Exploiting the Structure of Hierarchical Plans in Temporal Constraint Propagation

Neil Yorke-Smith

Artificial Intelligence Center, SRI International, Menlo Park, CA 94025, USA

nysmith@ai.sri.com

Abstract

Quantitative temporal constraints are an essential requirement for many planning domains. The HTN planning paradigm has proven to be better suited than other approaches to many applications. To date, however, efficiently integrating temporal reasoning with HTN planning has been little explored. This paper describes a means to exploit the structure of a HTN plan in performing temporal propagation on an associated Simple Temporal Network. By exploiting the natural restriction on permitted temporal constraints, the time complexity of propagation can be sharply reduced, while completeness of the inference is maintained. Empirical results indicate an order of magnitude improvement on real-world plans.

Introduction

Quantitative temporal constraints are an essential requirement for many real-life planning domains (Smith, Frank, & Jónsson 2000). The Hierarchical Task Network (HTN) planning paradigm has proven to be well-suited to many applications (Myers *et al.* 2002). To date, however, efficiently integrating temporal reasoning within the HTN planning process has been explored in only a few systems.

This paper describes a means to exploit the structure of a HTN plan in performing temporal propagation on an associated Simple Temporal Network (STN). We introduce an algorithm called *sibling-restricted propagation* that exploits the restricted structure of STNs that arise from an HTN plan. The idea behind the algorithm is to transverse a tree of sub-STNs that correspond to the expansions in the HTN task hierarchy. The HTN structure limits the sub-STNs to have constraints only between parent and child nodes and between sibling nodes. Because the STNs thus considered are small, compared to the *global* STN corresponding to the whole plan, the overall amount of work to perform propagation is much less. Empirical results demonstrate an order of magnitude improvement on real-world plans.

Many metric temporal planners adopt an STN to describe the temporal relations underlying the plan. HTN planners in this category include O-Plan (Tate, Drabble, & Kirby 1994), SIPE-2 (Wilkins 1999), HSTS/RA/Europa (Jónsson *et al.* 2000), PASSAT (Myers *et al.* 2002), and SHOP2 (Nau *et al.* 2003). Similar representations are used by other HTN systems, such as IxTeT (Laborie & Ghallab 1995). Our work is distinguished by explicit use of the HTN plan structure to propagate on the underlying STN.

Combining planning and scheduling has been approached from both sides of the gap (Smith, Frank, & Jónsson 2000). Specific algorithms have been developed for temporal propagation (e.g. (Tsamardinos, Muscettola, & Morris 1998)) and resource propagation (e.g. (Laborie 2003)) within a planning context. Again, while numerous systems feature methods to efficiently propagate temporal information and use it in the planning or scheduling process, we are not aware of any published results on specific algorithms to exploit HTN structure in STN propagation.

The next section presents necessary background on HTN planning, Simple Temporal Networks, and STN propagation algorithms. The following sections introduce the sibling-restricted propagation algorithm, present an initial characterisation of its theoretical properties, and evaluate its implementation in the PASSAT plan authoring system.

Background

Hierarchical Task Network planning (Erol, Hendler, & Nau 1994) generalises traditional operator-based planning through the addition of methods. Methods encode rich networks of tasks that can be performed to achieve an objective. Tasks within a method are temporally partially ordered, and may have associated preconditions and effects in addition to those of the method as a whole. With HTN methods, planning can assume a hierarchical flow, with high-level tasks being decomposed progressively into collections of lower-level tasks through the application of matching methods with satisfied preconditions. Many large-scale, realistic planning applications have employed the HTN paradigm (Smith, Frank, & Jónsson 2000).

Simple Temporal Networks For modelling and solving the temporal aspects of planning and scheduling problems, quantitative temporal constraint networks in the form of the *Simple Temporal Problem* (Dechter, Meiri, & Pearl 1991) are widely adopted. An STN is a restriction of the *Temporal Constraint Problem* to have a single interval per constraint. Variables X_k denote time-points and constraints represent binary quantitative temporal relations between them. A distinguished time-point, denoted *TR*, marks the start of time. Unary domain constraints are modelled as binary relations to

TR; thus all constraints have the form: $l_{ij} \leq X_j - X_i \leq u_{ij}$, where l_{ij} and u_{ij} are the lower and upper bounds respectively on the temporal distance between time-points X_i and X_j, i.e. $X_j - X_i \in [l_{ij}, u_{ij}]$.

Consistency of an STN can be determined by enforcing path consistency (PC) on the distance graph arising from the constraints (Dechter, Meiri, & Pearl 1991). Moreover, an STN, together with the *minimal network* of time-point domains, can be specified by a complete directed graph, its *d-graph*, where edge $i \rightarrow j$ is labelled by the shortest path length d_{ij} between X_i and X_j in the distance graph. Any All-Pairs Shortest Path algorithm (e.g. Floyd-Warshall) may be used to compute the d-graph given the distance graph, and the d-graph may be represented as a sparse or dense *distance matrix*. We denote its computation by PC.

Like many other planners, PASSAT employs an STN to represent the temporal aspects of plans, using an approach called constraint-based interval planning (Frank & Jónsson 2004). The temporal extent of each task is modelled by a time-point each for its start and end. At regular occasions in the planning process, checking consistency of the temporal constraints and propagation of temporal information is required. This is achieved by invoking PC on the plan's STN.

Propagation The basic method for PC is to use an All-Pairs Shortest Path algorithm on the distance matrix A. The STN is consistent iff no diagonal element is negative: $a_{ii} < 0$ for some i corresponds to a cycle in the d-graph (Dechter, Meiri, & Pearl 1991). In the terminology of Bessière (1996), this method is PC-1. Let the d-graph have V vertices and E edges. The complexity of PC-1 is $\Theta(V^3)$ (Floyd-Warshall) for a dense representation of the graph and $\Theta(V^2 \log V + VE)$ (Johnson's algorithm) for a sparse representation. Note that a HTN with n tasks has $2n + 1$ vertices in the d-graph of its STN: two time-points for each task, plus one for *TR*.

Dechter (2003), Bessière (1996) present other path consistency algorithms that can be specialised for the STN and used for PC. Of note is PC-2, which avoids redundant computation by use of an auxiliary data structure. For an STN with n time-points, if PC-1 is $\Theta(n^3)$ time and $\Theta(n^2)$ space, PC-2 is $\Theta(n^3)$ time but $\Theta(n^3)$ space, but exhibits better performance in practice provided the space requirements do not dominate (Bessière 1996). Dechter (2003) also presents an algorithm DPC that determines consistency but does not obtain the minimal network; separately, Cesta & Oddi (1996) present an incremental algorithm with the same function.

These algorithms are largely subsumed by $\triangle$STP (Xu & Choueiry 2003). This algorithm, which does find the minimal network, outperforms PC-1, and is comparable to (dense graphs) or outperforms DPC (sparse graphs). The algorithm proposed in this paper invokes an STN solver repeatedly on different STNs; $\triangle$STP or any of the other methods described for PC may be employed.

Sibling-Restricted Propagation

The idea behind sibling-restricted propagation is to exploit the HTN structure, under a mild restriction on permitted temporal constraints. Simple temporal constraints are permitted only between parent tasks and their children, and be-

tween sibling tasks. For example, suppose task **A** has been decomposed into tasks **B** and **E**. Temporal constraints are permitted between the start and end time-points of **A**, **B** and **E**. They are permitted between **B** and its children, but not between **A** or **E** and the children of **B**. Temporal constraints are also prohibited between **B** and any other task **X**.

This assumption on what STN constraints may exist between plan elements is inherent to HTN models. In particular, there is no way in standard HTN representations to specify temporal constraints between tasks in different task networks (Erol, Hendler, & Nau 1994). Thus sibling-restricted propagation imposes no additional limitations on the expressiveness of HTNs.

The STN that arises from an HTN with the sibling constraint restriction has marked structure properties. The STN can be decomposed into a tree of smaller STNs; the shape of this tree mirrors the shape of the hierarchical structure in the plan. By traversing this tree, invoking PC at each 'node' STN, we can propagate temporal information on the plan elements. The restriction on constraints guarantees we can propagate on this tree and lose no information compared to propagating with the whole global STN: it means that the algorithm SR-PC presented below is sound and complete.

Expanding a task τ into its children imposes some implied HTN constraints: each child τ_i cannot start before or finish after its parent; in terms of Allen's algebra, τ_i during τ. STN constraints can represent all of Allen's base relations. They can also represent the partial ordering of children in a task network, which we denote $\tau_i \leq \tau_j$ (of course, children need not be ordered). What cannot be expressed are disjunctive constraints such as "τ_i occurs before or after τ_j".

Algorithm Description

To explain the SR-PC algorithm we need some details on the distance matrix representation $A = (a_{ij})$ of a set of tasks. The *domain* of a time-point — its current known earliest possible start and latest possible finish times — is given by its current temporal distance from *TR*. In the minimal network form of the STN, the domain of every time-point is the broadest possible, given the constraints, such that every value in the domain participates in at least one feasible solution to the STN (Dechter, Meiri, & Pearl 1991). Without loss of generality, we order our distance matrices with $TR = X_0$ as the first time-point, i.e. the first row and column. Then the domain of a time-point X_i is $[-a_{i0}, a_{0i}]$. The initial domain of X_i is given by any constraints between it and *TR*; if there are none, its default initial domain is $(-\infty, \infty)$.

Secondly, the duration of a task τ is given the bounds on the distance between its start and end time-points (let them be X_i and X_j), i.e. the interval given by the minimum and maximum possible temporal distances between them.[1] If there is an explicit constraint between X_i and X_j, we call the temporal distance it describes the *local domain* of τ. For example, the constraint $10 \leq X_j - X_i \leq 20$, implies that τ's local domain is $[10, 20]$. The local domain is a bound on τ's

---

[1]This, the standard semantics for task durations (Frank & Jónsson 2004; Wilkins 1999), means that, given two of the task's start, duration and end, we can compute bounds for the third.

Algorithm 1 Sibling-Restricted Propagation
```
 1: SR-PC (TR, root task τ)
 2: if τ is not a leaf node then
 3:    Create distance matrix A for τ
 4:    Perform PC on A, and update domains
 5:    L ← children of τ          {list of pending child nodes}
 6:    repeat
 7:       for each child c ∈ L do
 8:          SR-PC (TR, c)                      {recurse}
 9:          Update local domain of c in A
10:       if any change to A occurred then
11:          Perform (incremental) PC on A and update domains
12:          L ← ∅
13:          for each child c of τ do
14:             if c's global or local domain changed by line 11 then
15:                Add c to L                {must reconsider c}
16:    until L = ∅
17: return
```


Figure 1: Example HTN plan with two levels of expansion

duration, possibly not tight if the STN is not minimal. Again without loss of generality, we order our distance matrices to pair the start and end time-points of each task, so that the kth task is modelled by time-points X_{2k-1} and X_{2k}. Then the local domain of task τ_k is $[-a_{2k,2k-1}, a_{2k-1,2k}]$.

Bounds on the duration of τ can be computed also from the domains of X_i and X_j, provided we can relate these time-points to *TR*, i.e. their domains are more informative than $(-\infty, \infty)$; we call this the *global domain* of τ. If the plan has no temporal constraints that relate τ to *TR*, and the user has not specified when τ starts or ends (in absolute time or relative to *TR*), then the global domain on τ will be computed as $(-\infty, \infty)$. However, if X_i and X_j are related to *TR*, then after PC is completed the duration of τ computed from their domains will coincide with the local domain of τ. In general, the duration of τ is contained in the intersection of the two sets of bounds, local and global.

Pseudocode for SR-PC is shown in Algorithm 1. Given a task in a HTN plan, which we call the *root* task τ for the invocation, the algorithm updates the durations of the task and all its descendents in the plan, by recursively following the HTN expansions. Note the root task need not be the top-level objective of the plan, i.e. the root of the whole HTN.

Provided τ is not a leaf in the plan hierarchy, i.e. is not a primitive action or an unexpanded task, we create a distance matrix A (line 3). The time-points in the matrix are those for the temporal reference point *TR*, and for the start and end time-points of the task and its children. On this distance matrix, which corresponds to a subproblem P_τ of the global STN of the whole plan, we perform PC (line 4) and update the domains of the time-points in P_τ.

We then build a list L of *pending* children, whose sub-STN may need to be updated (line 5), and recurse to each child in this list (line 8). Note the list of pending children is initially set to all children of the root task. In making the recursive step, the local domain of the child in its distance matrix is the intersection of its local domain in A and its global domain. This ensures that all inference on the child's duration to date is propagated.

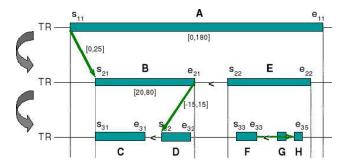Once the recursive steps are all complete, if the local or global domain of any task in P_τ were updated as a result, we update the distance matrix and perform PC again (line 11); this and subsequent invocations of PC may be incremental. Any child whose local domain changes as a result may have an impact on its siblings. Thus we must recurse again to all such children: these children are added to the new list L (line 15) and the loop repeats. The parent–children cycle terminates when parent and all child PC invocations are quiescent (line 16). If at any point a PC invocation finds an inconsistency, SR-PC halts and reports that the whole plan is temporally infeasible.

Worked Example

Figure 1 depicts a small HTN plan. The top-level objective task **A** has been decomposed (during the first expansion: first level to the second) into tasks **B** and **E**. **B** has been decomposed into **C** and **D**; and **E** into **F**–**H** (with two further expansions: second level to the third). The temporal bounds on each task are shown on the three timelines. Each task has start and end time-points. The constraints are the implied HTN constraints (not shown), some pairwise task ordering relations (depicted by $<$), and some quantitative STN constraints (depicted by the arrows). For example, the arrow between the end of task **B** and the start of **D** corresponds to the constraint $-15 \leq X_{s_{32}} - X_{e_{21}} \leq 15$.

The distance matrix of the global STN is shown in Figure 2; '–' denotes no explicit constraint between the two time-points, i.e. an uninformative $X_j - X_i \leq \infty$. Constraints prohibited by the sibling-restricted condition are shown by 'x'; note the marked block structure of these entries. Recall that time-point domains are found in the first row and column; local domains of tasks are found in the off-diagonal entries. For example, **B**'s domain is $[-a_{41}, a_{14}] = (-\infty, \infty)$ while its local domain is $[-a_{54}, a_{45}] = [20, 80]$. Although the matrix as shown is sparse, it will become dense after temporal propagation is complete.

On the STN this matrix represents, SR-PC considers subproblems with *TR* and the time-points of tasks as follows:

```
1:    TR, A
2:       TR, A, B, E
3:          TR, B, C, D
4:             C (leaf: just return)
5:             D (leaf: just return)
6:          TR, E, F, G, H
7:             F (leaf: just return)
8:             G (leaf: just return)
```

```
      TR  A     B      C      D      E   F   G   H
TR    0   0  -     -      -      -   -   -   -     - - - - - - - - -
A     0   0 180  25   -    X   X   X   X    - - X X X X X X   X
      -   0  0   -     0    X   X   X   X    - 0 X X X X X X   X
B     -   0  -    0    80  0    -   -   -     - - - - - - - -   -
      -   -  -  -20  0    -    0  15  0     - - - - - - - -   -
C     -   X  X    0   -    0    -   -   X X X X X X X X   X
      -   X  X   -    -  -5  0    -   -   X X X X X X X X   X
D     -   X  X    0  15  -    0    0  10 X X X X X X X X   X
      -   X  X   -    -   -    -  -5  0  X X X X X X X X   X
E     -   0  -    -    0   X   X   X   X    0 - 0 - - - - -   -
      -   -  -  -    -   X   X   X   X    0 0 - 0 - 0 -   0
F     -   X  X   -    -   X   X   X   X    0 - 0 5 - - - -   -
      -   X  X   -    -   X   X   X   X    - - 0 0 - - 15  -
G     -   X  X   -    -   X   X   X   X    0 - - 0 0 - -   -
      -   X  X   -    -   X   X   X   X    - - - - 0 0 -   -
H     -   X  X   -    -   X   X   X   X    0 - - - - - 0  20
      -   X  X   -    -   X   X   X   X    - - - - - - -5  0
```

Figure 2: Complete distance matrix for the example HTN

```
9:               H (leaf: just return)
10:      TR, A, B, E (no change)
11:   TR, A (no change)
```

To illustrate the propagation steps of SR-PC, consider its invocation with **A** as the root, i.e. line 2 (TR, A, B, E) above. After the initial call to PC, we recurse to each child: first to **B** (the next three lines), then to **E** (the following four lines). Since **A**'s distance matrix was updated by changed domains for **B** and **E** both, we call PC again (line 10 above, line 11 in Algorithm 1). After this step, neither child of **A** has had its global or local domain updated; thus there are no tasks in L for the next iteration (line 16 in Algorithm 1), and so SR-PC terminates.

Algorithm Properties

Because children can be added to the pending list (line 15) on every iteration, it is not obvious that the loop in the Algorithm 1 terminates. The proof comes from considering the circumstances when a local domain of a child task can be updated. We now sketch the principle ideas.

Lemma 1. *Let τ be a task with no grandchildren, and A be its distance matrix formed by Algorithm 1 after PC has been initially applied (i.e. on first entry to the loop). Suppose the local domain of τ in A, d_τ, is tightened, and all other local domains held constant. When consistency is restored with PC, a further tightening of d_τ cannot occur.* □

The main result applies this lemma in structural induction over the tree of STNs considered by SR-PC. The base case is trivial, since Algorithm 1 returns immediately, with no changes to any domain, when invoked on a leaf node.

Theorem 2. *Let Π be a HTN plan with P its underlying (global) STN. Let τ_0 be the top-level objective task of Π. Algorithm 1 invoked on τ_0 terminates.*

Proof. Let $\lambda(\Pi)$ be the number of iterations through the loop in lines 6–16. We proceed by induction over the tree of STNs that SR-PC transverses. Consider a task τ with children $\tau_1, \ldots \tau_f$. Let A be the distance matrix created for τ. Observe that if a child has no children itself, then PC and so SR-PC invoked on the child's distance matrix affects no change (since all of the child's distance matrix is contained in A, and PC has been invoked on A in line 4). In particular, this holds when τ has no grandchildren.

On the first iteration through the loop, any of the local domains of τ or one or more τ_i may be tightened by the consistency restoration of A in line 11. As a result, any of the children may potentially be re-added to L. On the second iteration through the loop, SR-PC is invoked on every child in L. Consider $\tau_i \in L$. The only change to τ_i's distance matrix A_i as formed by SR-PC on the recursive call, compared to A_i at the *end* of the previous call (i.e. during iteration 1), is that the local domain d_{τ_i} of τ_i may be narrowed. But by the inductive hypothesis and Lemma 1, then SR-PC affects no change to d_{τ_i}. Since this is true for every child $\tau_i \in L$, there are no changes to update to A when all the recursive calls have completed. Hence no children of τ are re-added to L on the second iteration, and since $L = \emptyset$ in line 16, the algorithm thus terminates with $\lambda(\Pi) \leq 2$. □

Theorem 2 tells us that, in the STN tree, we have $\lambda = 0$ for leaf nodes, 1 for nodes without grandchildren, and at most 2 otherwise. With termination established, we can prove that the inference obtained by SR-PC is exactly the same as PC, i.e. Algorithm 1 is sound and complete.

Theorem 3. *Algorithm 1 invoked on τ_0 reports inconsistency iff P is inconsistent; otherwise it computes identical minimal domains as PC on P.* □

Theoretical Complexity   For an HTN with approximately uniform branching from the root task, we have characterised an initial theoretical measure of expected time and space complexity of SR-PC. We suppose a mean branching factor (i.e. number of children for each expanded task) f, and mean depth (i.e. number of expansions from root to primitive action) d. For example, Figure 1 has $f = 7/3$ and $d = 2$. Our complexity analysis depends on $\lambda(\Pi)$, the number of iterations through the loop in Algorithm 1. Theorem 2 proved $\lambda \leq 2$. The mean value of λ for a plan depends on the entries of the distance matrices, i.e. on the temporal constraints. In practice, we find λ is often closer to 1 than 2. We are working to obtain a more precise characterisation by considering commonly-occurring structures of STN constraints.

Experimental Results

We have implemented SR-PC in the PASSAT system, with encouraging results. We compare SR-PC with *naive PC*, by which we mean invoking PC on the distance matrix of the global STN: in the example, the matrix of Figure 2. For both methods, the STN solver used for PC was full PC-1; we discuss below the role of more advanced STN solvers, such as incremental PC-1 based on Dijkstra's algorithm.

For an existing, real-world Special Operations Forces (SOF) domain, Table 1 compares SR-PC and naive PC on a selection of complete and partial plans.[2] The columns display, respectively, the name of the plan, the number of tasks

---

[2]The results are for a number of scenarios, including the hostage rescue scenario described in Myers *et al.* (2002). The experiments were conducted on a 1.6GHz Pentium M with 512MB of memory, using Allegro Lisp 6.2.

plan	tasks, vars	d	f	SR-PC time	SR-PC space	SR-PC cpu (s)	naive PC time	naive PC space	naive PC cpu (s)	cpu ratio
airfield-1	40, 27	3.15	2.62	2200	169	0.01	531000	6560	0.49	49
airfield-2	108, 27	8.66	3.19	80600	6500	0.4	10200000	47100	5.21	13.02
recon-1	61, 15	3.73	1.96	31900	2980	0.24	1860000	15100	1.08	4.50
hostage-1	48, 16	3.66	3.91	23700	1910	0.07	913000	9400	0.62	8.86
hostage-2	59, 27	3.23	4.58	36900	2690	0.1	1690000	14200	1.092	10.92
hostage-3	169, 82	4.58	3.67	90800	6890	0.331	42500000	122000	21.73	65.65

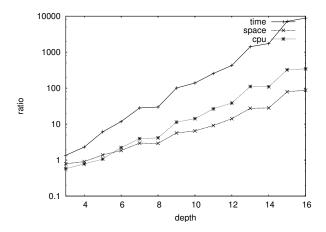Table 1: SR-PC and naive PC on SOF domain plans

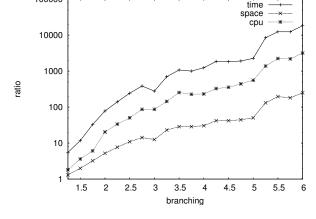Figure 3: Mean time, space, and runtime ratios vs depth

Figure 4: Mean time, space, and runtime ratios vs branching

and non-ground variables, the mean depth d and branching factor f of the HTN; and for each method, measures of the number of operations for time and space,[3] and the actual CPU runtime (in seconds). Note that all three measures are empirical: the time and space are counted operations during the experimental runs. The final column shows the ratio of CPU runtimes; greater than 1 is favourable to SR-PC. Overall, on these real plans, SR-PC outperforms PC by approximately an order of magnitude.

Figures 3 and 4 present a comparison of SR-PC and naive PC on randomly generated plans from an abstract domain. The random generator accepts the parameters: minimum and maximum bounds on the depth d; the mean f and the maximum of a geometric distribution for the number of children of each node; and bounds on the number of temporal constraints in each expansion. The temporal constraints are chosen uniformly from a predefined set. The top-level objective was co-identified with *TR*, i.e. $X_{\tau_0} - TR = 0$.

Figure 3 shows counted operations for time and space, and the empirical runtime, as HTN depth d increases. The ratios between the two methods plotted are for $f = 1.4$; the y axis is a log-scale. Note how the observed runtime ratio (denoted CPU) closely correlates with the time and

space measures. Even for the plans of greatest depth, SR-PC performs propagation within user reaction time. For instance, when $d = 16$, SR-PC requires 0.21s compared to 73s for naive PC. Indeed, the runtime for SR-PC increases approximately linearly with d, while PC exhibits exponential growth. This behaviour is characteristic across other values of f as d varies. Figure 4 shows the effect of varying the mean branching factor f. The ratios plotted are for $d = 5$. In contrast with the depth, as the branching factor increases, the advantage of SR-PC over PC begins to display a possibility of leveling off. Further experiments are needed to observe the trend at greater depths, and for plans with higher values for both d and f at the same time.

Additional experiments varying the temporal consistency of the plan (not reported for space reasons) indicate SR-PC has the greatest advantage when the probability of consistency is lower; for inconsistent problems, SR-PC is able to diagnose the inconsistency earlier. We conjecture this is because the cause of the inconsistency often arises from local interactions within a task network.

Discussion PASSAT is designed to assist the user in a mixed-initiative fashion. In such a user-interactive context, responsiveness of the system is crucial for effective plan authoring, even when developing significant plans with many temporal constraints. Thus, although the difference in absolute runtime for the SOF domains are in the order of seconds, temporal propagation with SR-PC makes the system noticeably and crucially more responsive.

---

[3]These metrics assume, for n time-points, that an implementation of PC has time complexity $\Theta(n^3)$ for full propagation, and $\Theta(n^2)$ when incremental, i.e. for (re)computation for one time-point. This is fitting for a dense representation of distance matrices, as currently in PASSAT.

The theoretical time complexity of naive PC is cubic in the number of time-points. In practice as the plan size grows, our results suggest that the space required comes to dominate; this explains why PC exhibits exponential runtime growth in Figure 3. Our preliminary characterisation of the complexity of SR-PC indicates, in terms of f, a complexity of $\Theta(f^4 f^d)$ versus $\Theta(f^3)$ for PC. As Figure 4 suggests, this implies that the deeper the HTN tree compared to its width, the greater the advantage (or the lesser the disadvantage, at least) of SR-PC if other factors are held constant. However, the influence of other factors is relevant, as shown by Table 1; consistency of the global STN is one of these.

By its operation, SR-PC automatically decomposes the global STN into sub-STNs based on the HTN structure. This is similar to decomposition of STN via its articulation points into biconnected components, which is known to be effective in speeding up propagation (Dechter, Meiri, & Pearl 1991). In our case, however, the global STN is a single biconnected component due to the implied HTN constraints. Thus SR-PC also propagates information between sub-STNs, via the parent task's distance matrix.

The STN solver is a black-box in SR-PC (in fact, within SR-PC, multiple methods for PC can be used on different occasions); the specific STN solver S parameterises SR-PC to the algorithm instance SR-S. Besides PC-1, we implemented the STN solver PC-2 in PASSAT, and compared PC-1, PC-2, SR-PC-1 and SR-PC-2. We found that the space required for PC-2 (building the queue of time-point triples) quickly dominates the runtime, even for modest size plans.

As future work, the sophistication of $\triangle$STP can be leveraged in SR-$\triangle$STP (similar to how it can be leveraged as a black-box in a TCSP solver (Xu & Choueiry 2003)). For naive PC, $\triangle$STP would be expected to outperform PC-1 because the initial distance matrices are relatively sparse — compare Figure 2. On the other hand, $\triangle$STP is expected to bring a smaller benefit to SR-PC because the sub-STNs are smaller and more dense.

Conclusion and Future Work

We have presented an algorithm to efficiently perform temporal propagation on the Simple Temporal Network underlying a temporal hierarchical plan. Sibling-restricted propagation exploits the restricted constraints due to the HTN structure, decomposing the STN into a tree of sub-STNs. The SR-PC algorithm has been implemented in the PASSAT planning system, and empirical results demonstrate the effectiveness of the algorithm. Ongoing work is provide a more precise theoretical characterisation of the complexity.

While the results in PASSAT for SR-PC are favourable over naive PC, we have several improvements to make to the implementation. As noted, the present implementation uses PC-1 as the STN solver. Despite the small average size of the STNs solved by SR-PC, better performance is likely with a stronger solver, such as $\triangle$STP. Second, there may be value in employing a sparse array representation. Third, coincidence of time-points is not actively exploited.

The reasoning problem addressed in this paper is determining the consistency and computing the minimal domains of time-points, for an STN underlying a plan. In both HTN and non-HTN planning, the plan is built incrementally; thus the associated STN is also built incrementally, and inference on it should exploit incremental constraint addition (and removal on backtracking). Incremental versions of classical STNs algorithms are widely used (Cesta & Oddi 1996). An important next step for us is therefore to extend SR-PC to an incremental version of the algorithm. In HTN planning, constraints are added (removed) when a task network is expanded (expansion backtracked). Besides making use of an incremental STN solver, incremental SR-PC thus involves determining the highest task in the HTN tree that has changed, and considering the STN tree rooted at this task rather than at the top-level objective.

References

Bessière, C. 1996. A simple way to improve path consistency in interval algebra networks. In *Proc. of AAAI-96*, 375–380.

Cesta, A., and Oddi, A. 1996. Gaining efficiency and flexibility in the simple temporal problem. In *Proc. of TIME-96*, 45–50.

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: Morgan Kaufmann.

Erol, K.; Hendler, J.; and Nau, D. 1994. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, Computer Science Department, University of Maryland.

Frank, J., and Jónsson, A. 2004. Constraint-based attribute and interval planning. *Constraints* 8(4):339–364.

Jónsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in interplanetary space: Theory and practice. In *Proc. of AIPS'00*, 177–186.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proc. of IJCAI'95*, 1643–1649.

Laborie, P. 2003. Algorithms for propagating resource constraints in ai planning and scheduling: existing approaches and new results. *Artificial Intelligence* 143(2):151–188.

Myers, K. L.; Tyson, W. M.; Wolverton, M. J.; Jarvis, P. A.; Lee, T. J.; and desJardins, M. 2002. PASSAT: A user-centric planning framework. In *Proc. of the Third Intl. NASA Workshop on Planning and Scheduling for Space*.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *J. Artificial Intelligence Research* 20:379–404.

Smith, D.; Frank, J.; and Jónsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Eng. Review* 15(1).

Tate, A.; Drabble, B.; and Kirby, R. 1994. O-Plan2: An architecture for command, planning and control. In Fox, M., and Zweben, M., eds., *Intelligent Scheduling*. Morgan Kaufmann.

Tsamardinos, I.; Muscettola, N.; and Morris, P. H. 1998. Fast transformation of temporal plans for efficient execution. In *Proc. of AAAI-98*, 254–261.

Wilkins, D. E. 1999. *Using the SIPE-2 Planning System*. Artificial Intelligence Center, SRI International, Menlo Park, CA.

Xu, L., and Choueiry, B. Y. 2003. A new efficient algorithm for solving the simple temporal problem. In *TIME'03*, 212–222.