chordal constraint graph.

Worst-Case Analysis of $\triangle$STP

Giving a tight bound on $\triangle$STP's time complexity in terms of the number of variables or constraints in the STP instance is not straightforward. However, looking at Algorithm 2, it can be seen that the time complexity mainly depends on the number of triangles t in the chordal STN that is the input of the algorithm. In a way, this number t represents the difficulty of the problem at hand. Note that there is no fixed relation between t and the size of the input except that there is an upper bound of $\mathcal{O}(n^3)$, attained for a complete graph.

In this section we give a class $\mathcal{P}$ of pathological STP instances for which the time complexity of $\triangle$STP is quadratic in the number of triangles. These instances have a constraint graph consisting of a single directed cycle with all zero weight edges; this cycle is filled with edges having carefully selected weights. Before formally defining class $\mathcal{P}$, we include instance $\mathcal{P}_6$ as an example in Figure 5; its triangles are labelled τ_1 through τ_6 for ease of reference.

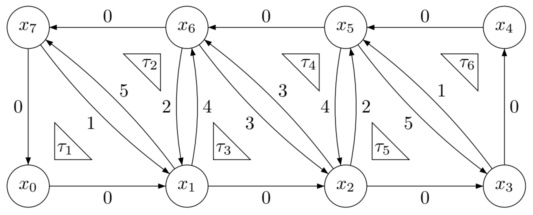

Figure 5: Pathological test case $\mathcal{P}_6$ for $\triangle$STP

path	weight
$x_0 \to x_7$	∞
$x_0 \to x_1 \to x_7$	5
$x_0 \to x_1 \to x_6 \to x_7$	4
$x_0 \to x_1 \to x_2 \to x_6 \to x_7$	3
$x_0 \to x_1 \to x_2 \to x_5 \to x_6 \to x_7$	2
$x_0 \to x_1 \to x_2 \to x_3 \to x_5 \to x_6 \to x_7$	1
$x_0 \to x_1 \to x_2 \to x_3 \to x_4 \to x_5 \to x_6 \to x_7$	0

Table 1: Total weights of paths in $\mathcal{P}_6$

Definition 2. *The class $\mathcal{P}$ consists of pathological STP instances $\mathcal{P}_t$ on t triangles for every $t \in \mathbb{N}$. Each instance $\mathcal{P}_t$ is defined on $t + 2$ variables $\{x_0, x_1, \ldots, x_{t+1}\}$ and has the following constraints (where x_{t+2} wraps around to x_0):*

- $\{c_{i \to i+1} \mid 0 \leq i \leq t + 1\}$ *with zero weight;*
- $\{c_{i \to j} \mid 1 \leq i \leq j - 2 < t \ \wedge\ i + j - t \in \{1, 2\}\}$ *with weight $j - i - 1$;*
- $\{c_{j \to i} \mid 1 \leq i \leq j - 2 < t \ \wedge\ i + j - t \in \{1, 2\}\}$ *with weight $t - (j - i - 1)$.*

None of the constraints in these instances are initially minimal, except for the ones with zero weight; because there exists a zero-weight path between every pair of vertices, a constraint is minimal if and only if its weight is zero. Furthermore, these instances are defined in such a way that longer paths have progressively lower total weights, as shown in Table 1 for the case of $\mathcal{P}_6$.

Theorem 1. *When solving $\mathcal{P}_t$, the $\triangle$STP algorithm may require processing $\Omega(t^2)$ triangles.*

Proof. Assume that the initial queue of triangles is $Q = (\tau_1, \tau_2, \ldots, \tau_t)$. First, triangle τ_1 is processed, leading to the adjustment of $w_{0 \to t+1}$ from infinity down to $t - 1$, and $w_{t+1 \to 1}$ from 1 to 0. Triangle τ_2 would be added to Q, but is already contained therein. Now, triangle τ_2 is processed, and $w_{1 \to t+1}$ has its weight decreased from $t - 1$ to $t - 2$, which causes triangle τ_1 to be appended to the queue; also, $w_{t \to 1}$ is set to 0. This process is repeated until triangle τ_t has been processed; at this point, all edges making up τ_t have minimal weights, and we have that $Q = (\tau_1, \ldots, \tau_{t-1})$.

The algorithm starts again at triangle τ_1 and proceeds to triangle τ_{t-1}, after which $Q = (\tau_1, \ldots, \tau_{t-2})$. By now, the pattern is clear: the total number of triangles processed is $t + (t-1) + \cdots + 1 = t(t+1)/2$, which is indeed quadratic in t. □

These cases make it clear that the sequence in which $\triangle$STP processes the triangles in a constraint graph is not optimal. It may be worthwhile to explore if there is a natural way to order triangles in a chordal graph. For this reason, we now turn to some theory concerning chordal graphs.

Graph Triangulation

In this section, we list some definitions and theorems from graph theory which underlie the new algorithm we propose in the next section. These results are readily available in graph-theoretical literature, e.g. (West 1996).

Definition 3. *Let $G = \langle V, E\rangle$ be an undirected graph. We can define the following concepts:*

- *A vertex $v \in V$ is* simplicial *if the set of its neighbours $N_v = \{w \mid \{v, w\} \in E\}$ induces a clique, i.e. if $\forall \{s, t\} \subseteq N_v : \{s, t\} \in E$.*
- *Let $d = (v_n, \ldots, v_1)$ be an ordering of V. Also, let G_i denote the subgraph of G induced by $V_i = \{v_1, \ldots, v_i\}$; note that $G_n = G$. The ordering d is a* simplicial elimination ordering *of G if every vertex v_i is a simplicial vertex of the graph G_i.*

We then have the following (known) result:

Theorem 2. *An undirected graph $G = \langle V, E\rangle$ is chordal if and only if it has a simplicial elimination ordering.*

In general, many simplicial elimination orderings exist. Examples of such orderings for the graph depicted in Figure 5 are $(x_0, x_7, x_1, x_6, x_2, x_5, x_3, x_4)$ and $(x_4, x_0, x_3, x_5, x_7, x_2, x_6, x_1)$.[*]

Chordality checking can be done efficiently in $\mathcal{O}(n + m)$ time (where $n = |V|$ and $m = |E|$) by the maximum cardinality search algorithm. This algorithm iteratively labels vertices with the most labeled neighbours, whilst checking that these neighbours induce a clique. At the same time, it produces (in reverse order) a simplicial elimination ordering if the graph is indeed chordal.

If a graph is not chordal, it can be made so by the addition of a set of *fill edges*. These are found by eliminating the vertices one by one and connecting all vertices in the neighbourhood of each eliminated vertex, thereby making it simplicial; this process thus constructs a simplicial elimination ordering as a byproduct. If the graph was already chordal, following its simplicial elimination ordering means that no fill edges are added. In general, it is desirable to achieve chordality with as few fill edges as possible.

Definition 4 (Kjærulff)**.** *Let $G = \langle V, E\rangle$ be an undirected graph that is not chordal. A set of edges T with $T \cap E = \emptyset$ is called a* triangulation *if $G' = \langle V, E \cup T\rangle$ is chordal. T is* minimal *if there exists no subset $T' \subset T$ such that T' is a triangulation. T is* minimum *if there exists no triangulation T' with $|T'| < |T|$.*

[*]Like Xu and Choueiry, we disregard graph directionality when discussing chordal STNs.

Algorithm 3: P^{3}C

```
Input: A chordal STN $\mathcal{S} = \langle V, E\rangle$ with a simplicial
       elimination ordering $d = (v_n, v_{n-1}, \ldots, v_1)$
Output: The PPC network of $\mathcal{S}$ or INCONSISTENT
1 call DPC($\mathcal{S}, d$)
2 return INCONSISTENT if DPC did
3 for $k \leftarrow 1$ to $n$ do
4     forall $i, j < k$ such that $\{i, k\}, \{j, k\} \in E$ do
5         $w_{i\to k} \leftarrow \min(w_{i\to k}, w_{i\to j} + w_{j\to k})$
6         $w_{k\to j} \leftarrow \min(w_{k\to j}, w_{k\to i} + w_{i\to j})$
7     end
8 end
9 return $\mathcal{S}$
```

Determining a minimum triangulation is an NP-complete problem; in contrast, a (locally) minimal triangulation can be found in $\mathcal{O}(nm)$ time (Kjærulff 1990). Since finding the smallest triangulations is so hard, several heuristics have been proposed for this problem. Kjærulff has found that both the minimum fill and minimum degree heuristics produce good results. The *minimum fill* heuristic always selects a vertex whose elimination results in the addition of the fewest fill edges; it has worst-case time complexity $\mathcal{O}(n^2)$. The *minimum degree* heuristic is even simpler, and at each step selects the vertex with the smallest number of neighbours; its complexity is only $\mathcal{O}(n)$, but its effectiveness is somewhat inferior to that of the minimum fill heuristic.

The New Algorithm

Given a chordal graph with t triangles, the best known method for enforcing partial path consistency (PPC) is the $\triangle$STP algorithm. As we demonstrated in Theorem 1, this algorithm may exhibit time complexity $\Omega(t^2)$. In this section we propose an algorithm that instead has time complexity $\mathcal{O}(t)$: it enforces PPC by processing every triangle exactly twice. To achieve this result, the simplicial elimination ordering d must be known; as we stated in the previous section, this is a byproduct of triangulation. Our new algorithm is called P^{3}C and is presented as Algorithm 3. It consists of a forward and backward sweep along d. The forward sweep is just DPC (see Algorithm 1); note that because d is simplicial, no edges will be added. We now state the main results of this paper.

Theorem 3. *Algorithm* P^{3}C *achieves PPC on consistent chordal STNs.*

Proof. Recall that for the STN, enforcing PPC corresponds to calculating the shortest path for every pair of nodes connected by an edge, i.e. calculating minimal constraints.

The algorithm first enforces DPC along d; thus, after this step, there exists an edge $\{v_i, v_j\}$ for every pair of nodes that are connected by a path in the subgraph induced by $\{v_k \in V \mid k > \max(i, j)\} \cup \{v_i, v_j\}$; moreover, this edge is labelled by the shortest path in that subgraph. This means in particular that $c_{1\leftrightarrow 2}$ (if it exists) is minimal.

It can now be shown by induction that after iteration k of the forward sweep (lines 3–8), all edges in the subgraph G_k induced by $\{v_i \in V \mid i \leq k\}$ are labelled with minimal constraint weights. The base case for $k \leq 2$ has already been shown to hold; assuming that the proposition holds for $k-1$, we show that it also holds for k. Consider any constraint $c_{k\to i}$ with $i < k$; by the induction hypothesis, we know that all constraints $c_{i\to j}$ with $i, j < k$ are already minimal. To arrive at a contradiction, assume that $c_{k\to i}$ is not minimal after the kth iteration; i.e. after the iteration completes, there still exists some path $\pi = (v_k \to v_{j_1} \to \cdots \to v_{j_l} \to v_i)$ with total weight $w_\pi < w_{k\to i}$. We show below that this cannot occur; by symmetry, this result then extends to $c_{i\to k}$.

If there is any part of π outside G_k, with endpoints u, w in G_k, then by the DPC property we can replace this part of π by the edge (u, w); assume therefore that π lies entirely within G_k. Now, since v_k appears in d before both its neighbours v_{j_1} and v_i, there must exist an edge $\{v_{j_1}, v_i\}$. By the induction hypothesis, $c_{j_1\to i}$ is minimal; the shortest path can thus be further reduced to $\pi' = (v_k \to v_{j_1} \to v_i)$. But then, we have that $w_{k\to i} \leq w_{\pi'} = w_{k\to j_1} + w_{j_1\to i} \leq w_\pi$ by the operations performed in the kth iteration, which contradicts our assumption. □

Theorem 4. *Algorithm* P^{3}C *enforces PPC in time* $\Theta(t)$, *where* t *is the number of triangles in the (chordal) STN. If the instance is inconsistent, this is discovered in time* $\mathcal{O}(t)$.

Proof. The first part of the algorithm is just the directed path consistency algorithm; if the problem is inconsistent, this is discovered in this phase. After this first leg, every triangle has been visited exactly once. The backward sweep (lines 3–8) then follows the same ordering backwards and again visits every triangle exactly once. Since each visit of a triangle takes constant time, our claims on time complexity easily follow. □

Upper bounds on the time complexity can also be expressed in other parameters:

Corollary 5. *The* P^{3}C *algorithm establishes PPC or finds inconsistency in time* $\mathcal{O}\left(n \cdot (w^*)^2\right) \subseteq \mathcal{O}\left(n\delta^2\right) \subseteq \mathcal{O}\left(n^3\right)$. *Here,* w^* *is the minimal induced width of the graph along any ordering, and* δ *is the maximum degree of any vertex in the graph.*

Proof. It is clear that to within a constant factor, the time complexity of P^{3}C is identical to that of DPC. Since it holds that for a chordal graph the induced width is minimal along a simplicial elimination ordering (Dechter 2003, p. 90), the first bound follows. The latter two bounds follow from the observation that $w^* \leq \delta \leq n$. □

Experimental Results

In this section, we evaluate our new P^{3}C algorithm against $\triangle$STP. We ran three types of test cases: (i) pathological instances, as described in Definition 2; (ii) STPs representing subproblems of job-shop problem instances; and (iii) the job-shop instances with enforced consistency. In our theoretical analyses above, the number of triangles in the constraint graph turned out to be of paramount importance. For this reason, we set out this parameter on the horizontal axis of our graphs, instead of the number of vertices. The relation between the numbers of vertices and triangles in our test cases is depicted in Figure 6; in this figure, we also delineate the theoretical maximum number of triangles (in case of a complete graph). Note that the job-shop benchmark set exhibits a jump in the number of triangles for instances bigger than 190 vertices, whereas the amount of edges continued to increase gradually.

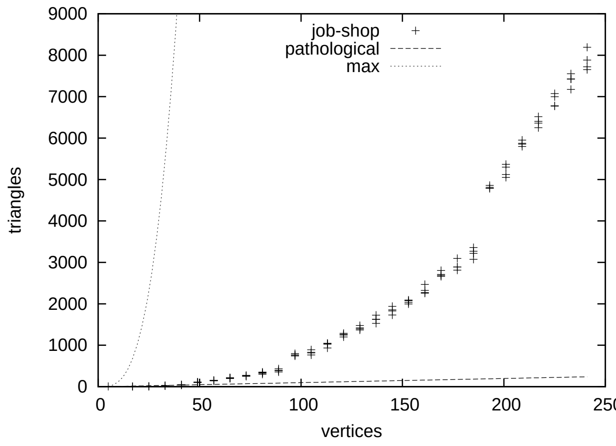

Figure 6: Relation between the numbers of vertices and triangles in our test sets

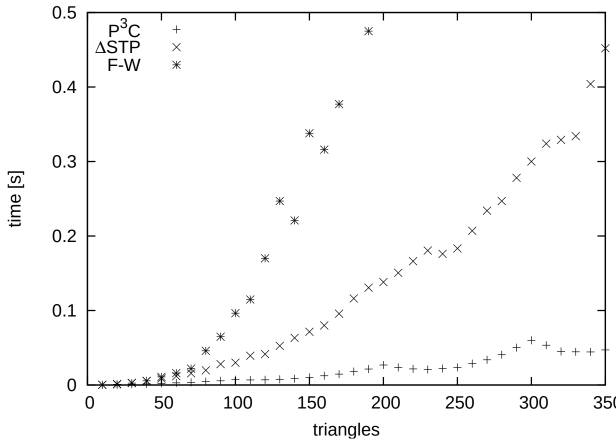

Figure 7: Performance of Floyd-Warshall, $\triangle$STP and P^{3}C on the pathological case

We implemented the algorithms in Java and ran them on a 2.4 GHz AMD Opteron machine with 4 GB of memory. We measured the time required to complete a run of the algorithms proper; i.e., the triangulation of the constraint graphs was excluded from the measurements.

For the evaluation of the pathological cases from Definition 2, we opted to include the Floyd-Warshall APSP algo-

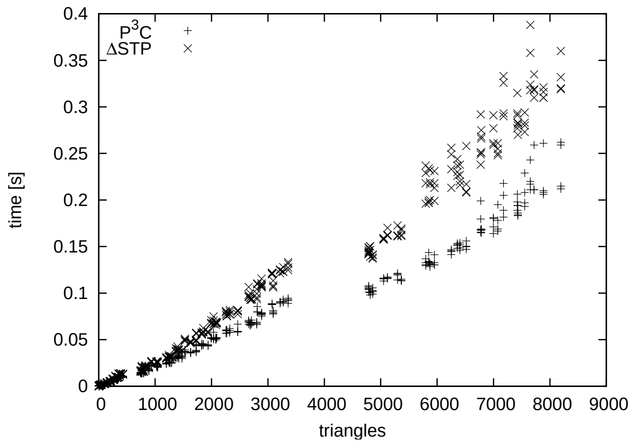

Figure 8: Performance of $\triangle$STP and P^{3}C on job-shop benchmarks

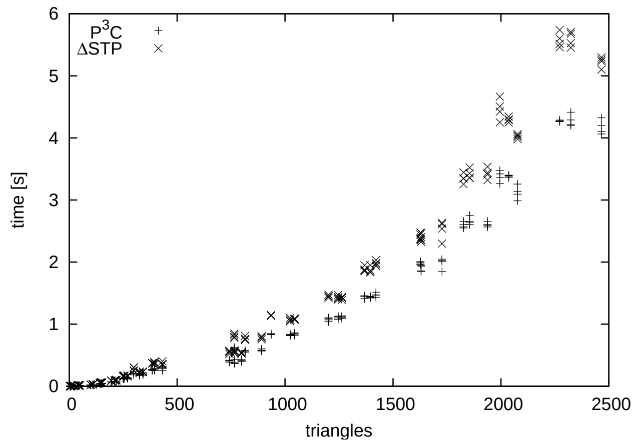

Figure 9: Performance of $\triangle$STP and P^{3}C on job-shop benchmarks with enforced consistency

rithm. The results are included in Figure 7. For this specifically crafted category of problem instances, in which the number of triangles is directly proportional to the number of vertices, the performance of the three algorithms closely follows the theoretical analysis. Floyd-Warshall exhibits performance cubic in the number of triangles, $\triangle$STP is quadratic, and our new P^{3}C algorithm remains linear in the amount of triangles, as was proven above.

The instances of the job-shop problem we considered are taken from the SMT-LIB benchmark library (Ranise and Tinelli 2003). In effect, these are instances of the NP-hard Disjunctive Temporal Problem (DTP): each constraint is a disjunction of temporal inequalities. We generated a set of STP instances by randomly selecting a single temporal inequality from each disjunction. If such a *component STP* were found to be consistent, it would constitute a solution to the job-shop problem; combined with the size of the DTP instances, this makes it highly unlikely that a random selection yields a consistent STP instance. Nevertheless, these instances can be considered representative of the types of problems that an STP solver may be expected to deal with as part of a planning algorithm that can deal with time.

In Figure 8, we include the results of running P^{3}C and $\triangle$STP on STP instances generated in this fashion. The jump that was already noticed in Figure 6 reappears in this figure. Note, however, that across this jump the run time does not keep pace with the number of triangles. Clearly, the number of triangles in the problem instance is not the only factor that determines problem complexity; recall that the number of vertices and constraint edges only gradually increase over the gap. We also observe that $\triangle$STP does not display a run time that is quadratic in the number of triangles for these problem instances; as a consequence, the difference between the results is much closer. However, P^{3}C can still clearly be seen to outperform its predecessor.

Running the algorithms on inconsistent instances is not a full test of their prowess, because inconsistency is generally discovered rather early along the way of calculating minimal constraints; in the case of P^{3}C, any inconsistency is guaranteed to be found within the first loop. For this reason, we randomly relabeled the constraint edges in the job-shop STP instances in such a way that consistency was guaranteed, and then ran the algorithms. The results of these tests are depicted in Figure 9. Calculating minimal constraints takes much more time than finding an inconsistency; for this reason, we only included the smaller problem instances (up to 2500 instead of 9000 triangles). Again, we can conclude that our P^{3}C algorithm consistently outperforms $\triangle$STP, though the results are somewhat closer than for the inconsistent STPs. Finally, note that the performance of P^{3}C is not exactly linear in the amount of triangles for these test cases. This is probably due to the overhead incurred by the used data structures (hash tables), which were implicitly assumed to have constant-time access in the theoretical analysis.

Conclusions and Future Research

We have taken a look at existing solution techniques for the Simple Temporal Problem, and identified a class $\mathcal{P}$ of problem instances for which the best existing algorithm $\triangle$STP exhibits a time complexity quadratic in the number of triangles. By visiting the triangles in a specific order in a new algorithm, called P^{3}C, we showed that we can solve STP within a time bounds linear in the number of triangles. This so-called simplicial elimination ordering is a byproduct of the triangulation phase, and therefore does not require any additional calculations compared to $\triangle$STP.

We corroborated our theoretical results with experiments that included instances from both $\mathcal{P}$ and a more realistic set of job-shop benchmarks. Throughout our experiments, P^{3}C clearly and consistently outperformed $\triangle$STP. Future work may evaluate the algorithms in a wider range of practical settings.

Sometimes, it suffices to just determine consistency of the STP. In these cases, instead of enforcing PPC, one can use the Bellman-Ford algorithm, which runs in $\mathcal{O}(nm)$ time. Although in the worst case this yields the same time complexity as P^{3}C, we have seen in a separate study that it is faster on some of our benchmark problems. However, P^{3}C also yields minimal constraints, which is very useful in the context of planning: here, the additional information about remaining slackness is often more important than just knowing whether the plan is consistent.

We expect that with little extra work, the results attained here extend to general constraint satisfaction problems. The original work on PPC by Bliek and Sam-Haroud had this wider scope, but in the proof of P^{3}C's soundness we chose to focus only on the STP, this being our subject of interest. Future research will have to determine whether the algorithm can indeed be applied in the context of general CSPs, which would mean that our work also becomes of interest to the CSP community.

Another line of future work comes from the observation that using P^{3}C, the worst-case complexity for establishing PPC is of the same order as the worst-case complexity for determining a minimal triangulation. It is worthwhile to investigate the trade-off between, on the one hand, faster triangulation (e.g. by relaxing the condition of minimality) implying worse run time of the P^{3}C algorithm, and, on the other hand, better but slower triangulation yielding faster run time of P^{3}C.

Finally, we plan to propose an *incremental* PPC algorithm in a future publication, building on the properties of chordal graphs already explored here. An incremental solver takes as input a partially path-consistent STP instance and a new constraint to be added or an existing constraint to be tightened; it then reports whether the new constraint still yields a consistent instance and, if so, enforces PPC again. This is especially useful for dealing with more complex (planning) problems which require solving many, progressively larger instances of the STP.

References

Anselma, L.; Terenziani, P.; Montani, S.; and Bottrighi, A. 2006. Towards a comprehensive treatment of repetitions, periodicity and temporal constraints in clinical guidelines. *Artificial Intelligence in Medicine* 38(2):171–195.

Bliek, C., and Sam-Haroud, D. 1999. Path consistency on triangulated constraint graphs. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence*, 456–461. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Buzing, P., and Witteveen, C. 2004. Distributed (re)-planning with preference information. In Verbrugge, R.; Taatgen, N.; and Schomaker, L., eds., *Proc. of the 16th Belgium-Netherlands Conf. on AI*, 155–162.

Castillo, L.; Fdez-Olivares, J.; and González, A. 2002. A temporal constraint network based temporal planner. In *Proc. of the 21st Workshop of the UK Planning and Scheduling Special Interest Group*, 99–109.

Cresswell, S., and Coddington, A. 2003. Planning with timed literals and deadlines. In *Proc. of the Twenty-Second Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG-03)*, 22–35.

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

Dechter, R. 2003. *Constraint Processing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Fukunaga, A.; Rabideau, G.; Chien, S.; and Yan, D. 1997. Aspen: A framework for automated planning and scheduling of spacecraft control and operations. In *Proc. of the Int. Symposium on AI, Robotics and Automation in Space*.

Garrido, A., and Barber, F. 2001. Integrating planning and scheduling. *Applied Artificial Intelligence* 15:471–491.

Halsey, K.; Long, D.; and Fox, M. 2004. CRIKEY – a temporal planner looking at the integration of planning and scheduling. In *Proc. of the ICAPS 2004 Workshop on Integrating Planning into Scheduling*, 46–52.

Kjærulff, U. 1990. Triangulation of graphs - algorithms giving small total state space. Technical report, Aalborg University.

McVey, C. B.; Durfee, E. H.; Atkins, E. M.; and Shin, K. G. 1997. Development of iterative real-time scheduler to planner feedback. In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence*, 1267–1272.

Montanari, U. 1974. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science* 7(66):95–132.

Myers, K., and Smith, S. 1999. Issues in the integration of planning and scheduling for enterprise control. In *Proc. of the DARPA Symposium on Advances in Enterprise Control*.

Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers.

Ranise, S., and Tinelli, C. 2003. The SMT-LIB format: An initial proposal. In *Proceedings of PDPAR'03*.

Srivastava, B.; Kambhampati, S.; and Do, M. B. 2001. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in realplan. *Artificial Intelligence* 131(1–2):73–134.

Stergiou, K., and Koubarakis, M. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120(1):81–117.

Vidal, T., and Bidot, J. 2001. Dynamic sequencing of tasks in simple temporal networks with uncertainty. In *CP 2001 Workshop in Constraints and Uncertainty*.

West, D. B. 1996. *Introduction to Graph Theory*. Prentice-Hall.

Xu, L., and Choueiry, B. Y. 2003. A new efficient algorithm for solving the Simple Temporal Problem. In *Proc. of the 10th Int. Symp. on Temporal Representation and Reasoning and 4th Int. Conf. on Temporal Logic*, 210–220. IEEE Computer Society.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of AI Research* 20:405–430.