Incrementally Solving the STP by Enforcing Partial Path Consistency

Léon Planken
Delft University of Technology, The Netherlands
l.r.planken@tudelft.nl

Abstract

The Simple Temporal Problem (STP) appears as a sub-problem of many planning and scheduling problems. Recently, we have published a new algorithm to solve the STP, called $\mathrm{P^3C}$, and have showed that it outperformed all existing algorithms. Our approach was based on enforcing partial path consistency (PPC), starting from a chordal constraint graph.

When dealing with complex problems, e.g. planning problems, dynamic situations, or the Disjunctive Temporal Problem, it is important to have efficient methods for building up and solving STP instances incrementally. In this paper, we present a brief survey of existing incremental algorithms and a new algorithm that incrementally enforces PPC; the performance of our new algorithm is experimentally compared against the existing ones.

Introduction

In planning and scheduling problems, an important role is played by time constraints. Planners may choose to maintain a selection of simple temporal constraints that must be satisfied in the ultimate plan; as search progresses, this collection of constraints grows until either a complete plan is found or an inconsistency is detected. Also, from a selection of temporal constraints, new information may be logically inferred and used to guide the search.

The Simple Temporal Problem (STP), proposed by Dechter, Meiri, and Pearl (1991), is defined as such a collection of simple temporal constraints, and efficient algorithms exist to decide whether the given constraints are consistent and to infer new information from them. Traditionally, the latter task of inferring new information was done by solving the all-pairs shortest paths (APSP) problem; the solution is then represented in a complete graph, which gives the inferred relations between all pairs of time points. It has been shown (Bliek and Sam-Haroud 1999; Xu and Choueiry 2003) that if one is only interested in a selection of this information, this can be inferred more efficiently by enforcing partial path consistency (PPC). We recently published an algorithm that used this approach, called $\mathrm{P^3C}$, and showed that it outperformed all other known approaches for calculating APSP or enforcing PPC (Planken, De Weerdt, and Van der Krogt 2008).

The contribution of this paper consists in a new algorithm for *incrementally* solving STP instances by enforcing partial path consistency (PPC): it takes an STP instance that is already partially path-consistent and a new constraint that is to be added to it. Incremental algorithms are of great interest for planners, because they gradually select more and more temporal constraints that must be satisfied. When adding a new constraint to such a collection, an incremental method can build upon the work that has already been done for the constraints currently in the collection, and can thus decide consistency and infer new information faster than single-shot approaches which start from scratch each time.

In this paper, we first formally introduce the STP and survey existing work on solving it, both as a single-shot approach and incrementally. Next, we give some graph-theoretical background which is required for our new IPPC algorithm, which we subsequently introduce. We theoretically analyse this algorithm, and test it against its competitors on two test cases. Finally, we give a brief overview of other related work, conclude, and give directions for future research.

The Simple Temporal Problem

In this section, we give a definition of the STP and state a motivating example, both based upon the seminal work by Dechter, Meiri, and Pearl (1991). Then, we examine what a solution to the STP might look like; for this, we also refer to publications by Bliek and Sam-Haroud (1999) and Xu and Choueiry (2003).

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 $c_{i\leftrightarrow j} : x_j - x_i \in [-\infty, w_{i\to j}]$.

We now give an example planning problem and illustrate how it may give rise to an STP instance as a sub-problem.

Example. *This example deals with planning for an industrial environment, including rostering and production plan-*

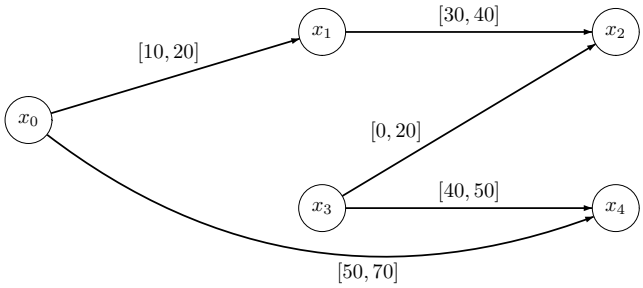

Figure 1: An example STN $\mathcal{S}$

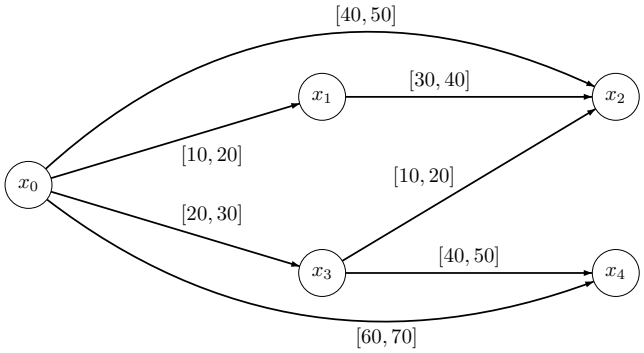

Figure 2: The partially path-consistent STN

ning. For the purposes 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 all times, an operator is assigned to monitor the **casting** *process. However, as* **casting** *is a relatively safe process, it can be left unattended for up to 20 minutes (between operator shifts). Furthermore, the monitoring room is small and can accommodate only a single person.*

Two operators are available to 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 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, at what times can Fred stop monitoring the* **casting** *process and John take it up again?*

We can encode this information into a Simple Temporal Problem; solving this will then answer 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 leaving the casting process for 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 values in minutes, a graph representation of the STP for this example is given in Figure 1. Here, each vertex denotes a time-point variable, and each edge denotes a constraint. When represented as a graph in this way, the STP is also referred to as a Simple Temporal Network (STN); however, note that 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 "clip" (Cresswell and Coddington 2003) can be used to ensure that the process does not go without an operator for more than 20 minutes. Note that intervals labelling constraints are used to represent both freedom of choice for the actors (e.g. John's departure from his previous activity) and uncertainty induced by the environment (e.g. travelling time).* It is up to the planner to decide how to act when an uncertainty interval is reduced (e.g. by having Fred rush through his paperwork).

A *solution* to an STP instance consists of 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. If an STP instance admits a solution, it is called *consistent*. 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.

In general, many solutions may exist; to capture them all, we are interested in calculating an equivalent *decomposable* STP instance, from which any solution can then be extracted in a backtrack-free manner. In the remainder of this text, we do not concern ourselves with finding individual solutions, but focus instead on finding decomposable networks. Traditionally, decomposability is attained by calculating the *minimal network* $\mathcal{M}$ corresponding to $\mathcal{S}$. $\mathcal{M}$ is a complete constraint graph in which all constraint intervals have been tightened as much as possible without invalidating any solutions from the original instance. If the STP is considered as a type of constraint satisfaction problem (CSP), calculating $\mathcal{M}$ is exactly equivalent to enforcing *path consistency* (PC) on it. On the other hand, when considered as a graph problem, calculating $\mathcal{M}$ corresponds exactly to calculating all-pairs-shortest-paths (APSP).

If the original STN was a sparse graph, the effort and memory requirements of calculating $\mathcal{M}$ are relatively large. For these cases, it is preferable to have a way of attaining decomposability that preserves the graph's sparseness as much as possible instead of having to calculate the complete graph of $\mathcal{M}$. This is possible by enforcing *partial path consistency* (PPC), which was proposed by Bliek and Sam-Haroud (1999) and first applied to the STP by Xu and Choueiry (2003). PPC is defined for chordal graphs†, which generally contain far fewer edges than the complete graph. Once

*Note that for modelling the latter, there exists an extension of the STP: the Simple Temporal Problem with Uncertainty (STPU).

†Informally, a graph is chordal if every cycle of length greater than 3 has a "shortcut"; for a formal definition, see Definition 1.

PPC has been enforced, the label of every edge present in the STN is minimal: it is equivalent to the label it would have in $\mathcal{M}$. Note that the original STP instance $\mathcal{S}$, the minimal network $\mathcal{M}$, and the PPC network are equivalent, which means that their solution sets are identical; however, only for the latter two it is guaranteed that any solution can be extracted in a backtrack-free manner.

The partially path-consistent network of our example is depicted in Figure 2. New information has become available, which answers the remainder of the question stated above: we know from constraint $c_{0\to 2}$ that John arrives in the monitoring room between 7:40 and 7:50, and from constraint $c_{0\to 3}$ that Fred left there between 7:20 and 7:30. Furthermore, the constraints $c_{0\to 4}$ and $c_{3\to 2}$ have been tightened, which means that our original information can be refined: Fred's shift won't end before 8:00, and the **casting** process will be left unattended for at least 10 minutes.

In the remainder of this section, we expand our example with a situation where new information becomes available.

Example (continued)**.** *John cannot finish his previous task before 7:20. How does this influence the results we attained thus far?*

Of course, it is possible to incorporate this information into the STP instance and to either decide consistency or enforce decomposability from scratch. However, it is generally far more efficient to build upon the consistency or decomposability result that has been achieved earlier and recalculate only those constraints which have to be changed. The importance of efficient incremental methods becomes clear if one thinks of problem instances where initially very little information is known, and frequent updates with new information have to be incorporated into a growing temporal plan.

Currently, several methods exist for incrementally maintaining consistency or decomposability, a selection of which we presently review.

Known Approaches

In this section, we first briefly mention some available single-shot methods for solving the STP; then, we move on to a more detailed description of known incremental approaches. The list of methods we include here is not complete; especially in the case of incremental algorithms, many other approaches exist, some of which are designed for special cases, e.g. where all edge weights are positive. We give some more references in a later section; here, our focus is mostly on some incremental algorithms that have been used in solvers of the Disjunctive Temporal Problem (DTP; Stergiou and Koubarakis 2000).

Single-Shot Algorithms

Single-shot methods take an STP instance and solve it by deciding consistency and/or enforcing decomposability, as described above.

Floyd-Warshall It was already noted by Dechter, Meiri, and Pearl (1991) that the Floyd-Warshall 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. This corresponds to enforcing the property of path consistency (PC), known from constraint satisfaction literature (Montanari 1974).

Johnson Like Floyd-Warshall, Johnson's algorithm also solves the APSP problem; however, it has better worst-case time complexity: $\mathcal{O}\left(nm + n^2 \log n\right)$. Here, m is the number of constraint edges in the STP instance.

Directed path consistency (DPC) This method for determining consistency of an STP instance was also described by Dechter, Meiri, and Pearl (1991); it is a tailored version of the directed path consistency algorithm for general constraint satisfaction problems (Dechter and Pearl 1987). Its time complexity is $\mathcal{O}\left(n(w^*)^2\right)$, where w^* is a parameter known as the *induced width* of the constraint graph. If δ is the graph degree (i.e. the maximum number of neighbours that any single vertex has), it holds that $w^* \leq \delta \leq n$. Note that unlike the others, this algorithm does not enforce decomposability; however, it is relevant as a subroutine in our new algorithms.

$\triangle$STP In 2003, Xu and Choueiry published the $\triangle$STP algorithm. It is based on a publication by Bliek and Sam-Haroud (1999), which introduced a new type of path consistency, called *partial path consistency* (PPC). Instead of producing complete constraint graphs, as does standard path consistency, PPC requires only a chordal graph, which may be much sparser. Thus, by avoiding computation of information which is not interesting anyway, computational effort is decreased, while maintaining decomposability. The worst-case time complexity of $\triangle$STP is not known; empirical evaluation by the authors showed that it nearly always outperformed other solution methods.

P^{3}C Recently, Planken, De Weerdt, and Van der Krogt (2008) identified a class of STP instances for which $\triangle$STP performs quite badly (requiring time $\Omega\left(n^4\right)$), and published a new method for enforcing PPC which does not suffer from this flaw: the P^{3}C algorithm, included in Figure 3 (see also Definition 1 below). Having established chordality, it performs just a forward and backward sweep along the vertices; the forward sweep is identical to enforcing DPC, as is the algorithm's time complexity: $\mathcal{O}\left(n(w^*)^2\right)$.

Incremental algorithms

The STP appears as a pivotal sub-problem of more complex temporal problems, such as the Disjunctive Temporal Problem, introduced by Stergiou and Koubarakis (2000). The usual approach for solving these problems is to gradually build up an STP instance (called a *component STP*) and backtrack whenever an inconsistency is encountered. Beside the consistency check, maintaining minimal relations is often also very relevant, as this information can be used in heuristics that guide the search process.

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

call DPC($\mathcal{S}, d$)
return INCONSISTENT if DPC did
for $k \leftarrow 1$ **to** n **do**
  forall $i, j < k$ **such that** $\{i, k\}, \{j, k\} \in E$ **do**
    $w_{i \to k} \leftarrow \min(w_{i \to k}, w_{i \to j} + w_{j \to k})$
    $w_{k \to j} \leftarrow \min(w_{k \to j}, w_{k \to i} + w_{i \to j})$
  end
end
return $\mathcal{S}$

Figure 3: The $\mathrm{P}^3\mathrm{C}$ algorithm

More concretely, at each step in the backtracking search, a single constraint is to be added to a component STP that is already known to be consistent (or decomposable). In this situation, single-shot STP algorithms are not the most efficient option; instead, one wants to build upon the consistency (or decomposability) result that has been achieved earlier and reconsider only those constraints which may have to be changed. In the past, two types of incremental algorithms have been employed by DTP solvers: incremental (full) path consistency and incremental directed path consistency. We describe these methods below.

Incremental Directed Path Consistency This method for incrementally enforcing directed path consistency (IDPC) was published by Chleq (1995). The algorithm is similar to the single-shot DPC algorithm mentioned above; the chief difference is that track is kept of the constraints that have been modified, to avoid unnecessary constraint checks. However, the worst-case complexity of the incremental algorithm is no better than that of the single-shot version, and remains $\mathcal{O}\left(n(w^*)^2\right)$.

The IDPC algorithm was employed in the original DTP solver by Stergiou and Koubarakis (2000). Tsamardinos and Pollack (2003) mention IDPC and incorrectly state that its worst-case time complexity is $\mathcal{O}\left(n^2\right)$.

Incremental Full Path Consistency (IFPC) Beside maintaining directed path consistency, it is also possible to incrementally maintain full path consistency (IFPC), i.e. all-pairs shortest paths. Tsamardinos and Pollack (2003) use this approach, citing a paper by Mohr and Henderson (1986); however, instead of an incremental approach, this paper only presented a new single-shot path consistency algorithm for general constraint satisfaction problems. Nevertheless, it is not hard to come up with a straightforward algorithm that does the job within the $\mathcal{O}\left(n^2\right)$ time bound stated by Tsamardinos and Pollack. We present such an algorithm in Figure 4. The sets I and J are maintained to improve efficiency by avoiding unnecessary constraint checks: they are not relevant for the algorithm's soundness.

As noted by Even and Gazit (1985), this is also the best attainable general upper bound on the time complexity for an

Input: A minimal STN $\mathcal{S} = \langle V, E \rangle$ and a new constraint $c'_{a \to b}$.
Output: CONSISTENT if $c'_{a \to b}$ has been added to $\mathcal{S}$, which is again minimal; INCONSISTENT otherwise.

if $w'_{a \to b} + w_{b \to a} < 0$ **then return** INCONSISTENT
if $w'_{a \to b} \geq w_{a \to b}$ **then return** CONSISTENT
$w_{a \to b} \leftarrow w'_{a \to b}$
$I \leftarrow \emptyset$; $J \leftarrow \emptyset$
forall $v_k \in V$, $v_k \neq v_a, v_b$ **do**
  if $w_{k \to b} > w_{k \to a} + w_{a \to b}$ **then**
    $w_{k \to b} \leftarrow w_{k \to a} + w_{a \to b}$
    $I \leftarrow I \cup \{k\}$
  end
  if $w_{a \to k} > w_{a \to b} + w_{b \to k}$ **then**
    $w_{a \to k} \leftarrow w_{a \to b} + w_{b \to k}$
    $J \leftarrow J \cup \{k\}$
  end
end
forall $i \in I$, $j \in J$, $i \neq j$ **do**
  if $w_{i \to j} > w_{i \to a} + w_{a \to j}$ **then**
    $w_{i \to j} \leftarrow w_{i \to a} + w_{a \to j}$
  end
end
return CONSISTENT

Figure 4: An incremental full path consistency algorithm (IFPC)

incremental APSP algorithm. They then proceed to describe a more involved algorithm whose complexity is $\mathcal{O}(\delta m^*)$, where m^* is the number of constraint edges that were already present whose weight is to be changed by the addition of the new constraint. Their algorithm (which we dub IAPSP) thus improves over the straightforward implementation especially if the original graph degree δ is low or few changes have to be made. Due to space constraints, we cannot include the pseudocode of this algorithm.

Graph Triangulation

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

Definition 1. *Let $G = \langle V, E \rangle$ be an undirected graph. We can define the following concepts:*

- *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.*[*]

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

- *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 1 (e.g. West, 1996)**.** *An undirected graph $G = \langle V, E \rangle$ is chordal if and only if it has a simplicial elimination ordering.*

Chordality checking can be done efficiently in $\mathcal{O}(n + m)$ time by the maximum cardinality search algorithm, which also 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 2 (Kjærulff, 1990)**.** *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 $|T'| < |T|$.*

Determining a minimum triangulation is an NP-hard 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.

Incremental Partial Path Consistency

As the main contribution of this paper, we now present in Figure 5 a new algorithm, called IPPC, that incrementally enforces partial path consistency to complement the incremental methods presented above. The algorithm as presented here assumes that a set $E' \subseteq V \times V$ of all constraint edges that may be added is known on beforehand; therefore, the graph representing all time points and all these constraint edges can be triangulated once. Alternatively, chordality itself could be enforced incrementally before running the incremental PPC algorithm; note, however, that this would also require computing the weights on the new edges added during incremental triangulation. This matter is subject to further research and as such outside the scope of this text.

Input: A partially path-consistent STN instance $\mathcal{S} = \langle V, E \rangle$ and a new constraint $c'_{a \to b}$ with $\{a, b\} \in E$.
Output: CONSISTENT if $c'_{a \to b}$ has been added to $\mathcal{S}$, which is again partially path-consistent; INCONSISTENT otherwise.

```
1  if w'_{a→b} + w_{b→a} < 0 then return INCONSISTENT
2  if w'_{a→b} ≥ w_{a→b} then return CONSISTENT
3  w_{a→b} ← w'_{a→b}
4  Fix a simplicial elimination ordering
   d = (x_n, x_{n−1}, ..., x_1) of S such that
   x_a = x_1 ∧ x_b = x_2
5  D ← {(1, 2)}
6  for k ← 1 to n do
7      forall i, j < k such that
       {i, k}, {j, k} ∈ E ∧ (i, j) ∈ D do
8          if w_{i→k} > w_{i→j} + w_{j→k} then
9              w_{i→k} ← w_{i→j} + w_{j→k}
10             D ← D ∪ {(i, k)}
11         end
12         if w_{k→j} > w_{k→i} + w_{i→j} then
13             w_{k→j} ← w_{k→i} + w_{i→j}
14             D ← D ∪ {(k, j)}
15         end
16     end
17 end
18 return CONSISTENT
```

Figure 5: Incremental partial path consistency algorithm (IPPC)

Like IFPC, this algorithm has the desirable property that any inconsistency caused by tightening a constraint is detected in constant time; this follows from the fact that every constraint in a partially path-consistent STP instance is minimal.

With respect to IPPC's soundness, the fundamental insight is that after tightening the new constraint $c'_{a \to b}$, the network is directionally path-consistent with respect to the simplicial elimination ordering d fixed in line 4. This follows from the observations that

- if a network is partially path-consistent, it is directionally path-consistent with respect to any simplicial elimination ordering; and
- directional path consistency with respect to some variable ordering cannot be lost by tightening the constraint that appears last in this ordering.

Recall also from the preceding section that fixing a simplicial elimination ordering requires only linear time using maximum cardinality search.

The main loop of IPPC is in effect similar to the second main loop from the $\mathrm{P}^3\mathrm{C}$ algorithm (Figure 3); having reestablished DPC, it enforces PPC in the reverse order of a

simplicial elimination ordering. The chief difference is that the algorithm maintains the set D of constraints that have been updated; in this way, only the necessary checks are performed. By now, this can be recognised as a recurring theme in all incremental algorithms presented.

The worst-case performance of this algorithm is no better than the "single-shot" PPC algorithm and remains $\mathcal{O}\left(n \cdot (w^*)^2\right)$. This suggests that IFPC remains the incremental method of choice for dense graphs, though the actual performance of IPPC may be better in practical cases. In the next section, we empirically evaluate the actual performance of IPPC against IAPSP for graphs of different densities.

Experimental Results

In this section, we discuss the results of putting our new IPPC algorithm to the test against its competitors described earlier. We mentioned that IPPC requires the constraint graph to be chordal, and that this could be ensured in several ways. In our experiments, we considered two methods: (i) naively triangulate (using the minimum-degree heuristic) after the addition of each new constraint edge; and (ii) construct the graph consisting of all constraint edges that will be added and triangulate it once with the minimum-fill heuristic. As was to be expected, option (i) is always slower than option (ii); however, the latter is only feasible if the structure of the constraint graph is known beforehand. In the results discussed here, we only include the latter option.

Our tests included three other algorithms: the straightforward IFPC implementation from Figure 4, the IDPC algorithm by Chleq (1995), and the improved IAPSP algorithm by Even and Gazit (1985). The algorithms were tested on two types of problem instances:

- STP instances with enforced consistency, generated from job shop problems. The structure of the constraint graphs of these instances is expected to be similar to the types of problems that STP solvers would be expected to deal with if they were used in a DTP solver on the job shop problem. The labels on the edges have been modified to ensure that the problem instances are consistent and all constraints can be incrementally added.
- Random consistent STP instances on scale-free constraint graphs, generated according to the BA model (Albert and Barabási 2002) with varying density parameter. In scale-free constraint graphs, the distribution of the vertex degree follows a power law; these graphs can be used to accurately model many real-life examples (e.g. social networks) in which few nodes have a high degree, and many nodes have a low degree.

The test results are included in Figures 6 through 8. All plots list, on the vertical axis, the time in seconds required for adding all constraints in the various STP instances; the horizontal axis depicts the number of vertices in Figures 6 and 7, whereas it depicts the density parameter in Figure 8.

The immediate first observation is that unfortunately, our algorithm is woefully outperformed in all cases (and sometimes even exceeds the time limit of 12 minutes). Further, in all of our experiments, the straightforward IFPC algorithm surprisingly is faster than the more advanced IAPSP,

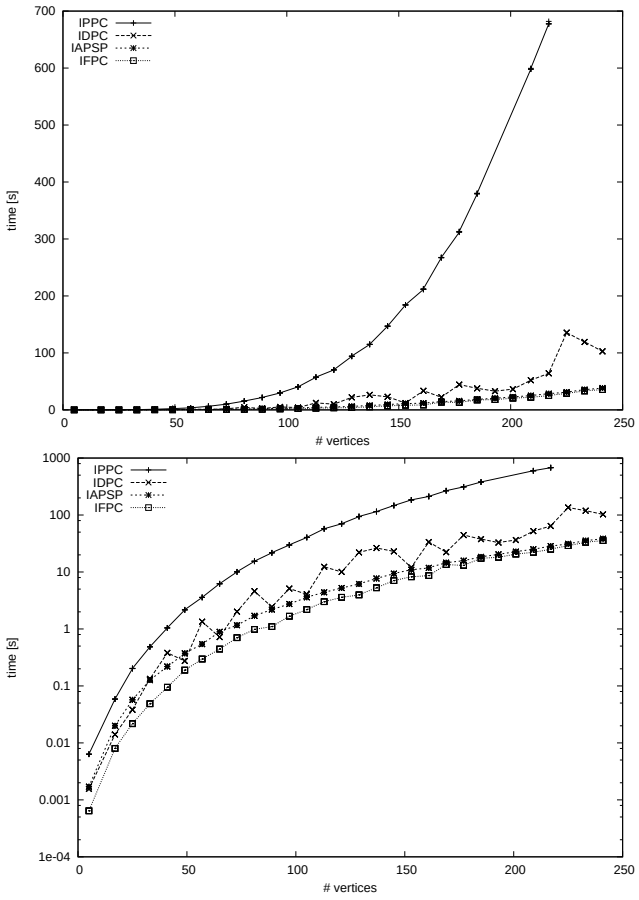

Figure 6: Incremental methods on the job shop problem linear scale (top) and log scale (bottom)

though the right-hand side of the plot in Figure 6 hints that for larger instances, IAPSP may gain the upper hand. Finally, the IDPC algorithm lies somewhere in between IAPSP and IPPC. Since this algorithm does not enforce decomposability, it is probably safe to state that an IFPC algorithm is always a better choice than IDPC.

We have not yet found a satisfying explanation why IPPC performs so poorly, whereas its sibling $\mathrm{P}^3\mathrm{C}$ is the new state-of-the-art algorithm for enforcing decomposability. It would seem that the additional information that is present in a complete graph representation as maintained by IFPC allows additions of constraints to be processed more efficiently, at the cost of a higher space complexity. This is subject to future research.

Related Work

Of course, the list of algorithms we gave earlier is not complete; this section lists some other publications relevant to incrementally solving the STP.

The Bellman-Ford algorithm can be used as a single-shot approach if one is interested in only determining consistency, and an incremental variant tailored to the STP has been published by Cesta and Oddi (1996). More work on

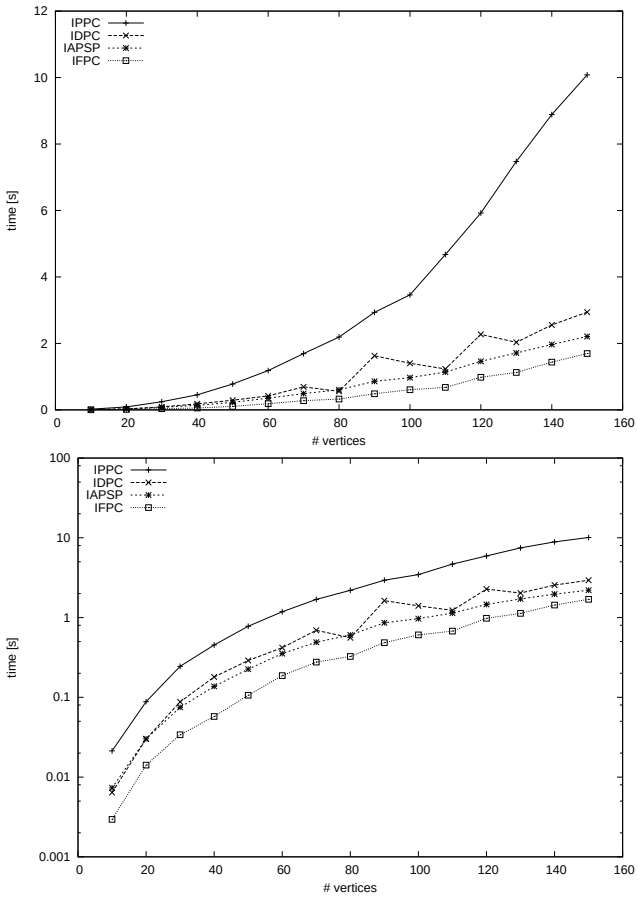

Figure 7: Incremental methods on sparse scale-free graphs
linear scale (top) and log scale (bottom)

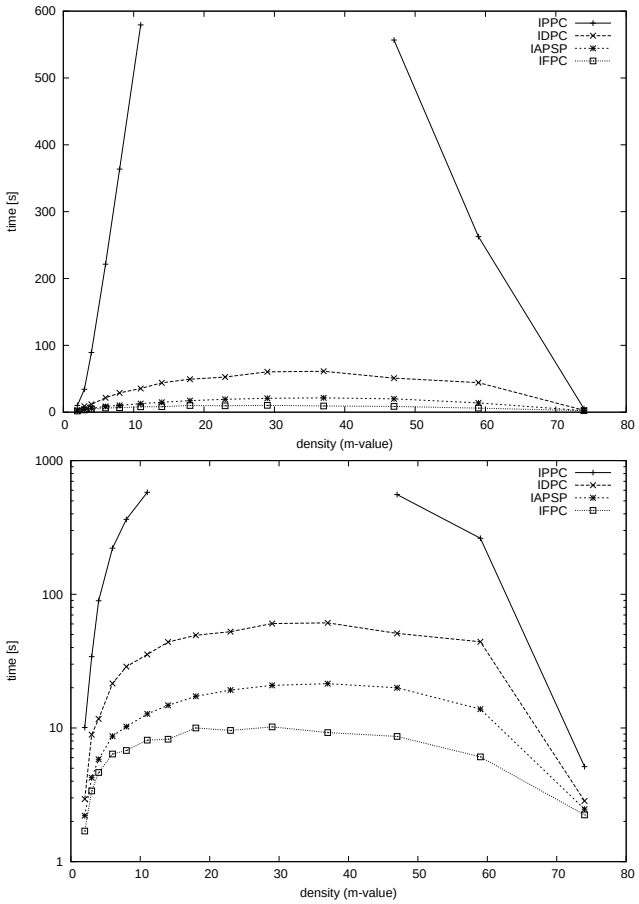

Figure 8: Incremental methods on scale-free graphs
with $n = 150$ and varying density
linear scale (top) and log scale (bottom)

incremental consistency checking for the STP has been done by Ramalingam et al. (1999), who also list some algorithms for special cases of the STP.

Demetrescu and Italiano (2002) give a mostly graph-theoretical overview of available algorithms for the dynamic all-pairs shortest paths problem. They also consider the problem with only positive edge-weights, and moreover list algorithms that can not only handle decreasing edge weights and edge additions, but also increases in edge weights and edge deletions; hence the name "dynamic APSP" instead of "incremental APSP".

We earlier briefly mentioned the possibility of representing uncertainty induced by the environment. The Simple Temporal Problem with Uncertainty (STPU) is an extension of the STP that can be used to model these situations. For real-time execution of STPU instances, the property of *dispatchability* is desirable; it means that the problem can be executed without fear of conflicts. Shah et al. (2007) present an algorithm that incrementally maintains dispatchability. An example of a situation where their approach is useful is when an exogenous event occurs, and a previous interval of uncertainty can be reduced to a single value representing the exact time of occurrence.

Conclusions and Future Research

Based on our earlier success with the $\mathrm{P^3C}$ algorithm, we made a natural adaptation of this algorithm to incrementally enforce partial path consistency on the STP. Unfortunately and surprisingly, the IPPC algorithm is consistently much slower than all its competitors. The foremost subject for future research is to investigate what causes this poor performance.

Further, since $\mathrm{P^3C}$ works so well as a single-shot STP algorithm, in which all constraints are considered at once, whereas processing a single constraint at a time in IPPC proved to be slow, a natural question is to ask whether there is a trade-off here. Might IPPC perform better, when compared to its competitors, in a situation where new constraints become available in batches, instead of one by one?

The theoretical worst-case time complexity of the improved IAPSP algorithm by Even and Gazit (1985) can be expressed in the number of constraints to be changed. If we could find a similar bound for IPPC, this could also give us an idea of areas where it performs better than our current experiments showed.

We also noted earlier that we have not yet addressed the

issue of incremental triangulation in our current work. Constraints that are added may form new (unchorded) cycles in the constraint graph, which have to be dealt with. There would appear to be another trade-off between dealing with these cycles in a fast and naive way, or in a clever but slower way. However, this issue is moot unless IPPC can be shown to be a viable alternative to other approaches.

Finally, the incremental algorithms we presented can only deal with lowering edge weights and adding new edges; if an edge weight is increased or an edge is deleted, solving has to start anew. Above, we mentioned the existence of algorithms that deal with the fully dynamic situation where both additions and deletions are possible; we hope to investigate in a future publication whether a PPC-based algorithm for this setting is feasible.

References

Albert, R., and Barabási, A.-L. 2002. Statistical mechanics of complex networks. *Reviews of Modern Physics* 74(1):47–97.

Bliek, C., and Sam-Haroud, D. 1999. Path consistency on triangulated constraint graphs. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 456–461. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Cesta, A., and Oddi, A. 1996. Gaining efficiency and flexibility in the simple temporal problem. In *TIME '96: Proceedings of the 3rd Workshop on Temporal Representation and Reasoning (TIME'96)*, 45–50. Washington, DC, USA: IEEE Computer Society.

Chleq, N. 1995. Efficient algorithms for networks of quantitative temporal constraints. In *Proceedings of CONSTRAINTS-95, First International Workshop on Constraint Based Reasoning*, 40–45.

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

Dechter, R., and Pearl, J. 1987. Network-based heuristics for constraint-satisfaction problems. *Artif. Intell.* 34(1):1–38.

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

Demetrescu, C., and Italiano, G. F. 2002. Improved bounds and new trade-offs for dynamic all pairs shortest paths. In *ICALP*, volume 2380, 633–643. Springer.

Even, S., and Gazit, H. 1985. Updating distances in dynamic graphs. *Methods of Oper. Res.* 49:371–387.

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

Mohr, R., and Henderson, T. C. 1986. Arc and path consistency revisited. *Artificial Intelligence* 28(2):225–233.

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

Planken, L.; De Weerdt, M.; and Van der Krogt, R. 2008. P3C: A new algorithm for the Simple Temporal Problem. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, 256–263. Menlo Park, CA, USA: AAAI Press.

Ramalingam, G.; Song, J.; Joskowicz, L.; and Miller, R. E. 1999. Solving systems of difference constraints incrementally. *Algorithmica* 23(3):261–275.

Shah, J.; Stedl, J.; Williams, B.; and Robertson, P. 2007. A fast incremental algorithm for maintaining dispatchability of partially controllable plans. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, 296–303. AAAI Press.

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

Tsamardinos, I., and Pollack, M. E. 2003. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence* 151(1–2):43–89.

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 *TIME-ICTL 2003: Proceedings of the 10th International Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic*, 210–220. Los Alamitos, CA, USA: IEEE Computer Society.