Journal of Artificial Intelligence Research 47 (2013) 95-156 Submitted 10/12; published 5/13

Distributed Reasoning for Multiagent Simple Temporal Problems

James C. Boerkoel Jr. BOERKOEL@CSAIL.MIT.EDU
Computer Science and Artificial Intelligence Laboratory,
Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Edmund H. Durfee DURFEE@UMICH.EDU
Computer Science and Engineering,
University of Michigan, Ann Arbor, MI 48109, USA

Abstract

This research focuses on building foundational algorithms for scheduling agents that assist people in managing their activities in environments where tempo and complex activity interdependencies outstrip people's cognitive capacity. We address the critical challenge of reasoning over individuals' interacting schedules to efficiently answer queries about how to meet scheduling goals while respecting individual privacy and autonomy to the extent possible. We formally define the Multiagent Simple Temporal Problem for naturally capturing and reasoning over the distributed but interconnected scheduling problems of multiple individuals. Our hypothesis is that combining bottom-up and top-down approaches will lead to effective solution techniques. In our bottom-up phase, an agent externalizes constraints that compactly summarize how its local subproblem affects other agents' subproblems, whereas in our top-down phase an agent proactively constructs and internalizes new local constraints that decouple its subproblem from others'. We confirm this hypothesis by devising distributed algorithms that calculate summaries of the joint solution space for multiagent scheduling problems, without centralizing or otherwise redistributing the problems. The distributed algorithms permit concurrent execution to achieve significant speedup over the current art and also increase the level of privacy and independence in individual agent reasoning. These algorithms are most advantageous for problems where interactions between the agents are sparse compared to the complexity of agents' individual problems.

1. Introduction

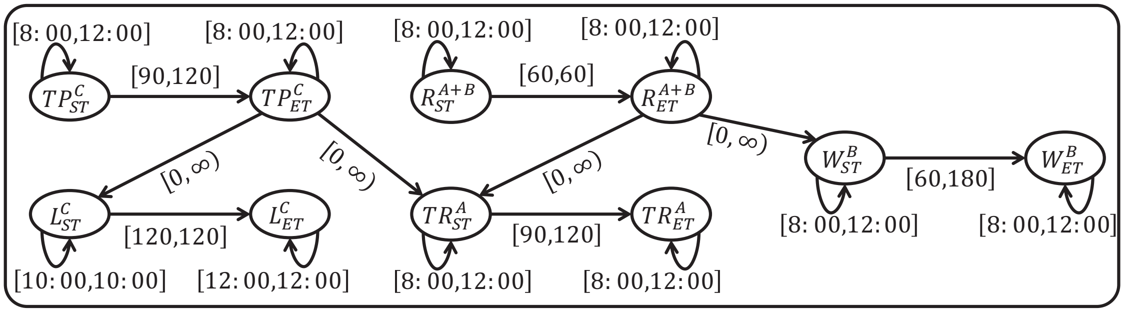
Computational scheduling agents can assist people in managing and coordinating their activities in environments where tempo, a limited view of the overall problem, and complex activity interdependencies can outstrip people's cognitive capacity. Often, the schedules of multiple agents interact, which requires that an agent coordinate its schedule with the schedules of other agents. However, each individual agent, or its user, may have strategic interests (privacy, autonomy, etc.) that prevent simply centralizing or redistributing the problem. In this setting, a scheduling agent is responsible for autonomously managing the scheduling constraints as specified by its user. As a concrete example, consider Ann, who faces the task of processing the implications of the complex constraints of her busy schedule. A further challenge for Ann is that her schedule is interdependent with the schedules of her colleagues, family, and friends with whom she interacts. At the same time, Ann would prefer the independence to make her own scheduling decisions and to keep a healthy separation between her personal, professional, and social lives.

The Simple Temporal Problem (STP) formulation is capable of representing scheduling problems where, if the order between pairs of activities matters, this order has been predetermined. As such, the STP acts as the core scheduling problem representation for many interesting planning problems (Laborie & Ghallab, 1995; Bresina, Jónsson, Morris, & Rajan, 2005; Castillo, Fernández-Olivares, & O. García-Pérez, 2006; Smith, Gallagher, Zimmerman, Barbulescu, & Rubinstein, 2007; Barbulescu, Rubinstein, Smith, & Zimmerman, 2010). One of the major practical advantages of the STP representation is that there are many efficient algorithms that compute flexible spaces of alternative solutions (Dechter, Meiri, & Pearl, 1991; Hunsberger, 2002; Xu & Choueiry, 2003; Planken, de Weerdt, & van der Krogt, 2008; Planken, de Weerdt, & Witteveen, 2010a). This improves robustness to the stochasticity of the real world and the unreliability of human users.

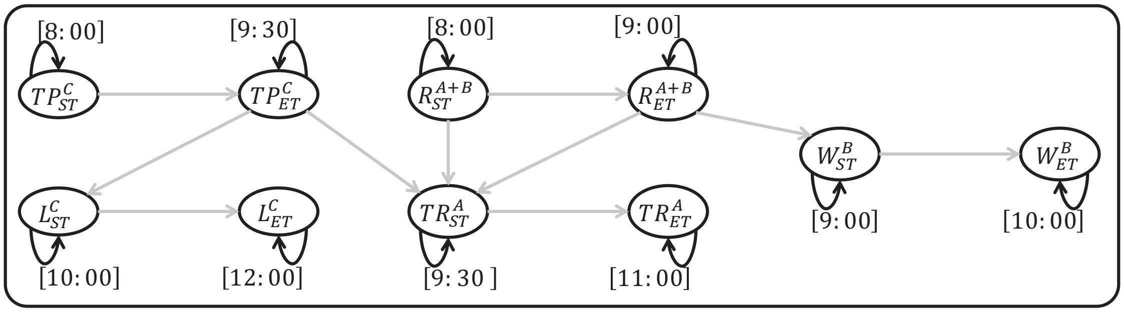
For instance, an STP can be used to schedule the morning agendas of Ann, her friend Bill, and her doctor Chris by representing events such as the start time, ST, and end time, ET, of each activity as variables, and capturing the minimum and maximum durations of time between events as constraints. Ann will have a 60 minute recreational activity (R^{A+B}) with Bill before spending 90 to 120 minutes performing a physical therapy regimen (TR^{A}) to help rehabilitate an injured knee (after receiving the prescription left by her doctor Chris); Bill will spend 60 minutes in recreation (R^{A+B}) with Ann before spending 60 to 180 minutes at work (W^{B}); and finally, Chris will spend 90 to 120 minutes planning a physical therapy regimen (TP^{C}) for Ann and drop it off before giving a lecture (L^{C}) from 10:00 to 12:00. This example scheduling problem is displayed graphically in Figure 1a as a Simple Temporal Network (STN)—a network of temporal constraints where each variable appears as a vertex with its domain of possible clock times described by a self-loop, and constraints appear as weighted edges. Figure 1b represents one example solution—an assignment of specific times to each event that respects constraints.

Suppose that Ann, Bill, and Chris would each like to task a personal computational scheduling agent with coordinating and managing the possible schedules that can accomplish his/her agenda while also protecting his/her interests. Unfortunately, current solution algorithms for solving STPs require representing the entire problem as a single, centralized STP, as shown in Figure 1, in order to calculate a (space of) solution schedule(s) for all agents. The computation, communication, and privacy costs associated with centralization may be unacceptable in multiagent planning and scheduling applications, such as military, health care, or disaster relief, where users specify problems to agents in a distributed fashion, and agents are expected to provide unilateral, time-critical, and coordinated scheduling assistance, to the extent possible.

This motivates our first major contribution: a formal definition of the Multiagent Simple Temporal Problem (MaSTP). The MaSTP, along with its associated Multiagent Simple Temporal Network (MaSTN) are *distributed* representations of scheduling problems and their solutions. The MaSTN corresponding to our running example is displayed in Figure 2a. Like their centralized counterparts, MaSTPs can be used to help solve *multiagent* planning problems and to monitor *multiagent* plan execution in a distributed manner. This representation allows each person to independently specify a scheduling problem to his or her own scheduling agent, and allows each agent, in turn, to independently maintain infor-



(a) The example problem represented as a network of temporal constraints.

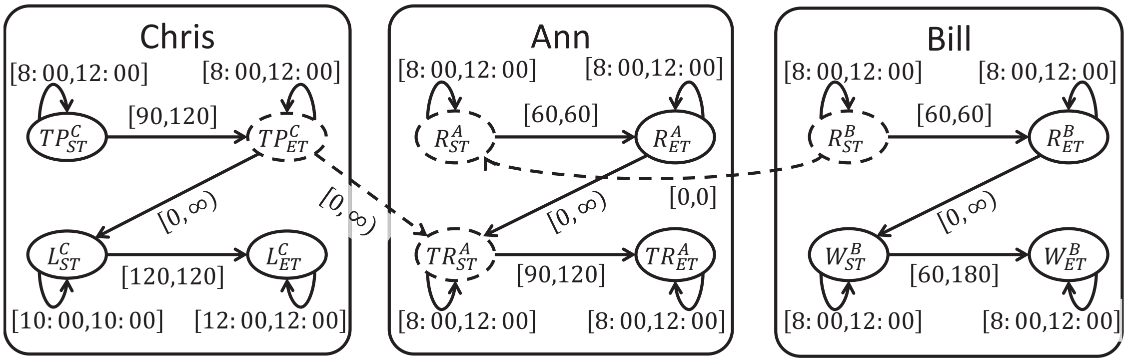


(b) A fully assigned point solution to the example problem.

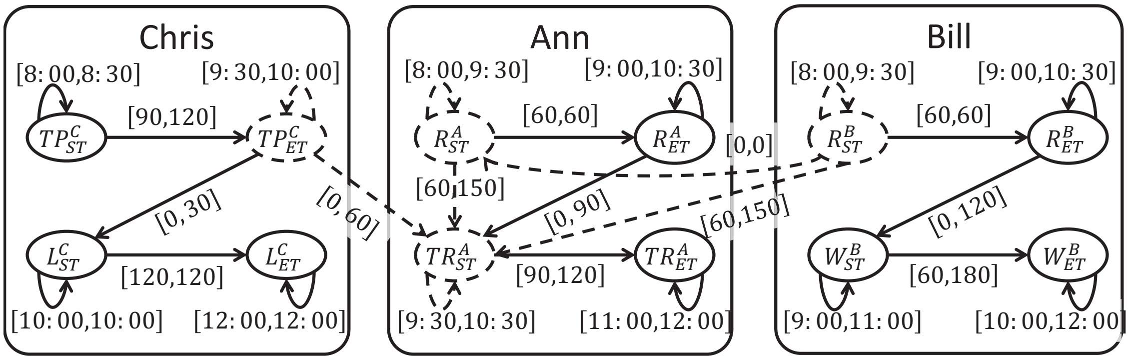
Figure 1: A typical, centralized STP representation of example problem.

mation about the interrelated activities of its user. Each scheduling agent, then, can use this information to provide its user with exact times when queried about possible timings and relationships between events, without unnecessarily revealing this information to other agents.

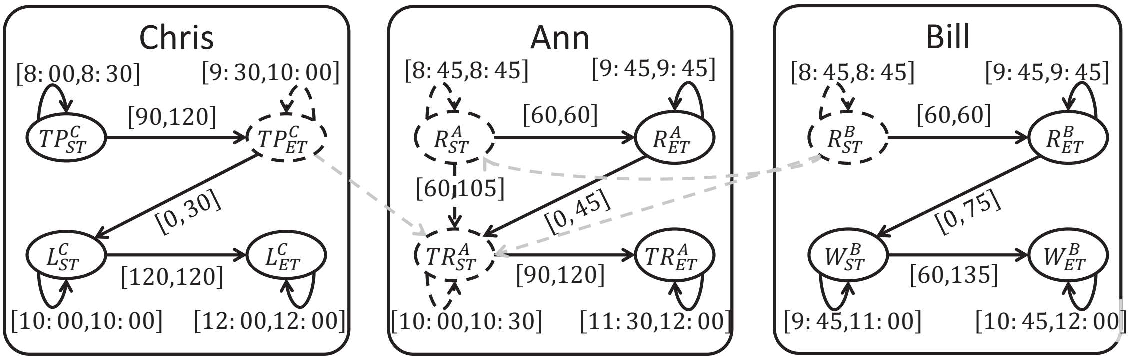
The second major contribution of this paper is a distributed algorithm for computing the complete joint solution space, as displayed in Figure 2b. This leads to advice that is more robust to disturbances and more accommodating to new constraints than finding single joint solutions like in Figure 1b. An advantage of this approach is that if a new constraint arrives (e.g., Chris' bus is late), the agents can easily recover by simply eliminating newly invalidated joint schedules from consideration. As agents make choices or adjust to new constraints, our distributed algorithm can simply be reapplied to recompute the resulting narrower space of remaining possible joint schedules, such that subsequent choices are assured to also satisfy constraints. So long as agents wait for the ramifications of choices/adjustments to propagate before making other choices, they eventually converge to a schedule that works, rather than prematurely picking a brittle schedule that needs to be replaced. However, this flexibility comes at the cost of ongoing communication, along with coordinating the distributed decisions to ensure the implications of one agent's decision propagates so that another agent's decision will be compatible. For example, if Ann decides to start recreating at 8:30, Bill's agent must wait for this decision to propagate before advising Bill on his options of when he can recreate, otherwise he might inadvertently choose a different time.



(a) Our first contribution is the Multiagent Simple Temporal Network representation, which allows distributed representation of multiagent STPs and their solutions.



(b) Our second contribution is a new distributed algorithm for computing the complete space of solutions to an MaSTP.



(c) Our third contribution is a new distributed temporal decoupling algorithm for computing locally independent solution spaces.

Figure 2: A summary of our contributions as applied to the example problem.

This motivates our third major contribution: a distributed algorithm for computing a temporal decoupling. A ***temporal decoupling*** is composed of *independent spaces* of locally consistent schedules that, when combined, form a space of consistent joint schedules (Hunsberger, 2002). An example of a temporal decoupling is displayed in Figure 2c, where, for example, Chris' agent has agreed to prescribe the therapy regimen by 10:00 and Ann's agent has agreed to wait to begin performing it until after 10:00. Here, not only will the agents' advice be independent of other agents', but the temporal decoupling also provides agents with some resiliency to new constraints and enhances users' flexibility and autonomy in making their own scheduling decisions. Now when Chris' bus is late by a minute, Chris' agent can absorb this new constraint by independently updating its local solution space. The advantage of this approach is that once agents establish a temporal decoupling, there is no need for further communication unless a new constraint renders the chosen decoupling inconsistent. It is only if and when a temporal decoupling does become inconsistent (e.g., Chris' bus is more than a half hour late, causing her to violate her commitment to finish the prescription by 10:00) that agents must calculate a new temporal decoupling (perhaps establishing a new hand-off deadline of 10:15), and then once again independently react to newly-arriving constraints, repeating the process as necessary.

The rest of our paper is structured to more fully describe each of these contributions. Section 2 builds foundational background necessary for understanding our contributions. Section 3 formally defines the MaSTP and explains important properties of the corresponding MaSTN. Section 4 describes our distributed algorithm for computing complete MaSTP solution spaces, analytically proving the algorithm's correctness and runtime properties, and empirically demonstrating the speedup it achieves over previous centralized algorithms. Section 5 describes our distributed algorithm for decoupling MaSTP solution spaces, again proving the algorithm's correctness and runtime properties, and empirically comparing its performance to previous art to demonstrate the trade-offs in completeness *versus* independence in reasoning. We give an overview of related approaches in Section 6 and conclude our discussion in Section 7.

2. Background

In this section, we provide definitions necessary for understanding our contributions, using and extending terminology from previous literature.

2.1 Simple Temporal Problem

As defined by Dechter et al. (1991), the Simple Temporal Problem (STP), $\mathcal{S} = \langle X, C \rangle$, consists of a set of n timepoint variables, X, and a set of m temporal difference constraints, C. Each timepoint variable represents an event and has a continuous domain of values (e.g., clock times) that can be expressed as a constraint relative to a special ***zero timepoint*** variable, $z \in X$, which represents the start of time. Each temporal difference constraint c_{ij} is of the form $x_j - x_i \leq b_{ij}$, where x_i and x_j are distinct timepoints, and $b_{ij} \in \mathbb{R} \cup \{\infty\}$ is a real number bound on the difference between x_j and x_i. Often, as notational convenience, two constraints, c_{ij} and c_{ji}, are represented as a single constraint using a bound interval of the form $x_j - x_i \in [-b_{ji}, b_{ij}]$.

	Availability	**Duration**	**Ordering**	**External**
Ann	$R^A_{ST} - z \in [480, 720]$	$R^A_{ET} - R^A_{ST} \in [60, 60]$	$R^A_{ET} - TR^A_{ST} \leq 0$	$R^A_{ST} - R^B_{ST} \in [0, 0]$
	$R^A_{ET} - z \in [480, 720]$			
	$TR^A_{ST} - z \in [480, 720]$	$TR^A_{ET} - TR^A_{ST} \in [90, 120]$		$TP^C_{ET} - TR^A_{ST} \leq 0$
	$TR^A_{ET} - z \in [480, 720]$			
Bill	$R^B_{ST} - z \in [480, 720]$	$R^B_{ET} - R^B_{ST} \in [60, 60]$	$R^B_{ET} - W^B_{ST} \leq 0$	$R^A_{ST} - R^B_{ST} \in [0, 0]$
	$R^B_{ET} - z \in [480, 720]$			
	$W^B_{ST} - z \in [480, 720]$	$W^B_{ET} - W^B_{ST} \in [60, 180]$		
	$W^B_{ET} - z \in [480, 720]$			
Chris	$TP^C_{ST} - z \in [480, 720]$	$TP^C_{ET} - TP^C_{ST} \in [90, 120]$	$TP^C_{ET} - L^C_{ST} \leq 0$	$TP^C_{ET} - TR^A_{ST} \leq 0$
	$TP^C_{ET} - z \in [480, 720]$			
	$L^C_{ST} - z \in [600, 600]$	$L^C_{ET} - L^C_{ST} \in [120, 120]$		
	$L^C_{ET} - z \in [720, 720]$			

Table 1: Summary of the constraints for the running example problem.

A ***schedule*** is an assignment of specific time values to timepoint variables. An STP is ***consistent*** if it has at least one ***solution***, which is a schedule that respects all constraints.

In Table 1, we formalize the running example (introduced in Section 1) with specific constraints. Each activity has two timepoint variables representing its start time (ST) and end time (ET), respectively. In this example, all activities are to be scheduled in the morning (8:00-12:00), and so are constrained (Availability column) to take place within 480 to 720 minutes of the zero timepoint z, which in this case represents midnight. Duration constraints are specified with bounds over the difference between an activity's end time and start time. Ordering constraints dictate the order in which an agent's activities must take place with respect to each other. Finally, while a formal introduction of external constraints is deferred until later (Section 3), the last column represents constraints that span the subproblems of different agents. Figure 1b illustrates a schedule that represents a solution to this particular problem.

2.2 Simple Temporal Network

To exploit extant graphical algorithms (e.g., shortest path algorithms) and efficiently reason over the constraints of an STP, each STP is associated with a Simple Temporal Network (STN), which can be represented by a weighted, directed graph, $\mathcal{G} = \langle V, E \rangle$, called a ***distance graph*** (Dechter & Pearl, 1987). The set of vertices V contains a vertex v_i for each timepoint variable $x_i \in X$, and E is a set of directed edges, where, for each constraint c_{ij} of the form $x_j - x_i \leq b_{ij}$, a directed edge e_{ij} from v_i to v_j is constructed with an initial weight $w_{ij} = b_{ij}$. As a graphical short-hand, each edge from v_i to v_j is assumed to be bi-directional, capturing both edge weights with a single interval label, $[-w_{ji}, w_{ij}]$, where

$v_j - v_i \in [-w_{ji}, w_{ij}]$ and w_{ij} or w_{ji} is initialized to ∞ if there exists no corresponding constraint $c_{ij} \in C$ or $c_{ji} \in C$, respectively. An STP is consistent if and only if there exist no negative cycles in the corresponding STN distance graph.

The ***reference edge***, e_{zi}, of a timepoint v_i is the edge between v_i and the zero timepoint z. As another short-hand, each reference edge e_{zi} is represented graphically as a self-loop on v_i. This self-loop representation underscores how a reference edge e_{iz} can be thought of as a unary constraint that implicitly defines v_i's domain, where w_{zi} and w_{iz} represent the earliest and latest times that can be assigned to v_i, respectively. In this work, we will assume that z is always included in V and that, during the construction of $\mathcal{G}$, a reference edge e_{zi} is added from z to every other timepoint variable, $v_i \in V_{\neg z}$.

The graphical STN representation of the example STP given in Table 1 is displayed in Figure 2a. For example, the duration constraint $TR^A_{ET} - TR^A_{ST} \in [90, 120]$ is represented graphically with a directed edge from TR^A_{ST} to TR^A_{ET} with label $[90, 120]$. Notice that the label on the edge from R^B_{ET} to W^B_{ST} has an infinite upper bound, since while Bill must start work after he ends recreation, there is no corresponding constraint given for how soon after he ends recreation this must occur. Finally, the constraint $L^C_{ST} - z \in [600, 600]$ is translated to a unary loop on L^C_{ST}, with a label of [10:00,10:00], which represents that Chris is constrained to start the lecture at exactly 600 minutes after midnight (or at exactly 10:00 AM). Throughout this work, we use both STP and STN notation. The distinction is that STP notation captures properties of the original problem, such as which pair of variables are constrained with which bounds, whereas STN notation is a convenient, graphical representation of STP problems that agents can algorithmically manipulate in order to find solutions by, for example, capturing implied constraints as new or tightened edges in the graph.

2.3 Useful Simple Temporal Network Properties

Temporal networks that are *minimal* and *decomposable* provide an efficient representation of an STP's solution space that can be useful to advice-giving scheduling agents.

2.3.1 Minimality

A ***minimal edge*** e_{ij} is one whose interval bounds, w_{ij} and w_{ji}, *exactly* specify the set of *all* values for the difference $v_j - v_i \in [-w_{ji}, w_{ij}]$ that are part of any solution (Dechter et al., 1991). A temporal network is ***minimal*** if and only if all of its edges are minimal. A minimal network is a representation of the solution space of an STP. For example, Figure 2b is a minimal STN, whereas Figure 2a is not, since it would allow Ann to start recreation at, say, 9:31 and Figure 2c also is not, since it does not allow Ann to start at 9:30. More practically, a scheduling agent can use a minimal representation to exactly and efficiently suggest scheduling possibilities to users without overlooking options or suggesting options that will not work.

2.3.2 Decomposability

Decomposability facilitates the maintenance of minimality by capturing constraints that, if satisfied, will lead to global solutions. A temporal network is ***decomposable*** if any assignment of values to a subset of timepoint variables that is locally consistent (satisfies all constraints involving only those variables) can be extended to a solution (Dechter et al., 1991). An STN is typically made decomposable by explicitly adding otherwise implicit constraints and/or tightening the weights of existing edges or variable domains so that only provably consistent values remain. For example, Figure 1b is trivially decomposable, since assigning a variable a single value within its minimal domain assures that it will be part of a solution. Figure 2b, on the other hand, is not decomposable, since, for instance, the assignment $R^A_{ET} = 10{:}30$ and $TR^A_{ET} = 11{:}00$, while self-consistent (because there are no constraints that directly link these two variables), cannot be extended to a full solution. A scheduling agent can use a decomposable temporal network to directly propagate any newly-arriving constraint to any other area of the network in a single-step, backtrack-free manner.

In sum, an STN that is both minimal and decomposable represents the entire set of solutions by establishing the tightest bounds on timepoint variables such that (i) no solutions are eliminated and (ii) any self-consistent assignment of specific times to a subset of timepoint variables that respects these bounds can be extended to a solution in an efficient, backtrack-free manner.

2.4 Simple Temporal Problem Consistency Algorithms

In this subsection, we highlight various existing algorithms that help establish the STN solution properties introduced in the previous subsection.

2.4.1 Full Path Consistency

Full Path Consistency (FPC) establishes minimality and decomposability of an STP instance in $\mathcal{O}\left(n^3\right)$ time (recall $n = |V|$) by applying an all-pairs-shortest-path algorithm, such as Floyd-Warshall (1962), to its STN, resulting in a fully-connected distance graph (Dechter et al., 1991). The Floyd-Warshall algorithm, presented as Algorithm 1, finds the tightest possible path between every pair of timepoints, v_i and v_j, in a fully-connected distance graph, where $\forall i, j, k, w_{ij} \leq w_{ik} + w_{kj}$. This graph is then checked for consistency by validating that there are no negative cycles, that is, $\forall i \neq j$, ensuring that $w_{ij} + w_{ji} \geq 0$. An example of the FPC version of Ann's STP subproblem is presented in Figure 3a. Note that an agent using an FPC representation can provide exact bounds over the values that will lead to solutions for *any* pair of variables, regardless of whether or not a corresponding constraint was present in the original STP formulation.

2.4.2 Graph Triangulation

The next two forms of consistency require a ***triangulated*** (also called chordal) temporal network. A triangulated graph is one whose largest non-bisected cycle is a triangle (of length three). Conceptually, a graph is triangulated by the process of considering vertices and their adjacent edges, one by one, adding edges between neighbors of the currently-considered vertex if no edge previously existed, and then eliminating that vertex from further consideration, until all vertices are eliminated (Golumbic, 1980). This basic graph triangulation process is presented as Algorithm 2. The set of edges that are added during this process are called ***fill edges*** and the order in which timepoints are eliminated from consideration is referred to as an ***elimination order***, o. Figure 3b shows the topology

Algorithm 1: Floyd-Warshall

```
Input: A fully-connected distance graph $\mathcal{G} = \langle V, E \rangle$
Output: A FPC distance graph $\mathcal{G}$ or INCONSISTENT
for $k = 1 \ldots n$ do
    for $i = 1 \ldots n$ do
        for $j = 1 \ldots n$ do
            $w_{ij} \leftarrow \min(w_{ij}, w_{ik} + w_{kj})$
            if $w_{ij} + w_{ji} < 0$ then
                return INCONSISTENT
return $\mathcal{G}$
```

Algorithm 2: Triangulate

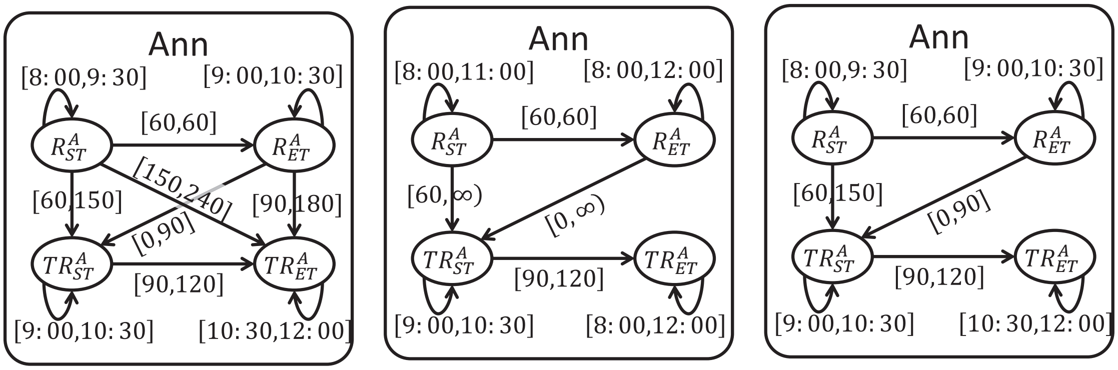
```
Input: A distance graph $\mathcal{G} = \langle V, E \rangle$; and elimination order $o = (v_1, v_2, \ldots, v_{n-1}, v_n)$
Output: A triangulated distance graph $\mathcal{G}$
for $k = 1 \ldots n$ do
    forall $i, j > k$ s.t. $e_{ik}, e_{jk} \in E$ do
        $E \leftarrow E \cup \{e_{ij}\}$
return $\mathcal{G}$
```

of a triangulated version of Ann's distance graph where timepoints are eliminated in order $o = (TR^A_{ET}, R^A_{ET}, R^A_{ST}, TR^A_{ST})$.

The quantity ω^*_o is the ***induced graph width*** of the distance graph relative to o, and is defined as the maximum, over all v_k, of the size of v_k's set of not-yet-eliminated neighbors at the time of its elimination. The triangulate algorithm, then, operates in $\mathcal{O}\left(n \cdot \omega^{*2}_o\right)$ time. Elimination orders are often chosen to attempt to find a minimal triangulation, that is to attempt to minimize the total number of fill edges. While, generally speaking, finding the minimum triangulation of a graph is an NP-complete problem, heuristics such as ***minimum degree*** (selecting the vertex with fewest edges) and ***minimum fill*** (selecting the vertex that adds fewest fill edges) are used to efficiently find elimination orders that approximate the minimum triangulation (Kjaerulff, 1990).

2.4.3 Directional Path Consistency

An alternative to FPC for checking STP consistency is to establish ***Directional Path Consistency*** (DPC) on its temporal network. The DPC algorithm (Dechter et al., 1991), presented as Algorithm 3, takes a triangulated graph and corresponding elimination order, o, as input. It traverses each timepoint, v_k, in order o, tightening edges between each pair of neighboring timepoints, v_i, v_j, (where v_i and v_j are connected to v_k via edges e_{ik} and e_{jk} respectively) that appear after v_k in order o, using the rule $w_{ij} \leftarrow \min(w_{ij}, w_{ik} + w_{kj})$. The time complexity of DPC is $\mathcal{O}\left(n \cdot \omega^{*2}_o\right)$, but instead of establishing minimality, it establishes the property that a solution can be recovered from the DPC distance graph in a backtrack-free manner if variables are assigned in reverse elimination order. An example



(a) Ann's FPC STN. (b) Ann's DPC STN. (c) Ann's PPC STN.

Figure 3: Alternative forms of consistency applied to Ann's STP.

Algorithm 3: Directional Path Consistency (DPC)

```
Input: A triangulated temporal network $\mathcal{G} = \langle V, E \rangle$ and corresponding elimination
       order $o = (v_1, v_2, \ldots, v_{n-1}, v_n)$
Output: A DPC distance graph $\mathcal{G}$ or INCONSISTENT
for $k = 1 \ldots n$ do
    forall $i > k$ s.t. $e_{ik} \in E$ do
        forall $j > i$ s.t. $e_{jk} \in E$ do
            $w_{ij} \leftarrow \min(w_{ij}, w_{ik} + w_{kj})$
            $w_{ji} \leftarrow \min(w_{ji}, w_{jk} + w_{ki})$
            if $w_{ij} + w_{ji} < 0$ then
                return INCONSISTENT
return $\mathcal{G}$
```

of a DPC version of Ann's problem, using elimination order $o = (TR^A_{ET}, R^A_{ET}, R^A_{ST}, TR^A_{ST})$, is presented in Figure 3b. Basically, establishing DPC is sufficient for establishing whether consistent schedules exist or not, but limits a scheduling agent to giving useful advice over only the last variable to be eliminated.

In Section 4.1, we discuss how we combine the reasoning of the DPC and triangulate algorithms so that their execution can be distributed. Combining this reasoning yields an example of a bucket-elimination algorithm. Bucket-elimination algorithms (Dechter, 1999) are a general class of algorithms for calculating ***knowledge compilations***, solution space representations from which solutions can be extracted in a backtrack-free, linear-time manner. The ***adaptive consistency*** algorithm (Dechter & Pearl, 1987; Dechter, 2003) calculates a knowledge compilation for general constraint satisfaction problems (Dechter, 1999). Adaptive consistency eliminates variables one by one, and for each variable that it eliminates, reasons over the "bucket" of constraints the variable is involved with to deduce new constraints over the remaining non-eliminated variables. Any solution to the remain-

Algorithm 4: Planken's Partial Path Consistency (P^3C)

```
Input: A triangulated temporal network $\mathcal{G} = \langle V, E \rangle$ and an elimination order
       $o = (v_1, v_2, \ldots, v_{n-1}, v_n)$
Output: A PPC distance graph $\mathcal{G}$ or INCONSISTENT
$\mathcal{G} \leftarrow DPC(\mathcal{G}, o)$
return INCONSISTENT if DPC does
for $k = n \ldots 1$ do
    forall $i, j > k$ s.t. $e_{ik}, e_{jk} \in E$ do
        $w_{ik} \leftarrow \min(w_{ik}, w_{ij} + w_{jk})$
        $w_{kj} \leftarrow \min(w_{kj}, w_{ki} + w_{ij})$
return $\mathcal{G}$
```

ing subproblem can be extended to a solution to the original problem, since the solution accounts for all constraints entailed by the eliminated variable. DPC is an important step in establishing Partial Path Consistency, as we discuss next.

2.4.4 Partial Path Consistency

Bliek and Sam-Haroud (1999) demonstrate that ***Partial Path Consistency*** (PPC) is sufficient for establishing minimality on an STP instance by calculating the tightest possible path for *only* the subset of edges that exist within a triangulated distance graph. An example is Xu and Choueiry's algorithm △STP (Xu & Choueiry, 2003), which processes and updates a queue of all potentially inconsistent triangles (△) from the triangulated graph. Alternatively, in their algorithm P^3C, Planken et al. (2008) sweep through these triangles in a systematic order, which gives a guaranteed upper bound on its runtime, while no such bound is known for △STP. The P^3C algorithm, included as Algorithm 4 executes the DPC algorithm as a first phase and then executes a reverse traversal of the DPC algorithm as a second phase, where edge weights are updated in reverse elimination order. Thus P^3C achieves the same complexity, $\mathcal{O}\left(n \cdot \omega_o^{*2}\right)$, as the DPC algorithm. By exploiting sparse network topology, PPC-based algorithms may establish minimality much faster than FPC algorithms in practice ($\mathcal{O}\left(n \cdot \omega_o^{*2}\right) \subseteq \mathcal{O}\left(n^3\right)$) (Xu & Choueiry, 2003; Planken et al., 2008). The PPC representation of Ann's subproblem using elimination order $o = (TR^A_{ET}, R^A_{ET}, R^A_{ST}, TR^A_{ST})$ is displayed in Figure 3c.

PPC only approximates decomposability since only assignments to fully-connected subsets of variables (those that belong to the same clique in the triangulated network) are guaranteed to be extensible to a full solution. However solutions can still be recovered in a backtrack-free manner by either requiring constraint propagation between each subsequent variable assignment or by assigning variables in any reverse ***simplicial elimination order***—any elimination order of variables that yields the same triangulated network (that is, introduces no new fill edges) (Planken et al., 2008). An agent using a PPC representation can offer advice over any pair of variables that share an edge in the triangulated graph—those that were originally related via a constraint in the original formulation and those connected by fill edges. While, unlike FPC temporal networks, an agent using a

PPC network cannot answer queries regarding arbitrary pairs of variables (i.e., those that do not share an edge), the sparser PPC structure will have important benefits for agents' independent and private reasoning, as discussed in Section 3.1.2.

2.5 The Temporal Decoupling Problem

Hunsberger (2002) formally defined the concept of a ***temporal decoupling*** for STPs. A partitioning of an STP $\mathcal{S}$'s variables into two sets, V^A and V^B, leads naturally to the definition of two sub-STPs, $\mathcal{S}^A = \langle V^A, C^A \rangle$ and $\mathcal{S}^B = \langle V^B, C^B \rangle$, where C^A and C^B are the constraints defined exclusively over the variables in V^A and V^B, respectively. Then $\mathcal{S}^A$ and $\mathcal{S}^B$ form a temporal decoupling of $\mathcal{S}$ if:

- $\mathcal{S}^A$ and $\mathcal{S}^B$ are consistent STPs; and
- Merging *any* locally consistent solutions to the problems in $\mathcal{S}^A$ and $\mathcal{S}^B$ yields a solution to $\mathcal{S}$.

Notice that a temporal decoupling exists if and only if the original STP is consistent. Alternatively, when $\mathcal{S}^A$ and $\mathcal{S}^B$ form a temporal decoupling of $\mathcal{S}$, $\mathcal{S}^A$ and $\mathcal{S}^B$ are said to be ***temporally independent***. The Temporal Decoupling Problem (TDP), then, is defined as finding two sets of *decoupling constraints*, C^A_Δ and C^B_Δ, such that if C^A_Δ and C^B_Δ are combined with $\mathcal{S}^A$ and $\mathcal{S}^B$ to form $\mathcal{S}^A_\Delta = \langle V^A, C^A \cup C^A_\Delta \rangle$ and $\mathcal{S}^B_\Delta = \langle V^B, C^B \cup C^B_\Delta \rangle$ respectively, then $\mathcal{S}^A_\Delta$ and $\mathcal{S}^B_\Delta$ form a temporal decoupling of STP $\mathcal{S}$. A ***minimal decoupling*** is one where, if the bound of any decoupling constraint in either C^A_Δ or C^B_Δ is relaxed (increasing the bound so that the constraint is more inclusive) or removed, then $\mathcal{S}^A$ and $\mathcal{S}^B$ no longer form a decoupling. The original TDP algorithm (Hunsberger, 2002) executes centrally and iterates between proposing new constraints to add to C^A_Δ and C^B_Δ and propagating these constraints to reestablish FPC on the corresponding global distance graph so that subsequently proposed decoupling constraints are guaranteed to be consistent. An iteration occurs for each constraint that spans between $\mathcal{S}^A$ and $\mathcal{S}^B$ until all such constraints have been rendered moot due to new decoupling constraints.

A temporal decoupling trades a complete solution space with possibly messy interdependencies for a partial solution space with nice independence properties. Independent reasoning, which can be critical in applications that must provide time-critical, unilateral scheduling advice in environments where communication is costly or uncertain, comes at the cost of eliminating valid joint solutions. In Section 4.3, we will present various *flexibility metrics* that quantify the portion of the solution space that is retained by a given temporal decoupling, and use these to quantify this trade-off.

3. The Multiagent Simple Temporal Problem

The problem that this paper addresses is that of developing a compact, distributed representation of, and distributed algorithms for finding (a temporal decoupling of) the joint solution space of, multiagent scheduling problems. This section formally defines the distributed representation in the form of the *Multiagent* Simple Temporal Problem, extends the definitions of minimality and decomposability to this representation, and characterizes independence and privacy within this representation. Sections 4 and 5 will then describe distributed algorithms that find either complete or temporally decoupled solution spaces for providing users with flexible and sound scheduling alternatives. These algorithms avoid unnecessarily centralizing or redistributing agents' subproblems and also achieve significant speedup over current state-of-the-art approaches.

3.1 Multiagent Simple Temporal Problem Formulation

The ***Multiagent Simple Temporal Problem (MaSTP)*** is composed of a local STP subproblems, one for each of the a agents, and a set of constraints C_X that establish relationships between the local subproblems of different agents (Boerkoel & Durfee, 2010, 2011; Boerkoel, 2012). An agent i's ***local STP*** subproblem is defined as $\mathcal{S}_L^i = \langle V_L^i, C_L^i \rangle^1$, where:

- V_L^i is defined as agent i's set of ***local variables***, which is composed of all timepoints *assignable* by agent i along with a variable representing agent i's reference to z; and

- C_L^i is defined as agent i's set of intra-agent or ***local constraints***, where a local constraint, $c_{ij} \in C_L^i$, is defined as a bound b_{ij} on the difference between two local variables, $v_j - v_i \leq b_{ij}$, where $v_i, v_j \in V_L^i$.

In Figure 2a, the variables and constraints entirely within the boxes labeled Chris, Ann, and Bill represent each person's respective local STP subproblem from the running example. Notice, the sets V_L^i partition the set of all (non-reference) timepoint variables, $V_{\neg z}$.

C_X is the set of inter-agent or ***external constraints***, where an external constraint is defined as a bound on the difference between two variables that are local to different agents, $v_i \in V_L^i$ and $v_j \in V_L^j$, where $i \neq j$. However, each agent knows only the subset of external constraints that involve its local timepoints and, as a by-product of these external constraints, is also aware of a subset of non-local variables, where:

- C_X^i is agent i's set of ***external constraints***, each of which involves exactly one of agent i's local timepoint variables (since all constraints are inherently binary); and

- V_X^i is agent i's set of ***external timepoint variables***, which are local to some other agent $j \neq i$, but are known to agent i due to an external constraint in C_X^i.

The sets of all external constraints is $C_X = \bigcup_i C_X^i$ and set of all external variables is $V_X = \bigcup_i V_X^i$. Together, an agent i's set of *known timepoints* is $V_L^i \cup V_X^i$ and its set of *known constraints* is $C_L^i \cup C_X^i$. Note that this assumes that each constraint is known by each agent that has at least one variable involved in the constraint. In Figure 2a, external constraints and variables are denoted with dashed edges.

More formally, then, an MaSTP, $\mathcal{M}$, is defined as the STP $\mathcal{M} = \langle V_{\mathcal{M}}, C_{\mathcal{M}} \rangle$ where $V_{\mathcal{M}} = \bigcup_i V_L^i$ and $C_{\mathcal{M}} = C_X \cup \bigcup_i C_L^i$.

---

[1] Throughout this paper, we will use superscripts to index agents and subscripts to index variables and constraints/edges.

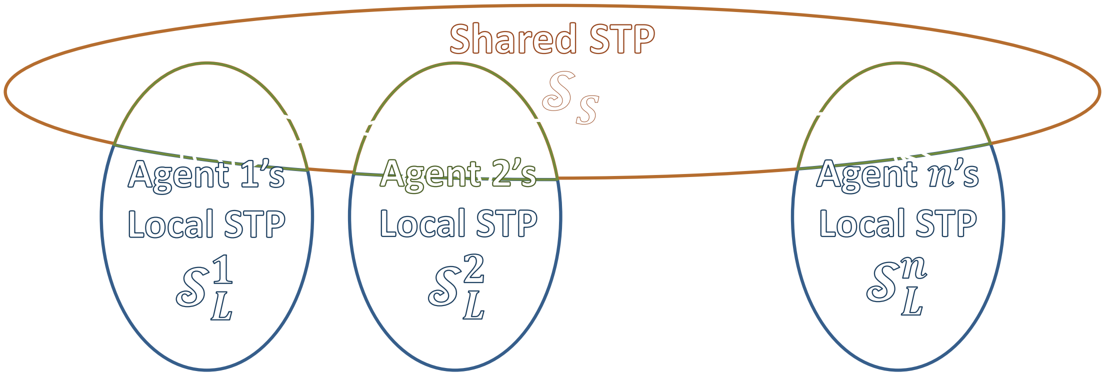


Figure 4: High-level overview of the MaSTP structure.

3.1.1 Minimality and Decomposability

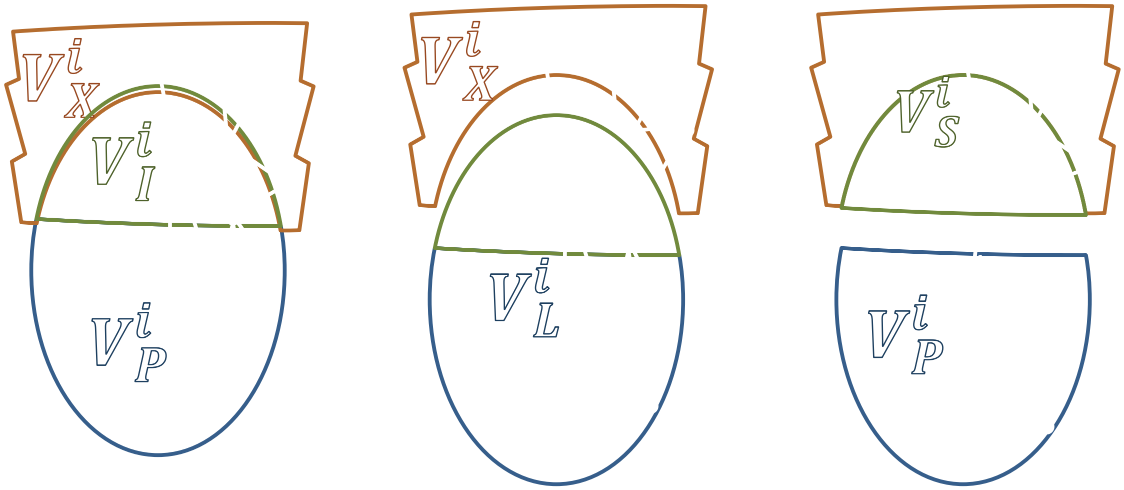
Up until this point, we have discussed the MaSTP as a problem formulation. However, we can also discuss properties of the corresponding *Multiagent Simple Temporal Network (MaSTN)*. An MaSTP is converted to an equivalent MaSTN in the same way that an STP is converted into an STN, where the definition of agent i's local and external edges, E_L^i and E_X^i, follows analogously from the definition of C_L^i and C_X^i, respectively. Like in the STN, the MaSTN can be algorithmically manipulated to represent useful information. Here, we will discuss how STN properties such as minimality and decomposability translate to the MaSTN.

Because an MaSTN is an STN (albeit a decentralized one), properties such as minimality and decomposability extend unhindered to multiagent temporal networks. Thus, a minimal MaSTN is one where all the edges are minimal. Likewise, an MaSTN is decomposable if any self-consistent assignment of values to a subset of variables can be extended to a full joint solution.

Calculating an MaSTN that is both minimal and decomposable requires computing a fully-connected network, which, in a multiagent setting, clobbers all independence between agents' subproblems: each of an agent's timepoints will now be connected to every other timepoint of every other agent. Full connectivity obliterates privacy (since now $\forall v \in V_L^i, v \in V_X^j \forall j \neq i$) and requires that every scheduling decision be coordinated among all agents. For this reason, our work focuses on instead establishing partial path consistency to approximate decomposability while retaining the loosely-coupled structure that may exist in an MaSTN.

3.1.2 An Algorithm-Centric MaSTP Partitioning

The MaSTP formalization just presented naturally captures MaSTPs using an agent-centric perspective. However, algorithmically, it is often easier to discuss an MaSTP in terms of which parts of the problem an agent can solve independently, and which parts require shared effort to solve. Thus, here we introduce some additional terminology that helps improve the precision and comprehension of both our algorithmic descriptions and our analytical arguments.



(a) Each agent has external, interface, and private timepoints.

(b) Local *vs.* External.

(c) Private *vs.* Shared.

Figure 5: Alternative partitionings of an agent's STP.

The natural distribution of the MaSTP representation affords a partitioning of the MaSTP into independent (private) and interdependent (shared) components. We start by defining the ***shared*** STP, $\mathcal{S}_S = \langle V_S, C_S \rangle$, which is composed of:

- $V_S = V_X \cup \{z\}$—the set of ***shared variables*** is comprised of all variables that are involved in at least one external constraint; and
- $C_S = \{c_{ij} \mid v_i, v_j \in V_S\}$—the set of ***shared constraints*** is defined as the constraints between pairs of shared variables, and includes the entire set of external constraints C_X, but could also include otherwise local constraints that exist between two shared variables belonging to a single agent.

Notice that, as illustrated in Figure 4, the shared STP overlaps with an agent's local subproblem, and thus divides each agent i's known timepoints, V^i, into three distinct sets:

- V_X^i—agent i's set of ***external variables***, defined as before;
- $V_I^i = V_L^i \cap V_S$—agent i's set of ***interface variables***, which is defined as agent i's set of local variables that are involved in one or more external constraints; and
- $V_P^i = V_L^i \setminus V_S$—agent i's set of ***private variables***, which is defined as agent i's local variables that are *not* involved in *any* external constraints.

These three sets of variables are depicted graphically in Figure 5a. Figure 5 also highlights the two alternate partitionings of an MaSTP into agent-centric local *versus* external components (Figure 5b) and algorithm-centric independent (private) *versus* interdependent (shared) components (Figure 5c). More formally, this allows us to define agent i's private subproblem, $\mathcal{S}_P^i = \langle V_P^i, C_P^i \rangle$, where agent i's set of ***private constraints***, $C_P^i = C_L^i \setminus C_S$, is the subset of agent i's local constraints that include at least one of its private variables.

The partitioning depicted in Figure 5c is useful algorithmically because it establishes which parts of an agent's subnetwork are independent of other agents (private), and which parts are inherently interdependent (shared). Notice that agent i's local constraints are included in its private subproblem as long as they include a private variable, even if they also involve a shared variable. This is because agent i is able to propagate changes to that constraint, and any other private constraint, without directly affecting a shared timepoint or constraint. Private edges connected to a shared timepoint appear to hang in Figure 5c because the shared timepoint is actually part of the shared subproblem.

3.1.3 Independence

Algorithms that use the distributed MaSTN representation to reason over scheduling problems that span multiple agents have strategic (e.g., privacy) and computational (e.g., concurrency) advantages. The extent to which these advantages can be realized largely depends on the level of independent reasoning that an agent is able to perform over its local problem. We define two timepoints as ***independent*** if there is no path that connects them in the MaSTN, and ***dependent*** otherwise. Notice that all dependencies between agents inherently flow through the set of shared variables, V_S. The implication is that each agent i can independently (and thus concurrently, asynchronously, privately, autonomously, etc.) reason over its private subproblem $\mathcal{S}_P^i$ independently of $\mathcal{S}_P^j \forall j \neq i$.

Theorem 1. *The only dependencies between agent i's local subproblem, $\mathcal{S}_L^i$, and another agent j's ($j \neq i$) local subproblem $\mathcal{S}_L^j$, exist exclusively through the shared STP, $\mathcal{S}_S$.*

Proof. By contradiction, assume there exist variables $v_i \in V_P^i$ and $v_j \in V_P^j$ such that v_i and v_j are *not* independent given $\mathcal{S}_S$. This implies that there exists a path in the constraint network between v_i and v_j that involves some pair of private variables $v_i' \in V_P^i$ and $v_j' \in V_P^j$ that are connected via a constraint. However, this is a contradiction, since v_i' and v_j' would, by definition, belong to V_S, and thus $\mathcal{S}_S$. Therefore, every pair of variables $v_i \in V_P^i$ and $v_j \in V_P^j$ are independent given $\mathcal{S}_S$. □

3.1.4 Privacy

In our work, we assume that agents are cooperative. However, at the same time, a user may still wish to avoid the gratuitous revelation of details about his or her schedule to the agents of other people, to the extent possible. We next look the privacy that is preserved as a byproduct of both the distributed problem representation and level of independent reasoning as established in Theorem 1. Obviously, any coordination between agents' activities has some inherent privacy costs. However, we show that these costs are limited to the shared timepoints and edges between them.

Notice that in Figure 2a the only variable of Ann's that Bill's agent starts out knowing is her recreational start time variable R_{ST}^A, due to Bill's shared constraint with Ann. However, while Ann's agent can establish PPC using a variety of different elimination orderings, Ann's agent alone cannot establish PPC over its timepoints without adding new external edges. This is because Ann's problem contains two different externally constrained timepoints that share a local constraint path, regardless of which elimination order is used. So, for example if Ann's agent were to eliminate both of its private timepoints (TR^A_{ET} and R^A_{ET}) and then eliminate R^A_{ST}, the triangulation process would construct a new external edge between shared timepoints TR^A_{ST} and R^B_{ST} as shown in Figure 2b. This effectively adds TR^A_{ST} to Bill's agent's set of external timepoints. Because TR^A_{ST} was already shared with Chris' agent, the fill edge between it and R^A_{ST} is a shared edge, and the triangulation process allows Bill's agent to become aware of these *shared* components of Ann's problem. The question becomes: can Bill's agent continue this process to draw inferences about any of Ann's *private* timepoints and edges? Theorem 2 shows that, without an exogenous source of information, an agent will *not* be able to infer the existence of, the number of, or bounds on another agent's private timepoints, even if they implicitly influence the agent's subproblem through shared constraint paths.

Theorem 2. *No agent can infer the existence or bounds of another agent's private edges, or subsequently the existence of private timepoints, solely from the shared STN.*

Proof. First, we prove that the existence or bounds of a private edge cannot be inferred from the shared STN. Assume agent i has a private edge, $e_{ik} \in E^i_P$. By definition, at least one of v_i and v_k is private; without loss of generality, assume $v_i \in V^i_P$. For every pair of edges e_{ij} and e_{jk} that are capable of entailing (the bounds of) e_{ik}, regardless of whether v_j is shared or private, $v_i \in V^i_P$ implies $e_{ij} \in E^i_P$ is private. Hence, any pair of edges capable of implying a private edge must also contain at least one private edge. Therefore, a private edge cannot be inferred from shared edges alone.

Thus, since an agent cannot extend its view of the shared STN to include another agent's private edges, it cannot infer another agent's private timepoints. □

Theorem 2 implies that $\mathcal{S}_S$ (the variables and constraints of which are represented with dashed lines in Figure 2b) represents the maximum portion of the MaSTP that agents can infer without an exogenous (or hypothesized) source of information, even if they collude to reveal the entire shared subnetwork. Hence, given the distribution of an MaSTP $\mathcal{M}$, if agent i executes a multiagent algorithm that does not reveal any of its private timepoints or constraints, it can be guaranteed that any agent $j \neq i$ will not be able to infer any private timepoint in V^i_P or private constraint in C^i_P by also executing the multiagent algorithm—at least not without requiring conjecture or ulterior (methods of inferring) information on the part of agent j. More generally, it is not necessary or inevitable that any one agent knows or infers the entire shared STP $\mathcal{S}_S$.

3.2 Multiagent Temporal Decoupling Problem

We next adapt the original definition of temporal decoupling (Hunsberger, 2002) as described in Section 2.5 to apply to the MaSTP. The set of agents' local STP subproblems $\{\mathcal{S}^1_L, \mathcal{S}^2_L, \ldots, \mathcal{S}^n_L\}$ form a ***temporal decoupling*** of an MaSTP $\mathcal{M}$ if:

- $\{\mathcal{S}^1_L, \mathcal{S}^2_L, \ldots, \mathcal{S}^n_L\}$ are consistent STPs; and
- Merging *any* combination of locally-consistent solutions to each of the problems in $\{\mathcal{S}^1_L, \mathcal{S}^2_L, \ldots, \mathcal{S}^n_L\}$ yields a solution to $\mathcal{M}$.

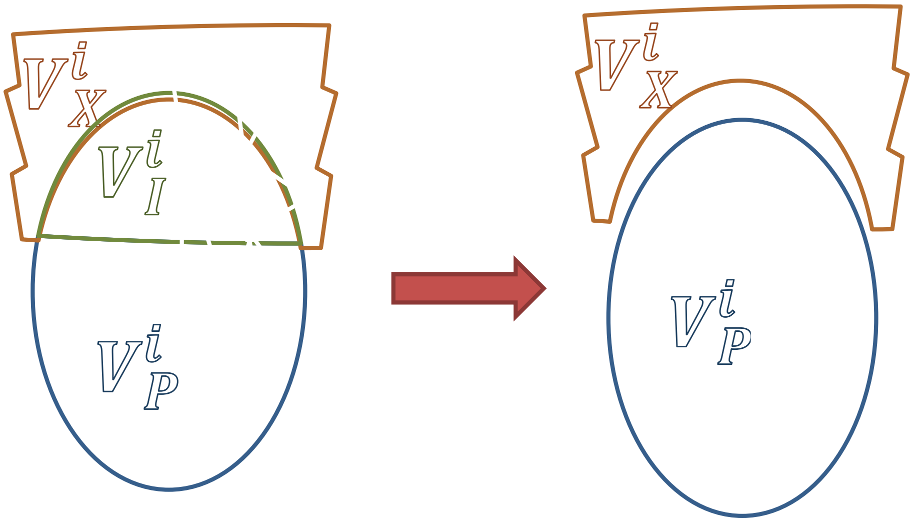


Figure 6: The temporal decoupling problem calculates new local constraints that render constraints between agents superfluous.

Alternatively, when $\{\mathcal{S}_L^1, \mathcal{S}_L^2, \ldots, \mathcal{S}_L^n\}$ form a temporal decoupling of $\mathcal{M}$, they are said to be ***temporally independent***. As illustrated in Figure 6, the objective of ***Multiagent Temporal Decoupling Problem (MaTDP)*** is to find a set of constraints C_Δ^i for each agent i such that if $\mathcal{S}_{L+\Delta}^i = \langle V_L^i, C_L^i \cup C_\Delta^i \rangle$, then $\{\mathcal{S}_{L+\Delta}^1, \mathcal{S}_{L+\Delta}^2, \ldots, \mathcal{S}_{L+\Delta}^n\}$ is a temporal decoupling of MaSTP $\mathcal{M}$. Note that solving the MaTDP does not mean that the agents' subproblems have somehow become inherently independent from each other (with respect to the original MaSTP), but rather that the new decoupling constraints provide agents a way to perform sound reasoning completely independently of each other.

Notice that new constraints, C_Δ^i, depicted with dotted lines in the right-hand side of Figure 6, allow the safe removal of the now superfluous external constraints involving agent i's local variables, and so the external constraints are also removed in the figure. Finally, notice that local variables and edges that were previously part of the shared problem (marked in the right-hand side of Figure 6 with double edges) can now be treated algorithmically as private. Figure 1b and Figure 2c both represent temporal decouplings of the example, where new or tighter unary decoupling constraints, in essence, replace all external edges (shown faded). A ***minimal decoupling*** is one where, if the bound of any decoupling constraint $c \in C_\Delta^i$ for some agent i is relaxed (or removed), then $\{\mathcal{S}_{L+\Delta}^1, \mathcal{S}_{L+\Delta}^2, \ldots, \mathcal{S}_{L+\Delta}^n\}$ is no longer a decoupling. Figure 2c is an example of a minimal decoupling whereas the *de facto* decoupling formed by a full assignment (such as the one in Figure 1b) is not minimal.

4. Distributed Algorithms for Calculating Partial Path Consistency

In this section, we introduce our Distributed Partial Path Consistency (D$\triangle$PPC) algorithm for establishing PPC on an MaSTN. As illustrated in Figure 7, our algorithm works by solving $a+1$ subproblems: a private agent subproblems and the one shared STP. Like the original P^{3}C algorithm (Algorithm 4 on page 105), each of these subproblems is solved

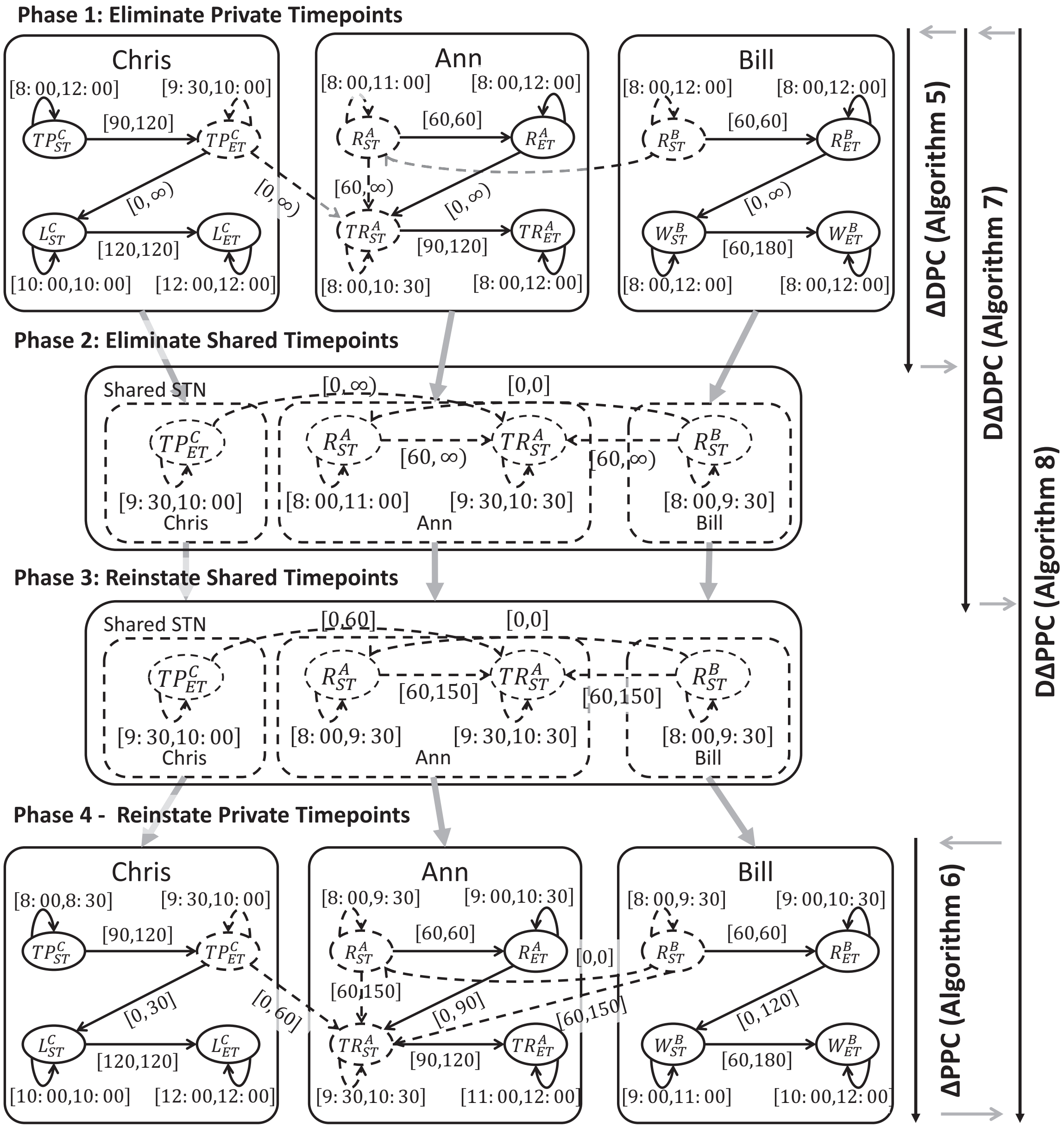


Figure 7: Our distributed PPC algorithms operate in four distinct phases.

in two phases: a forward sweep eliminates timepoints to compute implied constraints and establish DPC and a second, reverse sweep reinstates nodes and updates their incident edges to establish full minimality and PPC. In order to maximize concurrency and independence in our distributed version of this algorithm, we carefully partition execution into four phases. Agents work together to perform the forward (Phase 2, Figure 7) and reverse (Phase 3, Figure 7) sweeps of the P^3C algorithm respectively on the shared STP, but each agent *independently* performs the forward (Phase 1, Figure 7) and reverse (Phase 4, Figure 7) sweeps of the P^3C algorithm on its private subproblem.

In Section 4.1, we introduce our △DPC and △PPC algorithms, which both serve as subroutines of the D△PPC algorithm to establish DPC and PPC on each agent's private subproblem Our △DPC algorithm tweaks the original DPC algorithm so that each agent can independently triangulate and establish DPC over the private portion of its subproblem (Phase 1). Then, later in Phase 4, each agent executes △PPC to reinstate its private timepoints and complete the process of establishing PPC on its private subproblem. Next, in Section 4.2.1, we describe the D△DPC algorithm that carefully distributes the execution of the △DPC algorithm so that, after separately establishing DPC on their private STNs, agents work together to triangulate and update the remaining shared portion of the MaSTN in a globally consistent manner, thus establishing DPC on the MaSTN as a whole (Phases 1 and 2). Finally, we present our overarching D△PPC algorithm in Section 4.2.2. Each agent executes D△PPC separately. D△PPC first invokes D△DPC to establish DPC on the MaSTN (Phases 1 and 2), then executes a distributed version of △PPC's reverse sweep for agents to cooperatively establish PPC on the shared portion of their MaSTN (Phase 3), and finally invokes △PPC to finish PPC establishment on the agent's separate private subproblem (Phase 4). We conclude this section by proving that, despite the distribution of their execution, these algorithms maintain their correctness (Section 4.2.3), and by empirically demonstrating that, as a result of concurrent execution, our D△PPC algorithm achieves high levels of speedup over its centralized counterparts (Section 4.3).

4.1 Centralized Partial Path Consistency Revisited

The DPC (Algorithm 3 on page 104) and P^3C (Algorithm 4 on page 105) algorithms both take a variable-elimination ordering and already triangulated STN as input. However, if our aim is to decentralize algorithm execution, requiring an already triangulated network and complete variable elimination order punts on providing a distributed solution to a critical algorithmic requirement at best, or requires a centralized representation of the entire network at worst, thus invalidating many of the motivations for distribution in the first place. Hence, the point of this section is to demonstrate that both the graph triangulation process and the elimination order construction process can be incorporated into the DPC algorithm, whose execution we later distribute, with no added computational overhead.

Observe that both the triangulation (Algorithm 2 on page 103) and DPC algorithms end up traversing graphs in exactly the same order, and so their processing can be combined. Our △DPC algorithm (Algorithm 5) is the result of modifying the DPC algorithm based on two insights: (i) that △DPC can construct the variable elimination order *during* execution by applying the SelectNextTimepoint procedure (line 3), which heuristically chooses the next timepoint, v_k, to eliminate; and (ii) as △DPC considers the implications of each

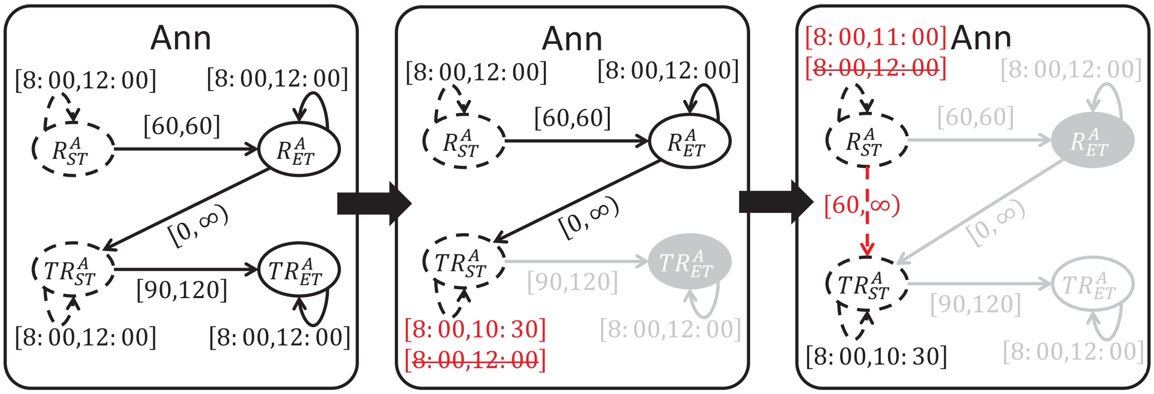
Algorithm 5: Triangulating Directional Path Consistency (△DPC)

```
Input: An STN $\mathcal{G} = \langle V, E \rangle$
Output: A triangulated, DPC STN $\mathcal{G}$ and corresponding elimination order
        $o = (v_1, v_2, \ldots, v_{n-1}, v_n)$, or INCONSISTENT
1  $o \leftarrow ()$
2  while $|V| > 0$ do
3      $v_k \leftarrow$ SelectNextTimepoint($\langle V, E \rangle, o$)
4      $V \leftarrow V \setminus \{v_k\}$
5      $o$.APPEND($v_k$)
6      forall $v_i, v_j \in V$ s.t. $e_{ik}, e_{jk} \in E$ do
7          $E \leftarrow E \cup \{e_{ij}\}$
8          $w_{ij} \leftarrow \min(w_{ij}, w_{ik} + w_{kj})$
9          $w_{ji} \leftarrow \min(w_{ji}, w_{jk} + w_{ki})$
10         if $w_{ij} + w_{ji} < 0$ then
11             return INCONSISTENT
12 return $\mathcal{G}, o$
```

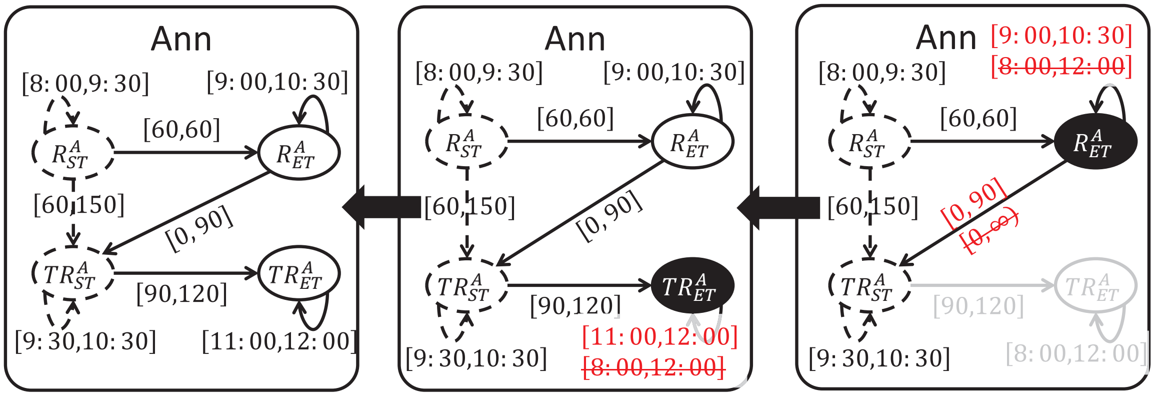
pair of temporal difference constraints involving the removed timepoint variable, it necessarily considers the exact fill edges that the triangulation process would have added. △DPC requires only the original distance graph representation of the STN, $\mathcal{G}$ as input. Line 7 adds, if necessary, any newly-created fill edges between v_k's non-eliminated neighbors and then proceeds to propagate the implications of the eliminated timepoint's constraints forward in lines 8–9. Like the DPC algorithm, △DPC halts as soon as it detects an inconsistency (line 11). Incorporating the triangulation process into the △DPC algorithm reduces the problem of distributing both the DPC and graph triangulation algorithms to that of distributing the execution of the △DPC algorithm alone. The △DPC algorithm is an example of a bucket-elimination algorithm and has the property that the elimination process could stop at any point, and any solution to the remaining subproblem is guaranteed to be extensible to a full solution involving the eliminated timepoints. Thus, a solution can be derived from a DPC network by assigning timepoints in reverse elimination order.

As an example, consider Ann's agent, which we refer to as agent A, executing the △DPC algorithm on Ann's subproblem (see Figure 8a). The algorithm starts by using the minimum fill heuristic to select variable TR^A_{ET} for elimination and adds TR^A_{ET} to its elimination order. Next agent A loops through each pair of TR^A_{ET}'s neighbors. In this case there is only one such pair of neighboring edges: the unary edge, which is represented algorithmically as an edge shared with the reference timepoint z, and the edge with TR^A_{ST}. Together, these edges imply a tighter unary constraint on TR^A_{ST}, lowering the upper bound to 10:30 so that it is guaranteed to occur at least 90 minutes before the eliminated TR^A_{ET}. By eliminating values from TR^A_{ST}'s domain that are inconsistent with TR^A_{ET}, agent A is guaranteed that if a solution to the remaining (non-eliminated) subproblem exists, this solution is extensible to include TR^A_{ET}.

Next agent A selects R^A_{ET} for elimination. In this case notice that R^A_{ET} is involved in a path from R^A_{ST} to TR^A_{ST}. So in addition to updating the domains of neighboring timepoints,



(a) The forward sweep of △PPC (△DPC) works by selecting each timepoint, and before eliminating it from further consideration, calculating the constraints implied over its remaining neighbors.



(b) The reverse sweep of △PPC works by reinstating each timepoint in reverse order, tightening its incident edges with respect to the newly-updated, explicit constraints of its neighbors.

Figure 8: Agent A executing the two-part △PPC on Ann's private subproblem. This can be viewed as the first (a) and last (b) phases of our overall D△PPC algorithm.

agent A must also capture the path from R^A_{ST} to TR^A_{ST} involving R^A_{ET}. To guarantee that the integrity of this path is retained, agent A adds a fill edge from R^A_{ST} to TR^A_{ST}, with a lower bound of 60 and infinite upper bound (as implied by the path). This addition of fill edges is the reason that the output of △DPC is a triangulated network. Note that typically the minimum fill heuristic would select variables that add no fill edges before selecting ones that do, but as we explain in Section 4.2.1, to improve concurrency, we restrict an agent to eliminating its private timepoints prior to its shared timepoints.

When △DPC completes, any remaining, non-eliminated timepoints are involved in external constraints, and will be addressed by the distributed algorithms described shortly in Section 4.2. Before turning to the distributed algorithms, however, we first describe △PPC, which each agent will apply to complete the computation of PPC on its private subproblem in Phase 4. The △PPC algorithm, included as Algorithm 6, nearly identically follows the original P^{3}C algorithm, only replacing DPC with the △DPC algorithm, dropping the triangulation and elimination order requirements, and adding line 5 to explicitly

Algorithm 6: Triangulating Partial Path Consistency (△PPC)

```
Input: An STN $\mathcal{G} = \langle V, E \rangle$; or DPC STN $\mathcal{G}$ w/ corresponding elimination order $o$
Output: A triangulated, PPC STN $\mathcal{G}$ or INCONSISTENT
if $o$ is not provided then
    $\mathcal{G}, o \leftarrow \triangle DPC(\mathcal{G})$
    return INCONSISTENT if △DPC does
for $k = n \ldots 1$ do
    $V \leftarrow V \cup v_k$
    forall $i, j > k$ s.t. $e_{ik}, e_{jk} \in E$ do
        $w_{ik} \leftarrow \min(w_{ik}, w_{ij} + w_{jk})$
        $w_{kj} \leftarrow \min(w_{kj}, w_{ki} + w_{ij})$
return $\mathcal{G}$
```

reinstate each eliminated timepoint during the reverse sweep. The △PPC algorithm complements the △DPC algorithm by reinstating eliminated timepoints in reverse elimination order. However as a timepoint is reinstated, its incident edges are updated with respect to its previously reinstated neighbors, whose now explicit edges are inductively guaranteed to be minimal as a property of the △DPC algorithm.

Figure 8b shows Ann's subproblem just as agent A has returned from establishing PPC over its shared timepoints, as soon described in Section 4.2. To convey the reverse reinstatement order, note that Figure 8b works from right to left. Agent A starts by selecting R^A_{ET}, the last private timepoint that it eliminated, for reinstatement. Then agent A loops through each pair of R^A_{ET}'s neighbors, this time updating each incident edge with respect to the now explicit and minimal third edge (in Figure 8b, there are three incident edges: the domain of each of R^A_{ET}'s neighbors and the edge between them). This results in an updated domain of [9:00,10:30] for R^A_{ET}, which occurs exactly 60 minutes after R^A_{ST}, and a new upper bound on the edge shared with TR^A_{ST}. TR^A_{ET} is similarly reinstated, completing the execution of our algorithm. Next, we prove that our △PPC algorithm is correct and that it runs in $\mathcal{O}\left(n \cdot (\alpha_{\mathcal{G}} + \omega_o^{*2})\right)$ time in Theorems 3 and 4 respectively.

Theorem 3. *The △DPC and △PPC algorithms establish DPC and PPC on an STN, respectively.*

Proof. The difference between Algorithm 6 and the P^{3}C algorithm (Algorithm 4, page 105), is the call to △DPC to obtain an elimination order, to triangulate, and to establish DPC. The `while` loop of Algorithm 5 (line 2), along with lines 3–5, establish a total order, o, of all vertices. Given o, lines 2, 6, and 7 exactly execute the triangulate algorithm (Algorithm 2, page 103), which triangulates the graph so that lines 2, 6, 8, and 11 can exactly execute the DPC algorithm (Algorithm 3, page 104). Thus the △DPC algorithm establishes DPC. The remainder of the algorithm exactly follows the P^{3}C algorithm, which Planken et al. (2008) prove correctly establishes PPC. □

Theorem 4. *$\triangle$PPC executes in $\mathcal{O}\left(n \cdot (\alpha_{\mathcal{G}} + \omega_o^{*2})\right)$ time, where $\alpha_{\mathcal{G}}$ is the complexity of the variable selection heuristic (as applied to $\mathcal{G}$) and ω_o^* is the graph width induced by o.*

Proof. Besides the call to $\triangle$DPC in the first line, Algorithm 6 exactly executes P^{3}C on the STN $\mathcal{G}$, which requires $\mathcal{O}\left(n \cdot \omega_o^{*2}\right)$ time, as proven by Planken et al. (2008). Meanwhile, the outer `while` loop of Algorithm 5 (line 2) is executed n times. For each iteration, all operations require constant time other than the SelectNextTimepoint heuristic (line 3), whose cost $\alpha_{\mathcal{G}}$ is a function of the size and complexity of $\mathcal{G}$, and the inner `for` loop (line 6), which has complexity ω_o^{*2}. ☐

Note that because an elimination order is not provided as input, the costs of the variable selection heuristic become embedded into the algorithm. These costs can range from constant time (if selection is arbitrary) to NP-hard (if selection is optimal), but are typically polynomial in the number of vertices n and number of constraints m (Kjaerulff, 1990). Thus, the algorithm internalizes a computational cost that is typically assumed to be part of the preprocessing. So that our analyses are consistent with convention, and to better capture only the computation costs associated with directly manipulating the underlying temporal network, we will not include these $\alpha_{\mathcal{G}}$ costs in our remaining analyses.

4.2 The Distributed Partial Path Consistency Algorithm

Agents execute the $\triangle$DPC and $\triangle$PPC algorithms separately on their private STNs (Phases 1 and 4, Figure 7), but correctly solving the shared STN (Phases 2 and 3, Figure 7) requires cooperation. To accomplish this, we introduce our distributed partial path consistency algorithm D$\triangle$PPC. Our presentation parallels that of the previous section, where we begin with the forward-sweeping D$\triangle$DPC algorithm for triangulating and establishing DPC on the MaSTN instance in Section 4.2.1, and follow this with the reverse sweep in the D$\triangle$PPC algorithm in Section 4.2.2.

4.2.1 The D$\triangle$DPC Algorithm

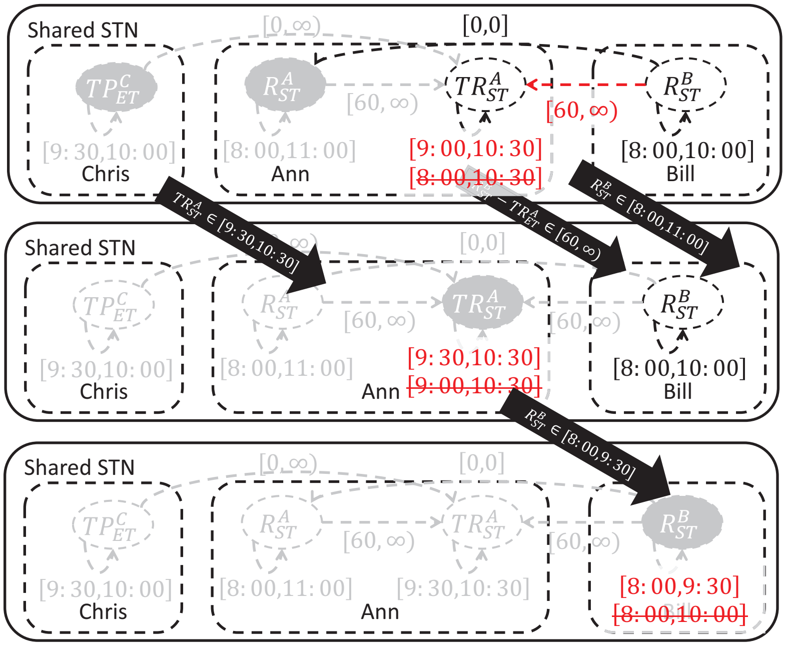
Consider once again the example in Figure 8a. Agent A can successfully and independently execute $\triangle$DPC on its two private timepoints TR_{ET}^A and R_{ET}^A. At this point, consider what would happen if agent A proceeded with eliminating its other timepoints R_{ET}^A and TR_{ST}^A. Each remaining timepoint is connected to portions of the MaSTN that belong to other agents, and so agent A must now consider how its actions will impact other agents and *vice versa*. For example, suppose agent A is considering eliminating R_{ST}^A, but unbeknownst to agent A, Bill's agent (agent B) has already eliminated R_{ST}^B. In this case, agent A would eliminate R_{ST}^A assuming that an edge with R_{ST}^B still exists, when in reality, it does not. As a result, the computation of agent A could result in superfluous reasoning or reasoning over stale information, which ultimately could jeopardize the integrity of the output of the algorithm as a whole. Next, we discuss how our algorithm avoids these problematic situations.

Our distributed algorithm D$\triangle$DPC (Algorithm 7) is a novel, distributed implementation of a bucket-elimination algorithm for establishing DPC in multiagent temporal networks (Phases 1 and 2, Figure 7). Each agent starts by completing Phase 1 by applying $\triangle$DPC on its private STP (lines 1–2), as presented in the previous section and illustrated in Figure 8a.

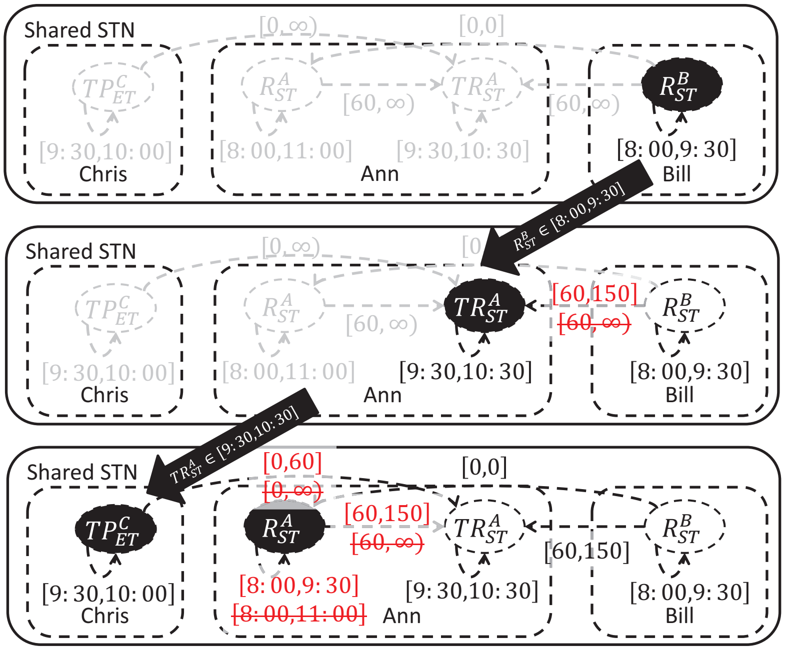
```
Algorithm 7: Distributed Directional Path Consistency (D△DPC)
Input: Agent i's portion of a distance graph $\mathcal{G}^i = \langle V^i, E^i \rangle$
Output: Agent i's portion of a triangulated, DPC distance graph $\mathcal{G}^i$ and
        corresponding elimination orders $o_P^i = \left(v_1, \ldots v_{n_P^i}\right)$ and $o_S = (v_1, \ldots v_{n_S})$, or
        INCONSISTENT
1  $\mathcal{G}^i, o_P^i \leftarrow \triangle DPC(\langle V_P^i, E^i \rangle)$
2  return INCONSISTENT if △DPC does
3  $o_S \leftarrow ()$
4  while $|V_I^i| > 0$ do
5      RequestLock($o_S$)
6      $v_k \leftarrow$ SelectNextTimepoint($\langle V_I^i, E^i \rangle, o_S$)
7      $o_S$.append($v_k$)
8      ReleaseLock($o_S$)
9      $V_I^i \leftarrow V_I^i \setminus \{v_k\}$
10     forall $v_i \in V_X^i \cap o_S$ s.t. $e_{ik} \in E$ and updates for $v_i$ have not yet been received do
11         $E^i \leftarrow E^i \cup$ BlockReceiveUpdatedEdges(Agent($v_i$))
12     forall $v_i, v_j \in V_S^i$ s.t. $e_{ik}, e_{jk} \in E$ do
13         $E \leftarrow E \cup \{e_{ij}\}$
14         $w_{ij} \leftarrow \min(w_{ij}, w_{ik} + w_{kj})$
15         $w_{ji} \leftarrow \min(w_{ji}, w_{jk} + w_{ki})$
16         if $w_{ij} + w_{ji} < 0$ then
17             Broadcast(INCONSISTENT)
18             return INCONSISTENT
19         else
20             SendUpdatedEdge($e_{ij}$, Agent($v_i$))
21             SendUpdatedEdge($e_{ij}$, Agent($v_j$))
22 return $\mathcal{G}^i, o_P^i, o_S^i$
```

The output of △DPC for an agent's private problem (shown in Figure 9a) can be viewed as a summary of how its private subproblem impacts the shared problem. In other words, any schedules that are consistent with the remaining shared portion of an agent's subproblem are guaranteed to be extensible to a solution for that agent's subproblem. As a result, rather than jointly reasoning over the entire MaSTN, agents only need to cooperatively reason over the potentially much smaller shared STN.

Continuing on to Phase 2, each agent is still responsible for eliminating each of its own timepoints in the shared STN, and does so by securing a lock on the shared timepoint elimination ordering (line 5), selecting one of its interface timepoints v_k to eliminate (line 6), recording v_k in the shared elimination ordering (line 7), and releasing the lock (line 8). This is done on a first-come, first-served basis. To avoid deadlocks and establish a precise ordering over all timepoints, if two or more agents request the lock at the same time, the lock simply goes to the agent with the smallest id. Then, before performing the basic △DPC inner loop for this selected timepoint, the agent blocks until it has received updated edge



(a) The D△DPC portion of the D△PPC algorithm uses blocking communication to ensure that timepoints are eliminated and the network is triangulated in a globally consistent manner.



(b) As agents reinstate timepoints during the reverse sweep of the D△PPC algorithm, each agent responds to each message it processed during the previous phase with updated edge information for that edge.

Figure 9: The second and third phases of our D△PPC algorithm require coordination (as represented by block arrows) between agents to establish PPC on the shared STN.

information with respect to all timepoints that share an edge with v_k but appear before it in the shared elimination ordering (line 11). This involves adding the edge, if necessary, and then updating its weights to the minimum of its existing and newly received weights. Once these steps are completed, an agent can safely proceed with lines 12–21, which are identical to the inner loop of the △DPC algorithm (Algorithm 5) except for lines 20–21, which send updated edge information to each neighboring agent. This continues until an agent has eliminated all of its interface timepoints.

Returning to our running example, Figure 8a represents the first step of the D△DPC algorithm. Note, not all agents need to complete eliminating their private timepoints before an agent can proceed to eliminating its shared timepoints. For example, in Figure 9b, agents A and C both begin eliminating before agent B does. This can be due to heterogeneity in either agents' capabilities or in their subproblems. In this case, agent C appends its timepoint TP^C_{ET} to the shared elimination order before agent A appends R^A_{ST}. However, because there are no shared edges between TP^C_{ET} and R^A_{ST}, both agents can continue concurrently (without blocking in line 11). Note that both agents A and C compute a new domain for TR^A_{ET}; however, agent A records only its update of [9:00,10:30] locally. It is not until agent A selects to eliminate TR^A_{ET}, and thus must block to receive updates due to TP^C_{ET}, that agent A records agent C's tighter update of [9:30,10:30]. The elimination of agent A's remaining timepoints also leads to messages to agent B. However agent B only receives these updates once it eliminates R^B_{ST} in the final step. Note that, if agent B had attempted to eliminate R^B_{ST} before agent A had completed computing these new edges for B, then agent B would have been forced to block until it had received the edge updates regarding R^A_{ST}, which would appear before R^B_{ST} in the shared elimination order.

4.2.2 The D△PPC Algorithm

The D△PPC algorithm (Algorithm 8) invokes the preceding algorithms as it accomplishes all four phases depicted in Figure 7. It starts by invoking the D△DPC algorithm, which results in a globally DPC and triangulated network (or detection of inconsistency), along with an elimination order over vertices (lines 1–2). The new contribution of the D△PPC algorithm then is in agents cooperatively performing the reverse sweep to establish PPC over the shared STN (Phase 3 in Figure 7), before invoking the △PPC algorithm to finish establishing PPC on the private STN (Phase 4).

To cooperatively establish PPC on the shared STN, the D△PPC algorithm traverses vertices in reverse order and, instead of calculating or updating a third edge based on its two neighboring edges as D△DPC does, it updates the neighboring edges based on the (newly) updated third edge. Thus, to guarantee that these updates are correctly shared with and correctly incorporate information from other agents, if that third edge is external to the agent, it must wait until it receives updated edge weights from the agent responsible for updating it last in line 8. For an edge e_{ij} where $i < j$, agent i will have been the last agent to update this edge, since its timepoint appears earliest in the elimination order. The incident edge weights are then updated with respect to this updated third edge (lines 9–12). After performing all updates on an edge, the agent then communicates the updated weights to any agent that also shares the edge (line 14). This will be any agent that sent

Algorithm 8: Distributed Partial Path Consistency (D△PPC)

```
Input: agent $i$'s local STP instance $\mathcal{G}^i = \langle V^i, E^i \rangle$
Output: agent $i$'s portion of the PPC network $\mathcal{G}^i$ or INCONSISTENT
1  $\mathcal{G}^i, o_P^i = \left(v_1, \ldots v_{n_P^i}\right), o_S = (v_1, \ldots v_{n_S}) \leftarrow$ D△DPC ($\mathcal{G}^i$)
2  return INCONSISTENT if D△DPC does
3  for $k = n_S \ldots 1$ s.t. $v_k \in V_I^i$ do
4      $V \leftarrow V \cup v_k$
5      for $i = n_S \ldots k+1$ s.t. $e_{ik} \in E^i$ do
6          for $j = n_S \ldots i+1$ s.t. $e_{jk} \in E^i$ do
7              if $e_{ij} \in E_X^i$ and $w_{ij}, w_{ji}$ have not yet been updated then
8                  $w_{ij}, w_{ji} \leftarrow$ BlockReceiveUpdatedEdge($Agent(v_i)$)
9              $w_{ik} \leftarrow \min(w_{ik}, w_{ij} + w_{jk})$
10             $w_{ki} \leftarrow \min(w_{ki}, w_{kj} + w_{ji})$
11             $w_{kj} \leftarrow \min(w_{kj}, w_{ki} + w_{ij})$
12             $w_{jk} \leftarrow \min(w_{jk}, w_{ji} + w_{ik})$
13         for $j = 1 \ldots k-1$ s.t. $e_{ij}, e_{jk} \in E^i \wedge v_j \in V_X^i$ do
14             SendUpdatedEdge($Agent(v_j), e_{ik}$)
15 △PPC ($\langle V_P^i, E^i \rangle, o_P^i$)
16 return $\mathcal{G}^i$
```

an update regarding this edge during D△DPC. Note that the mechanisms that are in place for guaranteeing correct communication only need to be executed for external edges.

Figure 9b shows this process in action. The first timepoint to be reinstated was also the last to be eliminated: R_{ST}^B. As a property of DPC graphs, R_{ST}^B's domain of [8:00-9:30] is guaranteed to be minimal, that is, represent the exact set of values that could be part of some joint solution. The D△PPC algorithm, then, propagates the information captured by the tighter edges and domains of variables that appear later in the elimination order back through the MaSTN to the variables that appear earlier. Agent B kicks off the algorithm by skipping the inner-most `for` loop (since it has only one neighbor that appears after it: the *de facto* z reference point), and sending its updated domain to agent A. Notice that even though agent A might need R_{ST}^B's updated domain to update both of its shared timepoints, the message is only received once. Then, agent A reinstates Ann's variables, one by one, and uses this information to update the upper bounds on the edges from R_{ST}^A and R_{ST}^B to T_{ST}^A, and also the domain of R_{ST}^A. After all shared timepoints have been reinstated, the algorithm terminates after each agent finishes propagating the shared updates through its private network (as illustrated in Figure 8b). Notice that in line 15, agent i supplies the previously computed o_P to △PPC, ensuring that △DPC will be skipped and that timepoints will be reinstated in reverse o_P order.

Overall, our D△PPC algorithm introduces three points of synchrony during execution: contention over the shared elimination order and the two calls to BlockReceiveUpdatedEdge. To simplify the understanding and analysis of our algorithms, we have made these synchronization points explicit. A previous formulation of these algorithms (Boerkoel & Durfee, 2010) allows agents to proceed optimistically beyond these synchronization points. For example, an agent could optimistically begin eliminating one of its timepoints before officially obtaining the shared elimination order lock, as long as it reevaluates whether any other agents had concurrently eliminated a neighboring timepoint once the lock is obtained. Likewise, during the reverse-sweep, an agent could begin updating its private edges with the optimistic assumption that its current view of the network is up-to-date, backjumping its computation when a new edge update arrives that does not fit this assumption. In both cases, proceeding optimistically never leads to incorrect computation, and while computation may be wasted if invalidating updates later arrive, the agents would have sat idle anyway waiting for those updates. Hence, at worst nothing is lost, and at best the optimistic assumptions hold and agents have gotten a head start on further processing.

4.2.3 THEORETICAL ANALYSIS

In this section, we prove that our D△PPC algorithm is correct by showing both that it is deadlock free (Theorem 5) and establishes PPC (Theorem 6), and also prove that the added communication does not change the underlying runtime complexity (Theorem 7).

Theorem 5. *D△PPC is deadlock free.*

Proof. We start by proving first that the call to D△DPC is deadlock free. There are two lines in D△DPC (Algorithm 7) where agents may block on other agents: line 5 and line 11. In line 5, there is only one lock (on the shared elimination order), and requests for the lock are granted on a first-come, first-served basis, with ties being broken according to agent id. Further, once an agent has a lock on the elimination order, o_S, it executes two local operations that select a variable to append to o_S before releasing the lock again in line 8. SELECTNEXTTIMEPOINT is an independent, local decision requiring only locally known information. Hence, a deadlock cannot occur as a result of contention over o_S.

This leaves line 11. Assume, by way of contradiction, that line 11 causes a deadlock. This implies that there are two (or more) agents, i and j, where $i \neq j$ such that each agent is simultaneously waiting for communication from the other in line 11. Thus, there exists a timepoint $v_x^j \in V_X^i \cap V_L^j$ for which agent i is waiting to receive updated edges from agent j, while there is also a $v_y^i \in V_X^j \cap V_L^i$ for which agent j is waiting to receive updated edges from agent i. Notice that v_y^j must appear before v_x^i in agent i's copy of o_S because, otherwise, by the time v_y^j appeared in o_S, agent i would have already sent agent j all edge updates pertaining to v_x^i (lines 20–21) in the previous loop iteration in which v_x^i was eliminated (and added to o_S in line 7). However, for the same reason, v_x^i must appear before v_y^j in agent j's copy of o_S. But this is a contradiction, because there is only one shared elimination order and agents can only append to it after being granted mutually-exclusive access. This argument extends inductively to three or more agents, and so line 11 can also not be the cause of a deadlock, which presents a contradiction.

Therefore the D△DPC algorithm is deadlock free. Because after its call to D△DPC, the D△PPC algorithm traverses the MaSTN in the exact opposite order of the D△DPC algorithm, the proof that the remainder of the D△PPC algorithm is deadlock free follows, *mutandis mutatis.* □

Theorem 6. *D△PPC correctly establishes PPC on the MaSTN.*

Proof (Sketch).[2] The algorithm starts by correctly establishing DPC on the MaSTN. Since, by definition, none of an agent's private timepoints share any edges with private timepoints of any other agent, each agent can apply △DPC to its private subproblem independently (lines 1–2). Given the nature of the △DPC algorithm as a bucket-elimination algorithm (solutions to the remaining subproblem are guaranteed extensible to the eliminated variables), each agent will have computed constraints over its interface variables that capture the impact its private subproblem has on the shared subproblem. The remaining algorithm applies the △DPC algorithm on an agent's shared timepoints. Lines 5–8 guarantee that a globally consistent elimination ordering of all shared timepoints is established. Finally, lines 11 and 20–21 guarantee that the edge weights used by an agent are not stale and are consistent among all agents involved.

Then, the algorithm executes the same operations as the second phase of the △PPC algorithm, but in a distributed fashion, using blocking communication to guarantee that all computation is performed using only the correctly updated edge weights. □

Theorem 7. *D△PPC executes in $\mathcal{O}\left((n_P + n_S) \cdot \omega_o^{*2}\right)$ time, where $n_P = \max_i |V_P^i|, n_S = |V_S|$, and ω_o^* is the graph width induced by o.*

Proof. Beyond the lines added for communication, the D△PPC algorithm exactly executes △PPC with the caveat that the elimination order is restricted to eliminating all private timepoints prior to all shared timepoints. The lines of code added for communication only increase work by a constant factor within the △PPC algorithm (for each edge update that is performed by △PPC, at most one message is sent or received). However, when agents must block on edge updates from other agents, an agent may need to wait for some other agent to complete some local computation. In the worst case, the elimination and subsequent revisiting of all shared timepoints must be done completely sequentially, which puts the algorithm in the same complexity class as the △PPC algorithm, $\mathcal{O}\left((n_P + n_S) \cdot \omega_o^{*2}\right)$. □

Note Theorem 7 provides a worst-case analysis. In the best case, complete concurrency is possible, putting the runtime closer to $\mathcal{O}\left(n_L \cdot \omega_o^{*2}\right)$, where $n_L = |V_L|$ is less than or equal to $n_P + n_S$. That is, in the best case, no blocking occurs (and agents can execute 100% concurrently), leading to only the costs of agents executing △DPC on their subproblems. Note that this best case is likely only realized when, for instance, there are no external constraints. In our empirical evaluations, which we present next, we find that run times fall somewhere between these two extremes in practice and closer to the best case for loosely-coupled problems.

4.3 Empirical Evaluation

In this section, we empirically evaluate our distributed algorithm for solving the MaSTP. The performance of our distributed algorithm relies on the size of each agent's private subproblem relative to the size and complexity of the collective shared subproblem. The greater the portion of the problem that is private to an agent, rather than shared, the

---

[2] The full version of this, and all remaining proof sketches, are available in the online appendix associated with this publication.

greater the level of independent reasoning and concurrency, and thus the faster the overall solve time of our distributed algorithm compared to a centralized algorithm.

4.3.1 Experimental Setup

Random Problem Generator. We evaluate our algorithms for solving MaSTPs using the random problem generator described by Hunsberger (2002), which we adapt so that it generates *multiagent* STP instances. Each problem instance has A agents that each have start timepoints and end timepoints for 10 activities. Each activity is constrained to occur within the time interval [0,600] relative to a global zero reference timepoint, z. Each activity's duration is constrained by a lower bound, lb, chosen uniformly from interval [0,60] and an upper bound chosen uniformly from the interval $[lb, lb + 60]$. In addition to these constraints, the generator adds 50 additional local constraints for each agent and X total external constraints. Each of these additional constraints, e_{ij}, has a bound that is chosen uniformly from the interval $[w_{ij} - t \cdot (w_{ij} + w_{ji}), w_{ij}]$, where v_i and v_j are chosen, with replacement, from the set of all timepoints with uniform probability, and $t \in [0, 1]$ is a tightness parameter that dictates the maximum portion that an interval can be tightened, and whose default value is set to $t = 1$ in these experiments. This particular problem generator provides us the upside of directly and systematically controlling the number of external constraints relative to the number of (and size/complexity of) agent subproblems.

Factory Scheduling Problem Benchmark. A second source of problems is from a multiagent factory scheduling domain (Boerkoel, Planken, Wilcox, & Shah, 2013). These randomly generated MaSTPs simulate A agents working together to complete T tasks in the construction of a large structural workpiece in a manufacturing environment, using realistic task duration, wait, and deadline constraint settings. This publicly available benchmark (Boerkoel et al., 2013) distinguishes between structural constraints—the existence of which provide the underlying structure of the temporal network (e.g., all task durations are between some minimum and maximum durations and must complete before some makespan), and refinement constraints—those that instantiate constraints' bounds with particular weights (e.g., task-specific deadline and duration constraints). The former encode general task structure and knowledge, while the later encode particular domain knowledge of a factory manager to refine the space of schedules so that only feasible schedules remain. In our experiments, we process all constraints equally, whether a structural constraint is refined with a more particular bound or not. We evaluate our approaches on the problem data available: both as number of agents increases ($A \in \{2, 4, 8, 12, 16, 20\}, T = 20 \cdot A$) and also as the total number of tasks increases ($A = 16, T \in \{80, 160, 240, 320, 400, 480, 560\}$). Unlike our Random Problem Generator, this set of problems is distinctive in that the underlying problem size and (inter)constrainedness of each agent problem is dictated by real-world factory needs, rather than generated to be uniformly distributed across agents.

Experimental Procedure. To capture expected trends for a particular problem source (Random or Factory Scheduling) and parameter setting, we evaluate the average performance of an algorithm over 50 independently-generated trials. We include error bars representing one standard deviation for all of our results. Note, in many cases, the error bars appear smaller than the tick marks used to represent the data points. Our algorithms were programmed in Java, on a 3 GHz processor using 4 GB of RAM. For the purposes of modeling a concurrent, multiagent system, we interrupted each agent after it was given the opportunity to perform a single edge update or evaluation and also a single communication (sending or receiving one message), systematically sharing the processor between all agents involved. All approaches use the minimum fill heuristic (Kjaerulff, 1990). We applied our approaches to connected networks of agents, although intuitively, the performance of any of our algorithms would be enhanced by applying them to disparate agent networks independently. Finally, all problem instances were generated to lead to consistent STP instances to evaluate a full application of each algorithm. While not necessary for our algorithms, excluding inconsistent STP instances avoids overestimating reductions in execution times when an algorithm halts as soon as an inconsistency is found. Moreover, unlike previous approaches, our algorithms do not require input STPs to be triangulated (Xu & Choueiry, 2003; Planken et al., 2008).

Evaluation Metrics. When solving a traditional CSP, one of the primary measures of a unit of computation is the *constraint check*. Meisels and Zivan (2007) extend this metric to a distributed setting by introducing the *non-concurrent constraint check* (*nccc*). Note that agents solving a distributed problem form a partial order over constraint checks based on the fact that (i) any two constraint checks performed within the same agent must be performed sequentially and (ii) any constraint check x_i performed by agent i prior to sending a message m_{ij} can be ordered before any constraint check x_j performed by agent j after receipt of m_{ij}. The *nccc* metric, then, is simply the length of the longest critical path in this partial ordering of constraint checks. We generalized the *nccc* metric to our work by counting the number of non-concurrent edge updates: the number of cycles it takes to establish PPC on the MaSTP, where each agent is given an opportunity to update or check the bounds of a single edge during each cycle of computation, although agents may spend this cycle idly blocking on updates from other agents.

We report non-concurrent edge updates rather than the simulated algorithm runtime due to limitations of simulating a distributed system. First, the heterogeneity of agent capabilities and underlying sources of message latency can vary dramatically across different real distributed computing systems. There is no systematic, compelling way to estimate the runtime of such distributed systems in an assumption-free, unbiased manner, so instead we provide an implementation- and system-independent evaluation. Second, separately maintaining the state of many agents in a simulated distributed system introduces a large amount of overhead due to practical issues such as memory swapping, which unduly inhibits accurate and fair algorithm runtime comparisons. Since D△PPC requires a significant number of messages (which would typically incur latency), we separately count the number of computation cycles where at least one agent sends a message, which as described later, allows for rudimentary runtime projections with latency. We report the total number of messages sent by D△PPC later in Section 5.3.1, where we compare it against our distributed temporal decoupling algorithm.

4.3.2 Empirical Comparison

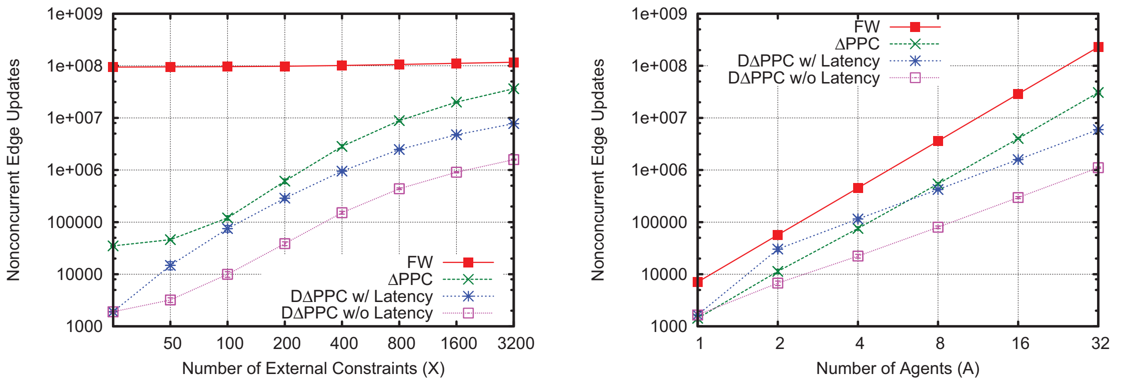
One of the main benefits that we associate with performing more of a computation in a distributed fashion is that it promotes greater concurrency. In this section, we explore how well our distributed algorithms can exploit concurrent computation, reporting the number of non-concurrent edge updates along with the number of computational cycles that require messages. We compare the following approaches:

- **FW** – the Floyd-Warshall algorithm executed on a centralized version of the problem;
- **△PPC** – our △PPC algorithm executed by a single agent on a centralized version of the problem;
- **D△PPC w/ Latency** – our D△PPC algorithm where we assume that every computational cycle that might incur message latency requires an-order-of-magnitude more time than performing a single edge update; and
- **D△PPC w/o Latency**—our D△PPC algorithm with no extra message latency penalties.

Random Problems. Figure 10a compares each algorithm's performance on problems from the Random Problem generator, as the level of coupling, as measured by the number of external constraints increases. As one would expect, as the density of the overall network increases, the △PPC approaches the same costs as the fully-connected FW algorithm. Without latency, our D△PPC algorithm, in expectation, maintains a steady improvement over the its centralized counterpart, ranging from 12 times speedup (centralized computation/distributed computation) when $X = 100$ to a nearly 23 times speedup when $X = 3200$, which is within 12% of perfect speedup for 25 agents.

Interestingly, even when there are no external constraints ($X = 0$), our D△PPC algorithm achieves only an 18-fold, rather than 25-fold, improvement over △PPC. This is due to heterogeneity in the complexity of individual agent problems. When there are no or only very few external constraints, there are few opportunities for agents with easy-to-solve local problems to help other agents out, and thus effectively load-balance. As the number external constraints increases, so does the overall time complexity of both the △PPC and D△PPC w/o Latency approaches; however, opportunities for load-balancing and synchronization points also increase. These results indicate that the increased opportunities for load-balancing outweigh the costs of increased synchronization as the number of external constraints increases.

Coordination does, however, introduce overhead. For the D△PPC w/ Latency curve of Figure 10a, we try to account for message latency by penalizing our D△PPC with the extra computational cost equivalent to 10 edge updates for every message cycle. In practice, this represents only a rough estimate of the costs of communication, since (i) not all messages sent will require an agent to block (i.e., many messages will be received prior to their need) and (ii) message latency could lead to compounding delays elsewhere in the network. Even with a penalty for message latency, our distributed approach maintains an advantage over the centralized approach, albeit a much smaller one (e.g., only a 20% improvement when $X = 100$). As the number of external constraints increases, however, the effects of the additional latency penalty become mitigated since (i) as the network becomes more dense, each new external constraint is increasingly less likely to spur new messages (i.e., an edge would have already been created and communicated), and (ii) the extra message latency costs are amortized across a greater number of agents.



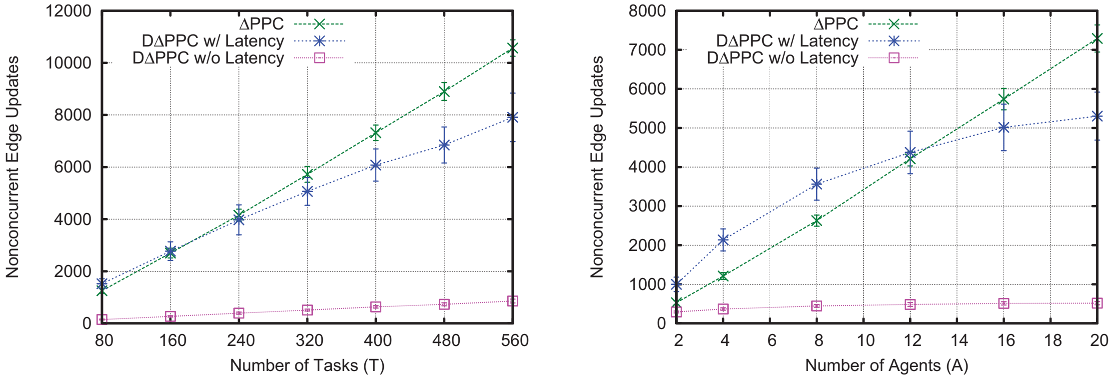
(a) Non-concurrent edge updates as the number of external constraints X increases.

(b) Non-concurrent edge updates as the number of agents A increases.

Figure 10: Empirical comparison of D△PPC on the randomly generated problem set.

Figure 10b shows the number of non-concurrent edge updates as the number of agents grows. The number of non-concurrent edge updates grows linearly with the number of agents across all approaches. Interestingly, here when we estimate the effects of message latency, there is a cross-over point between △PPC and D△PPC w/ Latency. A certain level of message latency can mean that, for a small number of agents, it is faster to centralize computation using △PPC than to utilize parallelism with slow communication, but as the number of agents grows, the centralized problem becomes unwieldy and the distribution of computation makes D△PPC w/ Latency superior once again. Figure 10b also shows that the expected runtime of D△PPC increases more slowly than FW or △PPC, and that D△PPC's speedup increases with the number of agents as seen by the widening relative gap between the △PPC and D△PPC curves. Thus, D△PPC scales with the increasing number of agents better than △PPC.

Factory Scheduling Problems. At a high level, Figure 11 for the Factory Scheduling Problems shows many of the same trends that we saw in Figure 10. There are, however, a few notable differences. First, we do not plot using a log scale, like in Figure 10, since (i) the parameters grow on a linear scale, and (ii) we leave out the much more expensive FW approach so that we can better compare the most similar approaches. Second, the number of tasks and constraints across agents' subproblems are not uniform, like in the Random Problem case, leading to constraint networks that are based on underlying structure of realistic problems. As both the number of tasks and agents increase, our D△PPC clearly scales better than its centralized counter-part, demonstrating a robustness to heterogeneity across agent problems. When the number of tasks is low ($T = 80$), D△PPC exhibits 8.5 times speedup over △PPC which steadily grows to a 12.3 fold speedup as the number tasks grows (recall problems are distributed across 16 agents). Similarly, the relative gap between D△PPC and △PPC grows linearly in the number of agents, settling to within 30% of perfect speedup. Finally, this set of experiments more clearly demonstrates that the costs of high message latency can be overcome as the number or complexity of agents' local problems grows sufficiently high.



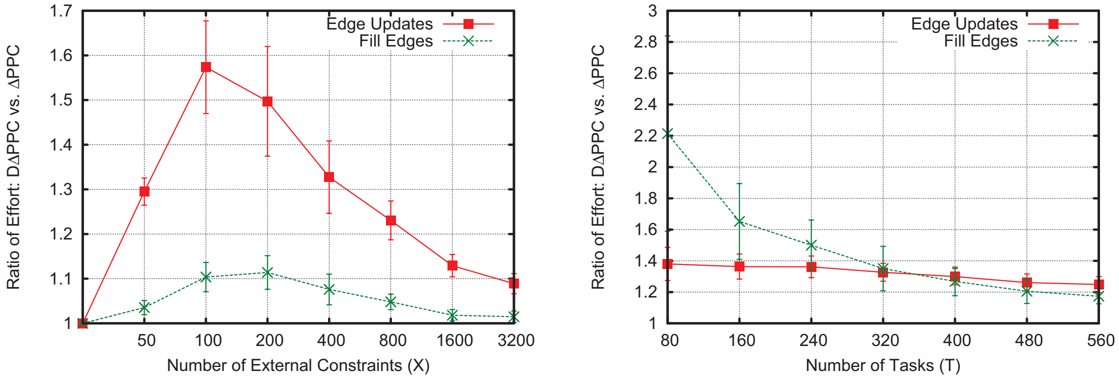
(a) Non-concurrent edge updates as the number tasks T increases.

(b) Non-concurrent edge updates as the number of agents A increases.

Figure 11: Empirical comparison of D△PPC on the factory scheduling problem set.

Total Effort. The minimum-fill variable ordering heuristic myopically selects the timepoint, from a set of timepoints, whose elimination it expects will lead to the fewest added fill edges. Since the centralized algorithm, △PPC, places no restrictions on this heuristic, we expect the heuristic to perform well, adding few fill edges. D△PPC, on the other hand, both restricts private timepoints to be eliminated prior to shared timepoints, and restricts each agent to only eliminating its own timepoints. Intuitively, we expect each of these additional restrictions to hurt heuristic performance, that is, to lead to triangulations with more fill edges. An increase in the overall number of fill edges, in turn, increases the number of edges, and thus overall number of edge updates required to update the network. To evaluate how efficiently effort is distributed across agents, we compare △PPC and D△PPC in the total number of fill edges added and the total number of edge updates (summed across all agents). We compute the ratio of D△PPC *versus* △PPC effort using both these metrics across the two parameters that most directly impact constraint density: number of external constraints X for our randomly generated problems, and the number of tasks T for the factory scheduling problems. Results for both are displayed in Figure 12.

Overall, for both algorithms, the total number of fill edges and edge updates increases as X or T increases. Figure 12a displays that relative to △PPC, the number of fill edges and edge updates computed by D△PPC increases at a faster rate with low X values, and at a slower rate for higher X values, leading to a 57% increase in the total edge updates and 11% increase in total fill edges at its peak. There are multiple trends occurring here. Early on $(X = 0 \ldots 200)$, heterogeneity between individual agents' private subproblems dictates that the elimination of shared timepoints is done by the agent that gets there first, which is not necessarily correlated with the least-connected shared timepoint. As the number external constraints increases, however, an agent's ability to load balance the elimination of shared timepoints improves—when more agents are spending more time competing for the shared elimination order lock, the one with the least amount of (shared) computation (which is correlated with fewest added shared fill edges) will get it. This improved load-balancing is also accentuated by the fact that as X increases, so does the shared network density,



(a) Relative (to centralized) number of added fill edges as the number of external constraints X increases for the randomly generated problem set.

(b) Relative (to centralized) number of added fill edges as the number of tasks T increases for the factory scheduling set.

Figure 12: Empirical comparison of D△PPC in terms of total effort.

increasing the number of fill edges that must be inevitably added by both approaches, and thus dampening the relative advantage that △PPC has over D△PPC.

The results displayed in Figure 12a show the trends as the density of external constraints increases on the random problems. Figure 12b, on the other hand, shows that when the overall number of constraints is increased more generally, the relative total effort monotonically decreases on the factory scheduling problems. This validates our previous study that showed that eliminating private timepoints before shared timepoints actually improves the performance of variable elimination heuristics by taking advantage of structural knowledge to avoid increasing the connectedness between agents' problems (Boerkoel & Durfee, 2009). Thus, as the complexity of local problems grows relative to the complexity of the shared problem, as is the case for these well-structured, loosely coupled factory scheduling problems, the distributed application of the variable ordering heuristic (as in D△PPC) improves performance over its centralized application (as in △PPC). Interestingly, in the factory scheduling problem set, the variability in the number of shared timepoints per agent initially causes the relative margin of extra fill edges to be much larger than in our randomly generated problems, settling after a sufficient number of tasks are added.

4.3.3 Summary

In this section, we presented the D△PPC algorithm, our distributed version of the P^3C algorithm, which exchanges local constraint summaries so that agents can establish minimal, partially path consistent MaSTNs. The D△PPC algorithm utilizes our D△DPC algorithm in which an agent independently eliminates its private timepoint variables before coordinating to eliminate shared variables, which has the effect of exactly summarizing the impact its subproblem has on other agents' problems. The remainder of the D△PPC algorithm is essentially a reverse sweep of the D△DPC algorithm, where care is taken to ensure that an agent synchronizes its edge weights with those of other agents prior to updating its local edges. We prove that our algorithm correctly calculates the joint solution space of an MaSTP without unnecessarily revealing private variables. While D$\triangle$PPC has the same worst-case complexity as its centralized counterpart, we show empirically that it exploits concurrency to outperform centralized approaches in practice. It scales much more slowly with the number of agents than centralized approaches and achieves within 12% of perfect speedup as the relative size and complexity of agents' local problems grow in the problems we studied.

5. A Distributed Algorithm for Calculating Temporal Decouplings

Our D$\triangle$PPC algorithm outputs the space of all possible joint solutions for an MaSTP, as depicted in Phase 4 of Figure 7. However, as discussed in Section 1, a potential limitation of this approach is that dependencies between agents' problems are maintained. So agents must coordinate decisions using this full representation to avoid giving inconsistent advice, such as if Ann's agent advises Ann to start recreation at 8:00 while Bill's agent advises him to wait until 9:00. Thus, any and all advice by one agent that could potentially lead to an update (e.g., Ann's decision to start recreation at 8:00) would need to be propagated before some other agent offers advice that can be guaranteed consistent. This presents a challenge if updates are frequent while communication is expensive (e.g., slow), uncertain (e.g., intermittent), or otherwise problematic. Agents must coordinate to re-establish partial path consistency either by re-executing our D$\triangle$PPC algorithm or by employing more recent *incremental* versions of our algorithm that recognize that updates might only propagate through a small portion of the MaSTN (Boerkoel et al., 2013). A temporal decoupling, on the other hand, gives each agent the ability to reason over its user's local schedule in a completely independent manner. This allows agents to can give unilateral, responsive, and sound, though generally incomplete, advice.

The goal of the Multiagent Temporal Decoupling (MaTD) algorithm is to find a set of decoupling constraints C_Δ that render the MaSTP's external constraints C_X moot, and thus agents' subproblems independent (see Section 3.2). This goal can be achieved by assigning values to all of the variables on the reverse sweep (as in Figure 1b), but doing so sacrifices all of the flexibility that is provided by maintaining solution spaces. Our insight is that assigning private timepoints does not matter to independence, and so we can assign just the shared variables, and then only tighten private variables in response to these assignments (Section 5.1). We show that, in many cases, assignments to shared variables can be relaxed to further increase flexibility (Section 5.2). Overall, our algorithms compute a temporal decoupling while making heuristic choices to reduce flexibility sacrificed. Later in Section 5.3, we describe ways of measuring flexibility, and empirically evaluate how different algorithmic choices affect flexibility as well as runtime.

As shown in Figure 13, Phases 1 and 2 of our algorithm, which establish DPC on the MaSTP, are the same as the D$\triangle$PPC algorithm (shown in Figure 7). In the new Phase 3, however, agents establish a temporal decoupling by assigning their shared timepoints in the reverse order that agents eliminated them in Phase 2. Next, in the optional Phase 4, agents revisit each shared timepoint in original elimination order and relax all decoupling constraints to the maximum extent possible, resulting in a minimal decoupling. Phase 5, in which each agent independently establishes PPC on its local problem, is again the same as the last phase of D$\triangle$PPC algorithm.

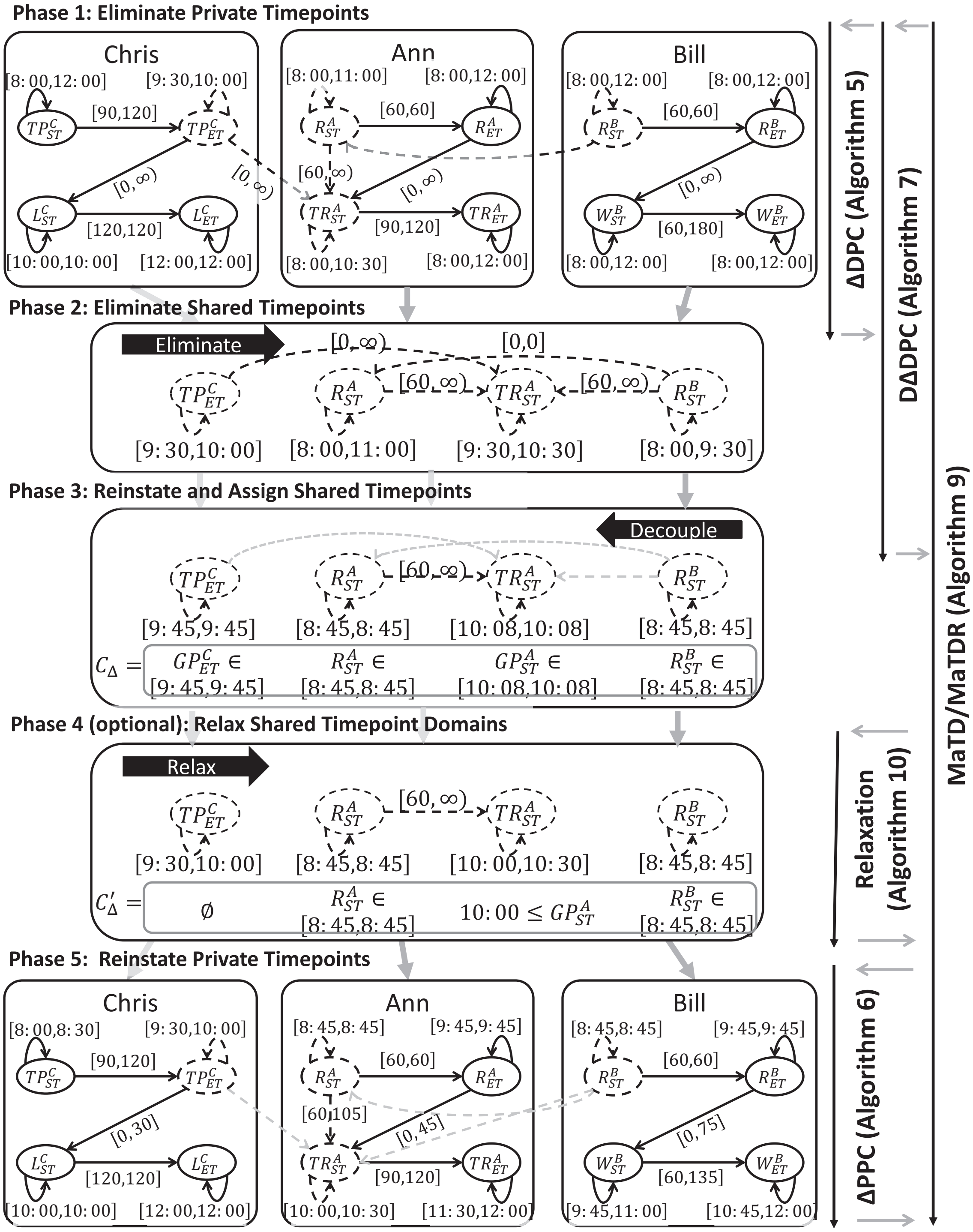


Figure 13: Our MaTD algorithm replaces the shared D△PPC phase of the D△PPC algorithm (depicted in Figure 7) with a decoupling phase that assigns shared timepoints in reverse elimination order (Phase 3) and an optional Relaxation phase, in which timepoints are revisited and flexibility is recovered to the extent possible.

5.1 A Distributed Multiagent Temporal Decoupling Algorithm

The MaTD algorithm, presented as Algorithm 9, starts (lines 1-2) by executing our D△DPC algorithm (Algorithm 7 on page 119) to triangulate and propagate the constraints, eliminating the shared timepoints V_S last. If D△DPC propagates to an inconsistent graph, the algorithm returns INCONSISTENT; else, as illustrated in the first two phases of Figure 13, the resulting MaSTN captures the implications of agents' private constraints as reflected in the tighter domains of the shared timepoints. Next, MaTD initializes an empty C_Δ (line 3) and then steps through vertices in reverse elimination order. In our example, this means starting with R^B_{ST} (line 4). However, as an agent reinstates each interface timepoint, it simply assigns it rather than updating all incident edges as was the case in the reverse sweep of the D△PPC algorithm. Typically, before assigning a reinstated timepoint, its domain must be updated with respect to any previously assigned shared timepoints. However, because R^B_{ST} is the first reinstated variable, MaTD skips over the inner loop (lines 7-11) because there are no vertices later than it in o_S.

At this point, the domain of R^B_{ST} is guaranteed to be minimal, meaning that agent B can assign it *any* remaining value and still be guaranteed that it will be globally consistent. The choice of which value to assign is made in line 12 by the function HeuristicAssign. By default, HeuristicAssign assigns the timepoint to the midpoint of its bounds interval to split remaining flexibility down the middle; later we discuss alternative assignment heuristics. Notice that if the D△DPC algorithm had not first been executed, agent B would assign R^B_{ST} to the midpoint of its original domain of 8:00-12:00 rather than its updated domain of 8:00-9:30, which would result in an inconsistent assignment (to 10:00). However, MaTD instead adds the constraint that R^B_{ST} happens at 8:45 to C_Δ (line 14). In line 13, this is sent to agent A, because R^B_{ST} shares external edges with Ann's timepoints.

The next vertex is TR^A_{ST}. Note, agent A would consider processing this variable right away, but the inner loop (lines 7-11) forces agent A to wait for the message from agent B (line 9). Once the message arrives, agent A updates its edge weights accordingly (lines 10-11). In this case, given that TR^A_{ST} is at least 60 minutes after $R^B_{ST} = 8{:}45$, TR^A_{ST}'s domain is tightened to [9:45, 10:30]. Then in line 12, agent A chooses the decoupling point by splitting the difference, thus adding (and communicating) the constraint that TR^A_{ST} occurs at 10:08. This same process is repeated until all timepoints in V_S have been assigned. The resulting network and decoupling constraints are shown in Phase 3 of Figure 13.

To avoid inconsistency due to concurrency, before calculating decoupling constraints for v_k, an agent blocks in line 9 until it receives the fresh, newly-computed weights w_{zj}, w_{jz} from v_j's agent ($Agent(v_j)$, as sent in line 13) for each external edge $e_{jk} \in E^i_X$ where $j > k$. While this implies some sequentialization, it also allows for concurrency whenever variables do not share an external edge. For example, in Phase 3 of Figure 13, because TP^C_{ET} and R^A_{ST} do not share an edge, once agent A has assigned TR^A_{ST}, both agent A and agent C can concurrently and independently update and assign R^A_{ST} and TP^C_{ET} respectively.

Finally, for the moment skipping over optional Phase 4 in Figure 13, each agent establishes PPC in response to its new decoupling constraints by executing △PPC on its private STN in Phase 5. Our algorithm represents an improvement over previous approaches (Hunsberger, 2002; Planken, de Weerdt, & Witteveen, 2010a) in that it: (i) is distributed, rather than centralized; (ii) executes on a sparser, more efficient network representation;

Algorithm 9: Multiagent Temporal Decoupling (MaTD) with optional Relaxation (MaTDR)

```
Input: $\mathcal{G}^i$, agent $i$'s known portion of the distance graph corresponding an MaSTP
       instance $\mathcal{M}$, and a field indicating whether to RELAX decoupling
Output: $C^i_\Delta$, agent $i$'s decoupling constraints, and $\mathcal{G}^i$, agent $i$'s PPC distance graph
        w.r.t. $C^i_\Delta$
1  $\mathcal{G}^i, o^i_L, o_S = (v_1, v_2, \dots v_n) \leftarrow$D△DPC $(\mathcal{G}^i)$
2  return INCONSISTENT if D△DPC does
3  $C^i_\Delta = \emptyset$
4  for $k = n \dots 1$ such that $v_k \in V^i_I$ do
5      $V \leftarrow V \cup v_k$
6      $w^{DPC}_{zk} \leftarrow w_{zk}$, $w^{DPC}_{kz} \leftarrow w_{kz}$
7      for $j = n \dots k+1$ such that $\exists e_{jk} \in E^i$ do
8          if $e_{jk} \in E^i_X$ then
9              $w_{zj}, w_{jz} \leftarrow$ BlockReceiveUpdatedEdges($Agent(v_j)$)
10         $w_{zk} \leftarrow \min(w_{zk}, w_{zj} + w_{jk})$
11         $w_{kz} \leftarrow \min(w_{kz}, w_{kj} + w_{jz})$
12     $w_{kz}, w_{zk} \leftarrow$ HeuristicAssign($v_k, \mathcal{G}$)
13     SendUpdatedEdge($w_{zk}, w_{kz}, Agent(v_j)$ $\forall j < k, e_{jk} \in E^i_X$
14     $C^i_\Delta \leftarrow C^i_\Delta \cup \{(z - v_k \in [-w_{zk}, w_{kz}])\}$
15 if RELAX then
16     $\mathcal{G}^i, C^i_\Delta \leftarrow$ Relaxation $(\mathcal{G}^i, w^{DPC})$
17 return △PPC $(\mathcal{G}^i_{L+\Delta}, o^i_L)$, $C^i_\Delta$
```

(iii) eliminates the assumption that input graphs must be consistent; and (iv) performs decoupling as an embedded procedure within the existing D△PPC algorithm, rather than as an outer-loop.

As mentioned, a simple default for HeuristicAssign is to assign v_k to the midpoint of its directional path consistent domain (which corresponds to using the rules $w_{zk} \leftarrow w_{zk} - \frac{1}{2}(w_{zk} + w_{kz}); w_{kz} \leftarrow -w_{zk}$). In Section 5.3.2, we explore and evaluate an alternative HeuristicAssign function. In general, however, narrowing variable assignments to single values is more constraining than necessary. Fortunately, agents can optionally call a *relaxation* algorithm (introduced in Section 5.2 and shown in Figure 13 as Phase 4) that replaces C_Δ with a set of *minimal* decoupling constraints.

Theorem 8. *The MaTD algorithm has an overall time complexity of* $\mathcal{O}\left((n_P + n_S) \cdot \omega^{*2}_o\right)$, *where* $n_P = \max_i |V^i_P|$ *and* $n_S = |V_S|$, *and requires* $\mathcal{O}(m_X)$ *messages, where* $m_X = |E_X|$.

Proof. The MaTD algorithm calculates DPC and PPC in $\mathcal{O}\left((n_P + n_S) \cdot \omega^{*2}_o\right)$ time. Unary, decoupling constraints are calculated for each of n_S external variables, $v_k \in V_S$ (lines 4-14), after iterating over each of v_k's $\mathcal{O}(\omega^*_o)$ neighbors (lines 7-11). Thus decoupling requires $\mathcal{O}(n_S \cdot \omega^*_o) \subseteq \mathcal{O}\left(n_S \cdot \omega^{*2}_o\right)$ time, and so MaTD has an overall time complexity of $\mathcal{O}\left((n_P + n_S) \cdot \omega^{*2}_o\right)$. The MaTD algorithm sends exactly one message for each external constraint in line 13, for a total of $\mathcal{O}(m_X)$ messages. □

Theorem 9. *The MaTD algorithm is sound.*

Proof. Lines 1-2 return INCONSISTENT whenever the input MaSTP $\mathcal{M}$ is not consistent. By contradiction, assume that there exists some external constraint c_{xy} with bound b_{xy} that is *not* satisfied when the decoupling constraints c_{xz} and c_{zy}, calculated by MaTD, with bounds b'_{xz} and b'_{zy} respectively, are. In such a case, $b'_{xz} + b'_{zy} > b_{xy}$. WLOG, let $x < y$ in the elimination order o_S.

Note, by the time v_x is visited (in line 4), the following are true:

$$w_{xy} \leq b_{xy}; \tag{1}$$

$$w_{zy} + w_{yz} = 0; \tag{2}$$

$$b'_{zy} = w_{zy}. \tag{3}$$

(1) is true since line 1 computes DPC using o_S; (2) is true since line 12 will have already been executed for v_y, and (3) is true by construction of c_{zy} in line 14. The only update to w_{xz} occurs in line 11, and, since e_{xy} is external, one of these updates will be $w'_{xz} = \min(w_{xz}, w_{xy} + w_{yz})$, and thus

$$w'_{xz} \leq w_{xy} + w_{yz}. \tag{4}$$

Combining (1), (2), and (4), yields the fact:

$$w'_{xz} + w_{zy} \leq w_{xy} \leq b_{xy}. \tag{5}$$

The only time w'_{xz} may be further updated is in future iterations of line 11 and then possibly in line 12 to produce w''_{xz}, but both lines 11 and 12 *only tighten* (never relax). Thus, with (5) this implies that

$$w''_{xz} + w_{zy} \leq w'_{xz} + w_{zy} \leq w_{xy} \leq b_{xy}. \tag{6}$$

In line 14, c_{xz} is constructed such that $b_{xz} = w''_{xz}$; this fact, along with (3) and (6), implies $b_{xz} + b_{zy} \leq b_{xy}$. However, this is a contradiction to the assumption that $b'_{xz} + b'_{zy} > b_{xy}$, so the constraints C_Δ calculated by MaTD form a temporal decoupling of $\mathcal{M}$. ☐

Theorem 10. *The MaTD algorithm is complete.*

Proof (Sketch). The basic intuition for this proof is provided by the fact that the MaTD algorithm is simply a more general, distributed version of the basic backtrack-free assignment procedure that can be consistently applied to a DPC distance graph. We show that when we choose bounds for new, unary decoupling constraints for v_k (effectively in line 12), w_{zk}, w_{kz} are path consistent with respect to all other variables. This is because not only is the distance graph DPC, but also the updates in lines 10-11 guarantee that w_{zk}, w_{kz} are path consistent with respect to v_k for all $j > k$ (since each such path from v_j to v_k will be represented as an edge e_{jk} in the distance graph). So the only proactive edge tightening that occurs, which happens in line 13 and guarantees that $w_{zk} + w_{kz} = 0$, is done on path-consistent edges and thus will never introduce a negative cycle (or empty domain). So if the MaSTP is consistent, the MaTD algorithm is guaranteed to find a temporal decoupling. ☐

5.2 A Minimal Temporal Decoupling Relaxation Algorithm

One of the key properties of a minimal MaSTP is that it can represent a space of consistent joint schedules, which in turn can be used as a hedge against scheduling dynamism. The larger this space of solutions is, the more *flexibility* agents have to, e.g., add new constraints, while mitigating the risk of invalidating all solution schedules. ***Flexibility***, then, is a measure of the completeness of the solution space. Flexibility is good in that it increases agents' collective abilities to autonomously react to disturbances or new constraints, but at the same time, flexibility in the joint solution space limits each individual agent from unilaterally and independently exploiting this autonomy. Our MaTD algorithm explicitly trades away flexibility for increased independence during the assignment process in line 12. In this section, we describe an algorithm that attempts to recover as much flexibility as possible to maximize agents' ability to autonomously, yet independently, react to dynamic scheduling changes.

The goal of the Multiagent Temporal Decoupling with Relaxation (MaTDR), which is the version of the MaTD that executes the Relaxation algorithm (Algorithm 10) as a subprocedure, is to replace the set of decoupling constraints produced by the MaTD algorithm, C_Δ, with a set of *minimal* decoupling constraints, $C^{'}_{\Delta}$. Recall from page 111 that a minimal decoupling is one where, if the bound of any decoupling constraint $c \in C^i_\Delta$ for some agent i is relaxed, then $\{\mathcal{S}^1_{L+\Delta}, \mathcal{S}^2_{L+\Delta}, \ldots, \mathcal{S}^n_{L+\Delta}\}$ is no longer guaranteed to form a decoupling. Clearly the temporal decoupling produced when running MaTD using the default heuristic on the example problem, as shown in Phase 3 of Figure 13, is not minimal—the decoupling bounds on TP^C_{ET} could be relaxed to include its entire original domain and still be decoupled from TR^A_{ST}. The basic idea of the Relaxation algorithm is to revisit each external timepoint v_k and, while holding the domains of all other external timepoint variables constant, relax the bounds of v_k's decoupling constraints as much as possible.

We describe the execution of the algorithm by describing an execution trace over the running example problem. As depicted in Phase 4 of Figure 13, Relaxation works in original o_S order, and thus starts with TP^C_{ET}. First, Chris' agent removes TP^C_{ET}'s decoupling constraints and restores TP^C_{ET}'s domain to [9:30,10:00] by updating the corresponding edge weights to their stored DPC values (line 3). Notice that lines 2-19 are similar to the reverse sweep to the D$\triangle$PPC algorithm, except that a separate, "shadow" δ bound representation is used and updated only with respect to the original external *constraint bounds* (not tightened edge weights) to ensure that the later-constructed decoupling constraints are minimally constraining. Also, in lines 13-18, a decoupling constraint is *only* constructed when the bound of the potential, new constraint (e.g., δ_{kz}) is tighter than the already implied edge weight (e.g., when $\delta_{kz} < w_{kz}$). For example, the only constraint involving TP^C_{ET} is that it should occur before TR^A_{ST}. Therefore because TR^A_{ST} is currently set to occur at 10:08 (δ=10:08) and TP^C_{ET} is already constrained to occur before 10:00 (w =10:00), $\delta \not< w$, no decoupling constraints are added to the set $C^{'}_{\Delta}$ for TP^C_{ET} (allowing it to retain its original domain of [9:30,10:00]).

The next variable to consider is R^A_{ST}, whose domain relaxes back to [8:00,9:30]. However, since R^A_{ST} shares a synchronization constraint with R^B_{ST}, whose current domain is [8:45,8:45], Ann's agent will end up re-adopting the original decoupling constraint of $R^A_{ST} \in$ [8:45,8:45]. Next, agent A recovers TR^A_{ST}'s original DPC domain of [9:30,10:30] and tightens it to ensure

Algorithm 10: Relaxation

```
Input: $\mathcal{G}^i$, and the DPC weights, $w_{zk}^{DPC}, w_{kz}^{DPC}$, for each $v_k \in V_X^i$
Output: $C'^i_\Delta$, agent $i$'s minimal decoupling constraints, and $\mathcal{G}^i$, agent $i$'s PPC distance
        graph w.r.t. $C'^i_\Delta$
1  $C'^i_\Delta \leftarrow \emptyset$
2  for $k = 1 \dots n$ such that $v_k \in V_I^i$ do
3      $w_{zk} \leftarrow w_{zk}^{DPC}$ , $w_{kz} \leftarrow w_{kz}^{DPC}$
4      $\delta_{zk} \leftarrow \delta_{kz} \leftarrow \infty$
5      for $j = 1$ to $n$ such that $\exists e_{jk} \in E^i$ do
6          if $e_{jk} \in E_X^i$ then
7              if $j < k$ then $w_{zj}, w_{jz} \leftarrow$ BlockReceiveUpdatedEdges($Agent(v_j)$)
8              if $c_{jk}$ exists then $\delta_{zk} \leftarrow \min(\delta_{zk}, b_{jk} - w_{jz})$
9              if $c_{kj}$ exists then $\delta_{kz} \leftarrow \min(\delta_{kz}, b_{kj} - w_{zj})$
10         else if $j < k$ then
11             $w_{zk} \leftarrow \min(w_{zk}, w_{zj} + w_{jk})$
12             $w_{kz} \leftarrow \min(w_{kz}, w_{kj} + w_{jz})$
13     if $\delta_{kz} < w_{kz}$ then
14         $w_{kz} \leftarrow \delta_{kz}$
15         $C'^i_\Delta \leftarrow C'^i_\Delta \cup \{(z - v_k \leq \delta_{kz})\}$
16     if $\delta_{zk} < w_{zk}$ then
17         $w_{zk} \leftarrow \delta_{zk}$
18         $C'^i_\Delta \leftarrow C'^i_\Delta \cup \{(v_k - z \leq \delta_{zk})\}$
19     SendUpdatedEdge($w_{zk}, w_{kz}, Agent(v_j)$ $\forall j > k, e_{jk} \in E_X^i$
20 return $\mathcal{G}^i, C'^i_\Delta$
```

that it follows TP_{ET}^C's new domain of [9:30,10:00]. In this case, decoupling from TP_{ET}^C requires a new lower bound of 10:00 and results in a more flexible domain of [10:00,10:30] for TR_{ST}^A. The minimal decoupling constraints and corresponding temporal network that Relaxation calculates for the running example are presented in Phase 4 of Figure 13 for the shared portion of the network. Similarly, Phase 5 shows the implications of incorporating the decoupling constraints on each agent's PPC subproblem.

The Relaxation algorithm applies two different kinds of updates. When the edge e_{jk} considered in line 6 is local to agent i, the Relaxation algorithm executes lines 11-12, which update the actual edge weights w_{zk} and w_{kz} in the same manner as the D△PPC algorithm, guaranteeing that the values captured within the bounds interval are consistent with at least *some* value of v_j's domain (so that they will be locally PPC). On the other hand, if edge e_{jk} is external, the Relaxation algorithm instead executes lines 8-9, which update the shadow edge weights δ_{zk} and δ_{kz} in a way that guarantees they will be consistent with *all* values of v_j's domain (so that they will be temporally decoupled). Note that while these updates are more restrictive, they only lead to new decoupling constraints if they are tighter than v_k's current domain.

Theorem 11. *The* Relaxation *algorithm has an overall time complexity of* $\mathcal{O}\left(n_S \cdot \omega_o^*\right)$ *and requires* $\mathcal{O}\left(m_X\right)$ *messages.*

Proof. Unary, decoupling constraints are recalculated for each of n_S shared variables, but require visiting each of $v_k \in V_S$'s $\mathcal{O}\left(\omega_o^*\right)$ neighbors (lines 2-19), after iterating over each of v_k's $\mathcal{O}\left(\omega_o^*\right)$ neighbors (lines 5-12). Thus the Relaxation algorithm requires $\mathcal{O}\left(n_S \cdot \omega_o^*\right)$ time. The Relaxation algorithm sends exactly one message for each external constraint in line 19, for a total of $\mathcal{O}\left(m_X\right)$ messages. □

Notice that the Relaxation algorithm is called as a subroutine of the MaTD algorithm, but runs in less time, so the overall MaTDR algorithm runtime is still $\mathcal{O}\left(\left(n_P + n_S\right) \cdot \omega_o^{*2}\right)$.

Theorem 12. *The local constraints calculated by the* Relaxation *algorithm form a* minimal *temporal decoupling of* $\mathcal{S}$*.*

Proof (Sketch). The proof that the set $C_\Delta^{'}$ forms a temporal decoupling is roughly analogous to the proof for Theorem 5.1. By contradiction, we show that if the bound b_{xz} of some decoupling constraint $c_{xz} \in C_\Delta^{'}$ is relaxed by some small, positive value $\epsilon_{xz} > 0$, then $C_\Delta^{'}$ is no longer a temporal decoupling. This is because lines 8-9 imply that there exists some y such that either $b_{xz} = b_{xy} - b_{zy}$, and thus $b_{xz} + \epsilon_{xz} + b_{zy} > b_{xy}$ (and thus no longer is a temporal decoupling), or that $b_{zy} = b_{xy} - (b_{xz} + \epsilon_{xz})$ (and so is either not a decoupling or requires altering b_{zy} in order to maintain the temporal decoupling). □

5.3 Evaluation

In the following subsections, we reuse the basic experimental setup from Section 4.3 to empirically evaluate the performance of the MaTDR algorithm's computational effort in Section 5.3.1, and the impact on flexibility of variations of the MaTDR algorithm in Section 5.3.2. Like the original D△DPC algorithms, our decoupling algorithms' performance depends on the size of each agent's private subproblem *versus* the size of the shared subproblem. As the number of external constraints relative to the number of agents increases, not only can less reasoning occur independently, but also the resulting decoupled solution spaces are subject to an increasing number of local constraints, and thus diminish in flexibility.

5.3.1 Evaluation of Computational Effort

We empirically compared:

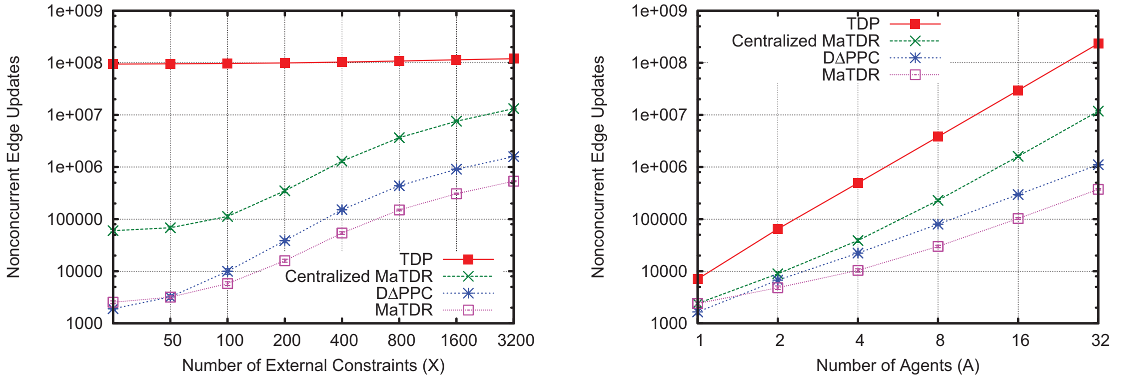
- **MaTDR** – the default MaTD algorithm with Relaxation;
- **Centralized MaTDR** – a single agent that executes MaTDR on a centralized version of the problem;
- **D△PPC** – the execution of the D△PPC distributed algorithm for establishing PPC for an MaSTP (but not a decoupling); and
- **TDP**—our implementation of the fastest variation (the RGB variation) of the (centralized) TDP algorithm as reported by Hunsberger (2002).

To implement the original TDP approach, we use the Floyd-Warshall algorithm to initially establish FPC, and the incremental update described by Planken (2008) to maintain FPC as new constraints are posted. We evaluated approaches across two metrics. The non-concurrent edge update metric, which, as described in Section 4.3, is the number computational cycles during which an edge is updated before all agents in the simulated multiagent environment have completed their executions of the algorithm. The other metric we report in this section is the total number of messages exchanged by agents.

Random Problems. We evaluate our algorithms on the same randomly generated and factory scheduling problem sets as described in Section 4.3.1. The results shown in Figure 14 demonstrate that the MaTDR algorithm clearly dominates the original TDP approach in terms of execution time, even when the MaTDR algorithm is executed in a centralized fashion, demonstrating the advantages of exploiting structure by using PPC (*versus* FPC) and dividing the problem into local and shared subproblems. The other advantage of MaTDR over the original TDP approach is that it incorporates the decoupling procedure within a single execution of constraint propagation rather than introducing decoupling constraints one at a time as an outer loop that reestablishes FPC after each new decoupling constraint is added. Additionally, when compared to the centralized version of the MaTDR algorithm, the distributed version has a speedup that varies between 19.4 and 24.7. This demonstrates that the structures of the generated problem instances support parallelism and that the distributed algorithm can exploit this structure to achieve significant amounts of parallelism.

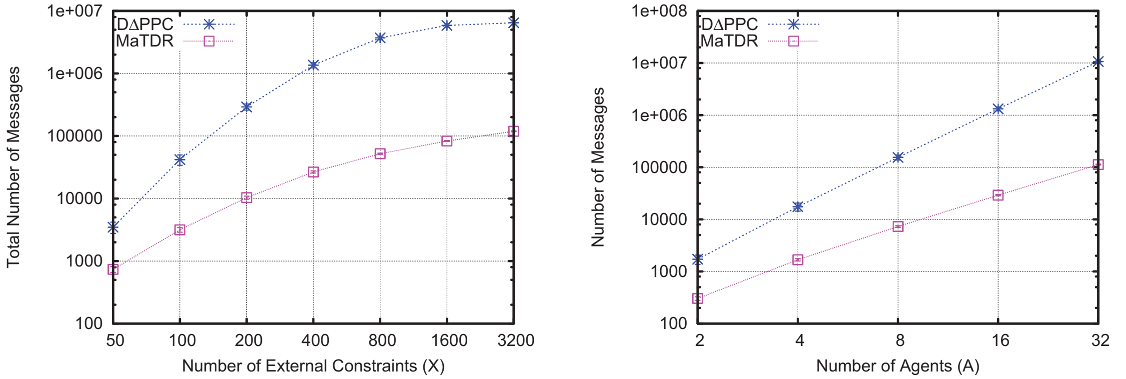
The MaTDR algorithm also dominates the D△PPC algorithm in terms of both computation and number of messages. This means the MaTDR algorithm can calculate a temporal decoupling with less computational effort than the D△PPC algorithm can establish PPC on the MaSTP. While the MaTDR is generally bound by the same runtime complexity as the D△PPC (due to both applying the D△DPC algorithm), the complexity of the actual decoupling portion of the procedure is less in practice as argued in Theorem 8. So while the MaTDR algorithm calculates and communicates new bounds for all unary reference edges, by doing so it renders all external edges moot and thus does not need to reason over them. In contrast, the D△PPC algorithm must calculate, communicate, and maintain new bounds for *every* shared edge. The total number of messages sent by both algorithms grows linearly with the number of agents and, perhaps less intuitively, sublinearly with the number of external constraints. This indicates that, as the network becomes more saturated, each new external constraint is increasingly more likely to be expressed over an edge that would already require communication, and thus requires no *new* messages.

Factory Scheduling Problems. When we also evaluated our algorithms using the factory problem set, MaTDR exhibited similarly large speedups over the Centralized MaTDR approach (ranging from 4 to 5 orders of magnitude depending on the number of tasks). We excluded the TDP approach in our charts in Figure 15 to zoom in on the differences between the (Centralized) MaTDR and D△PPC algorithms, which were much more subtle. While still achieving impressive speedups over its centralized counterpart, the MaTDR algorithm's computational advantages over D△PPC are less pronounced than before (Figure 14) due to the more loosely-coupled nature of the factory problem set. Similarly, the relative drop in



(a) Nonconcurrent computation as the number of external constraints X increases.

(b) Nonconcurrent computation as the number of agents A increases.



(c) Number of messages as the number of external constraints X increases.

(d) Number of messages as the number of agents A increases.

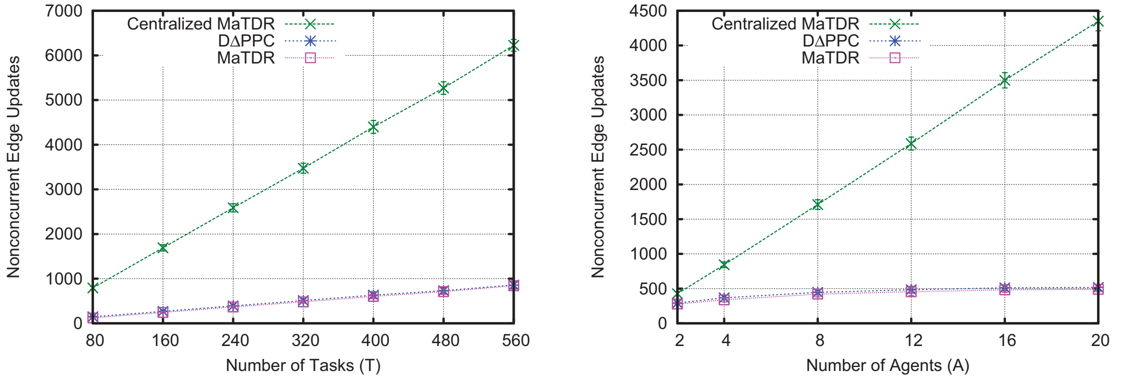
Figure 14: Empirical evaluation of MaTDR on our randomly generated problem set.

external constraints leads, in turn, to fewer shared edges, and thus fewer messages overall, mitigating the more substantial gap in Figures 14c and 14d.

The fact that the MaTDR algorithm dominates the D△PPC algorithm implies that, even if the original TDP algorithm were adapted to make use of the state-of-the-art D△PPC algorithm, the MaTDR algorithm would still outperform the revised TDP approach in terms of computational effort. Overall, we confirmed that we could exploit the structure of the MaSTP instances to calculate a temporal decoupling not only more efficiently than before, but also in a distributed manner, avoiding the (previously-required) centralization costs, and exploiting parallelism to lead to significant levels of speedup. We next ask whether the quality of a solution produced by the MaTDR algorithm is competitive in terms of the solution space completeness.

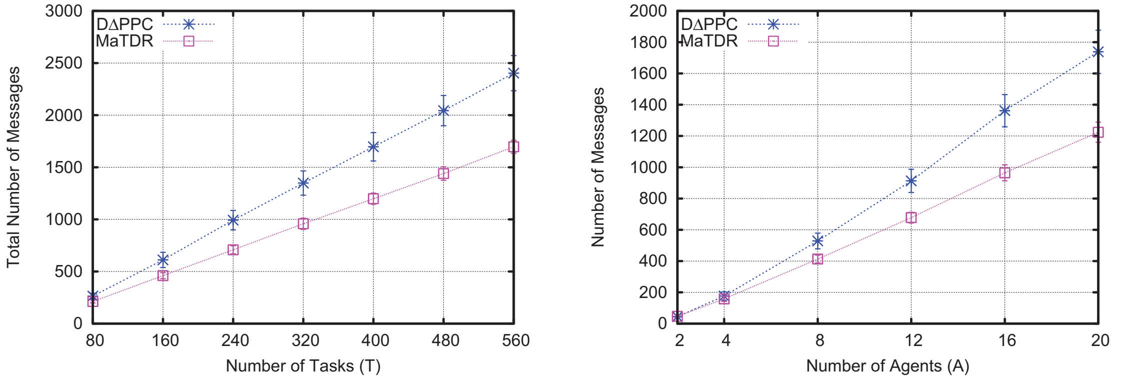
5.3.2 Evaluation of Completeness (Flexibility)

Hunsberger (2002) introduced two metrics, ***flexibility*** ($Flex$) and conversely ***rigidity*** (Rig), that act as relative measures of the number of total solutions represented by a



(a) Nonconcurrent computation as the number of tasks T increases.

(b) Nonconcurrent computation as the number of agents A increases.



(c) Number of messages as the number of tasks T increases.

(d) Number of messages as the number of agents A increases.

Figure 15: Empirical evaluation of MaTDR on the factory scheduling problem set.

temporal network, allowing the quality or "level of completeness" of alternative temporal decouplings to be compared. He defined the flexibility between a pair of timepoints, v_i and v_j, as the sum $Flex(v_i, v_j) = w_{ij} + w_{ji}$ which is always non-negative for consistent STPs. The rigidity of a pair of timepoints is defined as $Rig(v_i, v_j) = \frac{1}{1+Flex(v_i,v_j)}$, and the rigidity over an entire STP is the root mean square (RMS) value over the rigidity values for *all* pairs of timepoints:

$$Rig(\mathcal{G}) = \sqrt{\frac{2}{n \cdot (n+1)} \sum_{i<j} \left[Rig(v_i, v_j)\right]^2}.$$

This implies that $Rig(\mathcal{G}) \in [0, 1]$, where $Rig(\mathcal{G}) = 0$ when $\mathcal{G}$ has no constraints and $Rig(\mathcal{G}) = 1$ when $\mathcal{G}$ has a single solution (Hunsberger, 2002). Since $Rig(\mathcal{G})$ requires FPC to calculate, in our work we apply this metric only as a post-processing evaluation technique by centralizing and establishing FPC on the temporal decouplings returned by our algorithms. There does exist a centralized, polynomial time algorithm for calculating an optimal temporal decoupling (Planken, de Weerdt, & Witteveen, 2010a), but it requires an evaluation metric that is a linear function of distance graph edge weights, which the aggregate rigidity function, $Rig(\mathcal{G})$, unfortunately, is not.

Our MaTDR algorithm's default HeuristicAssign function assigns a timepoint to the midpoint of its currently-calculated bounds. While the assumption is that assignment to the midpoint (along with the relaxation) attempts to divide slack evenly, in practice subsequent assignments are influenced by earlier ones. For example, in Figure 13, TR_{ST}^{A} is assigned to 10:08AM, rather than 10:00AM, due to the earlier assignment of R_{ST}^{B}. Hence, perhaps there are smarter ways to assign timepoints that maintain greater amounts of flexibility. We evaluate one such alternative heuristic, the ***locality*** heuristic, which can be thought of an application of the least-constraining value heuristic that attempts to be less myopic in its assignment procedure by considering the loss of flexibility of neighboring timepoints. Here, an agent assigns v_k to the value that reduces the domains of its neighboring timepoints the least. As described by Hunsberger (2002), the original TDP approach attempts to maximize flexibility by progressively tightening the reference bounds of timepoints by a fraction of the total amount that would be required for decoupling.

Evaluation. In this set of experiments, we compare the rigidity of the temporal decouplings calculated by:

- **Midpoint** – a variant of the MaTD algorithm where HeuristicAssign uses the (default) midpoint heuristic, but without Relaxation;
- **Midpoint+R** – the MaTD algorithm using the midpoint heuristic along with the Relaxation sub-routine;
- **Locality** – a variant of the MaTD algorithm where HeuristicAssign assigns v_k to the value that reduces the domains of its neighboring, local timepoints the least (no Relaxation);
- **Locality+R** – the MaTD algorithm using the locality heuristic along with the Relaxation sub-routine;
- **Input** – the rigidity of the input MaSTN; and
- **TDP** – our implementation of Hunsberger's RLF variation of his TDP algorithm (where $r = 0.5$ and $\epsilon = 1.0$ which lead to a computational multiplier of approximately 9) that was reported to calculate the least rigid decoupling by Hunsberger (2002) (rather than the RGB variation used earlier, which was reported to be most efficient).

In our first experiment, we used our Random Problem Generator with parameter settings $A = 25$ and $X = \{50, 200, 800\}$. The left-hand side of Table 2 displays the rigidity of the temporal decoupling calculated by each approach over these problems. Unsurprisingly, as the number of external constraints increases, so does the level of rigidity across all approaches, including the inherent rigidity of the input MaSTN. Combing these rigidity results with our evaluation of computational effort displayed in Figure 14, we can look at the trade-offs between efficiency and quality of our temporal decoupling approaches. On average, as compared to Midpoint, the Midpoint+R approach decreases rigidity by 51.0% while increasing computational effort by 30.2%, and the Locality approach decreases rigidity

Table 2: A comparison of how much four different MaTD variants increase rigidity as compared to the input MaSTN and the previous centralized approach (TDP) across randomly generated and factory scheduling problem sets.

	Randomly Generated Problems			Factory Scheduling Problems		
	$X = 50$	$X = 200$	$X = 800$	$T = 80$	$T = 320$	$T = 560$
Input	0.418	0.549	0.729	0.274	0.148	0.119
Locality+R	0.508	0.699	0.878	0.756	0.838	0.838
Midpoint+R	0.496	0.699	0.886	0.765	0.853	0.855
Locality	0.621	0.842	0.988	0.965	0.983	0.982
Midpoint	0.628	0.849	0.988	0.968	0.983	0.988
TDP	0.482	0.668	0.865	0.720	0.722	0.706

by 2.0% while increasing computational effort by 146%. The Midpoint+R approach, significantly improves the output decoupling with the least computational overhead. The Locality heuristic, on the other hand, more than doubles computational overhead while providing no significant improvement in rigidity. We also explored combining these rigidity-decreasing heuristics, and while the increase in computational effort tended to be additive (the Locality+R approach increases effort by 172%), the decrease in rigidity did not. In fact, no other heuristics that we tried led to a statistically significant decrease in rigidity compared to the original Midpoint+R approach in the cases we investigated. The Locality+R approach came the closest, decreasing rigidity by 49.9% in expectation.

In our second experiment, we used the Factory Scheduling Problem benchmark with parameter settings $A = 16$ and $T = \{80, 320, 560\}$ and ensured that each problem was set with a tight makespan; these results appear in the right-hand side of Table 2. Interestingly, as the number of tasks increases, the level of rigidity of the input MaSTN decreases, while the rigidity of the calculated decouplings plateaus. This is because adding tasks does not directly correlate to adding interdependencies: only if the tasks happen to be naturally dependent on one another and assigned to different agents is the level of coupling affected. So setting a tight makespan on a set of tasks has the effect of making the critical path through the workflow rigid, but leaves many other workflow paths flexible. However, decoupling effectively introduces many additional interim makespan deadlines, which can lead to many rigid paths in the workflow. Overall, the increased structure of these problems led to starker differences between approaches. Computing a temporal decoupling results in a 3-fold increase in rigidity over the input MaSTP regardless of the decoupling approached used. Once again, including the relaxation phase decreased rigidity, by an additional 20.2% margin when using the Midpoint heuristic and by a 21.5% margin when using the Locality heuristic. The differences in flexibility between the outputs of the Midpoint(+R) and the computationally more expensive Locality(+R) approaches are insignificant.

This is far from conclusive evidence that there are no other variable assignment heuristics that would outperform either the default midpoint or locality heuristics. It does, however, point to the robustness of the MaTDR algorithm at recovering flexible decouplings. While there was not a significant difference between the Locality and Midpoint heuristics, nor the Locality+R and Minimality+R approaches, the addition of the relaxation led to a significant improvement in both cases. The fact that the MaTDR algorithm alone increases rigidity less than any other strategy can be attributed to both the structure of an MaSTP and how rigidity is measured. The MaTDR algorithm improves the distribution of flexibility to the shared timepoints reactively, instead of proactively trying to guess good values. As the MaTD algorithm tightens bounds, the general triangulated graph structure formed by the elimination order branches out the impact of this tightening. For instance, when a timepoint is assigned, it defers flexibility to its possibly many neighboring timepoints, and then during Relaxation our algorithm recovers flexibility for each these neighboring timepoints *before* recovering flexibility for the originally assigned timepoint.

Notice from Table 2 that the original TDP approach increases the rigidity the least, averaging 20.6% less rigidity than our Midpoint+R approach for the randomly generated problem set and 15.9% for the factory scheduling problems. However, this lower rigidity comes at a significant computational cost—the original TDP approach incurs, in expectation, over 10,000 *times* more computational effort than the Midpoint+R approach. Further, the original TDP approach has access to edge information involving any pair of timepoints in the entire, centralized MaSTN, clobbering all privacy, while the MaTDR makes its heuristic decoupling decisions with much more limited information, maintaining greater levels of agent privacy. While in some scheduling environments the costs of centralization (e.g., privacy) alone would discourage using the original TDP approach, in others the computational effort may be prohibitive if constraints arise faster than the centralized TDP algorithm can calculate a temporal decoupling. Further, differences in rigidity between decoupling approaches may become mitigated in scheduling problems where many external constraints enforce synchronization (e.g., Ann's and Bill's recreational start time). Because synchronization requires fully assigning timepoints in order to decouple, there may be no flexibility to recover, which makes our efficient, assignment-based MaTD algorithm the better choice.

5.3.3 Summary

Our MaTDR algorithm combines the reasoning of the D△PPC algorithm with a decoupling procedure that first assigns, then relaxes, shared timepoints in a way that leads to a minimal decoupling for an MaSTP. We showed that while this algorithm has the same complexity as the original D△PPC algorithm, it reduces solve time in expectation. Overall, on the space of problems that we investigated, the Midpoint+R approach, in expectation, outputs a high-quality temporal decoupling that approaches the quality (within 20.6% for random problems and 15.9% for factory scheduling problems) of the state-of-the-art centralized approach (Hunsberger, 2002), in a *distributed*, privacy-maintaining manner and multiple orders-of-magnitude faster than this previous centralized approach.

6. Related Work

In this section, we summarize related approaches and highlight applications that could benefit from using our MaSTP formulation and algorithms.

6.1 Simple Temporal Problem with Uncertainty

A ***Simple Temporal Problem with Uncertainty*** (STPU) (Vidal & Ghallab, 1996; Vidal & Fargier, 1999) partitions the set of constraints of an STP into a set of ***requirement links*** and a set of ***contingent links***. Requirement and contingent are both temporal difference constraints, as defined in Section 2.1. The difference is that a requirement link can be assigned by the agent whereas a contingent link is exogenously assigned outside of the control of the agent to some value $\beta_{ij} \in [-b_{ji}, b_{ij}]$. An STPU is checked not only for consistency, but also for various classes of ***controllability***, including strong, weak, and dynamic variants (Vidal, 2000; Morris, Muscettola, & Vidal, 2001; Morris & Muscettola, 2005; Lau, Li, & Yap, 2006; Hunsberger, 2009). Whereas a consistent STP is one where there exist schedules that satisfy all constraints, a controllable STPU is one where there exist satisfying schedules regardless of how the contingent links are assigned. Typical strategies for establishing or maintaining controllability include preemptively constraining requirement timepoints so that the remaining values are consistent with *all* possible contingencies. Our work does not explicitly differentiate between which constraints can be exogenously updated and which cannot. We instead use the space of all solutions to hedge against dynamic constraints. However, as a nice parallel to the STPU literature, our MaTD algorithm can be viewed as a negotiation in which agents preemptively add new decoupling constraints to control against the uncertainty and contingencies introduced by their interactions with other agents.

6.2 Applications of Multiagent Scheduling

Temporal constraint networks represent spaces of feasible schedules as compact intervals of time that can be calculated efficiently. For this reason, a temporal network naturally lends itself to online plan monitoring and dispatchable execution—an online approach whereby a dispatcher uses the flexible times representation to efficiently adapt to scheduling upheavals by introducing new constraints. The plan monitor can flag when a current schedule has become infeasible, and a dispatcher can notify agents of the time it assigns to variables immediately prior to execution (Muscettola, Morris, & Tsamardinos, 1998). Another contribution of our work is to show that these advantages extend to distributed, multiagent temporal constraint networks, while also introducing a level of independence that agents can exploit in many ways. Properties of temporal networks such as minimality and decomposability have proven essential in representing the solution space for many centralized applications such as project scheduling (Cesta, Oddi, & Smith, 2002), medical informatics (Anselma, Terenziani, Montani, & Bottrighi, 2006), air traffic control (Buzing & Witteveen, 2004), and spacecraft control (Fukunaga, Rabideau, Chien, & Yan, 1997). Unfortunately, multiagent scheduling applications (Laborie & Ghallab, 1995; Bresina et al., 2005; Castillo et al., 2006; Smith et al., 2007; Barbulescu et al., 2010) wishing to exploit these properties have previously relied on either a centralized temporal network representation or, if independence was also needed, completely disjoint, separate agent networks. We now highlight a few specific example applications that may benefit from the work described in this paper.

The approach originally described by Smith et al. (2007) and extended by Barbulescu et al. (2010) exploits the flexibility of STNs in a *distributed* framework. The general framework of the problems they solve contains models of uncertainty over both the durations and utilities of different activities. Using a greedy heuristic, their scheduler selects a set of activities that would maximize agent utility, and extracts duration bounds from the distribution of possible durations for each activity, thus creating an STP for each local agent. Each agent captures the current state of its local problem in the form of an STN representation of its local solution space, which the agent uses to help hedge against uncertainty. An agent maintains the current state of the problem as new constraints arise by using efficient, incremental STN consistency algorithms (e.g., Cesta & Oddi, 1996; Planken, de Weerdt, & Yorke-Smith, 2010b). Each agent maintains its local STN problem representation until an improved replacement STN is identified by the scheduler mechanism. This efficient state-maintenance strategy frees agents to spend a greater portion of time exploring alternative allocations and schedulings of activities between agents. Barbulescu et al.'s approach divides the problem into separate, localized STP instances, requiring a distributed state manager to react to and communicate local scheduling changes that may affect other agents. To deal with the challenge of coordination, Barbulescu et al. establish an acceptable time, δ, within which interdependent activities between agents are considered synchronized. As long as an agent can execute its activities within the prescribed δs, it can assume consistency with other agents. The risk of inconsistencies between agents is mitigated by (i) restricting synchronization scheduling to a limited time horizon and (ii) allowing agents to abandon a synchronization as soon as it is determined to be unrealizable.

Shah and Williams (2008), Shah, Conrad, and Williams (2009) and Conrad and Williams (2011) generalize this idea to multiagent, disjunctive scheduling problems by calculating *dispatchable* representations. Shah and Williams (2008) recognize that many of the disjunctive scheduling possibilities contain significant redundancy, and so gain representational efficiency by reusing the redundant portions of existing solution representations as much as possible. The algorithm propagates each disjunct using a recursive, incremental constraint compilation technique called dynamic back propagation, keeping a list of the implications of the disjunct to the temporal network. This centralized compilation leads to not only a much more compact representation than the previous approach (Tsamardinos, Pollack, & Ganchev, 2001), but also faster plan monitoring by avoiding the need to simultaneously and separately update each disparate STP.

Each of these domains offer examples of work that could benefit by putting our approach into practice. Representing the joint solution space as a multiagent temporal network could instead offer an agent a more complete view of available scheduling possibilities as well as an increased understanding of how its problem impacts (or is impacted by) other agents' problems. Further, directly representing and reasoning over the interacting scheduling problem of multiple agents also eliminates the need for agents to execute separate threads of execution to monitor and communicate state changes. Finally, directly implementing a multiagent temporal network allows agents to more flexibly and directly strike a trade-off between representing the complete joint solution space and internalizing decoupling constraints in a just-in-time manner (e.g., using the time-horizon concept of Barbulescu et al., 2010), rather than having to rely on additional mechanisms manage and coordinate state.

6.3 Distributed Finite-Domain Constraint Reasoning

The Distributed Constraint Satisfaction Problem (DCSP) is a distributed constraint-based problem formulation where all variables have a discrete, finite domain, rather than temporal, continuous domain. Two seminal algorithms for solving the DCSP are the Asynchronous Backtracking (ABT) and the Asynchronous Weak-Commitment (AWC) search algorithms (Yokoo, Durfee, Ishida, & Kuwabara, 1998). ABT, AWC, and their variants are algorithms based on asynchronous variable assignment, and use message passing of no-goods to resolve conflicts. Armstrong and Durfee (1997), Hirayama, Yokoo, and Sycara (2004), and Silaghi and Faltings (2005) provide variants of these algorithms that perform more intelligent prioritization of agents, order local agent variables dynamically, and aggregate exchanges of information for agents with complex local problems. However, approaches based on assignment and no-good learning may be less applicable to continuous-domain, constraint-based scheduling problems, where typical strategies focus on maintaining consistent sets of solutions using inference. Mailler and Lesser's Asynchronous Partial Overlay (APO) algorithm (Mailler & Lesser, 2006) deals with inconsistencies between agents through mediation, which over time centralizes the view of the problem, and thus may conflict with the privacy goals of our work. Asynchronous Forward Checking (AFC), proposed and evaluated by Meisels and Zivan (2007), provides mechanisms for asynchronously increasing consistency across agents; however the pay-off of this algorithm—avoiding expensive backtracking operations—is not a challenge faced by our MaSTP algorithms.

A Distributed Constraint Optimization Problem (DCOP) is a generalization of the DCSP with more general utility or cost functions (soft constraints) that assign a utility or cost to every possible combination of assignments to a particular subset of variables, replacing the set of hard constraints. A DCSP can be translated into a DCOP by converting constraints into cost functions with infinite cost for inconsistent assignments. As before, each utility function is known by at least one agent, and a DCOP is solved by finding the assignment of values to variables that maximizes total utility (or minimizes total cost). ADOPT (Modi, Shen, Tambe, & Yokoo, 2005) and BnB-ADOPT (Yeoh, Felner, & Koenig, 2010) are both decentralized, complete search algorithms for solving a DCOP using best-first and branch-and-bound depth-first principles respectively. The OptAPO algorithm is an optimization variant of the APO algorithm by Mailler and Lesser (2004). ADOPT and OptAPO are generalizations of AWC and APO respectively, though in each case, instead of terminating after the first feasible assignment of values to variables, agents must exhaust the entire search space to guarantee the assignment they return is the optimal one. The DPOP algorithm (Petcu & Faltings, 2005), which is also a distributed implementation of the more general bucket-elimination algorithm for solving DCOPs, is probably the most similar to our approach. While DPOP requires only a linear number of messages to solve a DCOP, it suffers from exponentially (in the induced width of the constraint graph) large message sizes. Our approach, on the other hand, can exploit the relatively simple temporal constraints to compactly encode the impact of eliminated variables using binary constraints.

6.4 Multiagent Planning

The term planning encompasses many specific problem domains including classical (Fikes & Nilsson, 1972), Hierarchical Task Networks (Erol, Hendler, & Nau, 1994), and MDP-based planning (Bellman, 1966; Sutton & Barto, 1998). At a high level, planning involves developing policies or (partially-ordered) sequences of actions that (provably or probabilistically) evolve the state of the world in a way that achieves a set of goals or optimizes some objective function. For many types of planning problems, the plan or policy must also consider types of uncertainty not typically found in scheduling (e.g., uncertainty over observations, effects of actions, etc.). In comparison, in our work, the events that are to be scheduled have already been determined and are taken as input. The output, instead of some sort of general policy or (partial) sequence of actions, is a specification of how times can be assigned to timepoint variables to satisfy the temporal constraints, where all such schedules are considered equal. Planning and scheduling, while interrelated (Myers & Smith, 1999; Garrido & Barber, 2001; Halsey, Long, & Fox, 2004), are often treated as separate subproblems (e.g., McVey, Atkins, Durfee, & Shin, 1997). Smith, Frank, and Jónsson (2000), who give a more complete comparison of planning and scheduling, suggest "the difference [between planning and scheduling] is a subtle one: scheduling problems only involve a small, fixed set of choices, while planning problems often involve cascading sets of choices that interact in complex ways".

There are many planning approaches, particularly *multiagent* planning (for a more complete introduction, see desJardins, Durfee, Ortiz Jr, & Wolverton, 1999; de Weerdt & Clement, 2009) or decentralized planning (e.g., desJardins & Wolverton, 1999; Bernstein, Zilberstein, & Immerman, 2000) that relate to and inspire our approach for solving multiagent scheduling problems. First, there is long history of exploiting loosely-coupled structure in multiagent planning; Witwicki and Durfee (2010) offer one recent example. Second, a main challenge in multiagent planning—how to interleave local planning and interagent coordination—is also apt for multiagent scheduling problems. In planning, there have been approaches where agents develop local plans and then work to integrate the plans (e.g., Georgeff, 1983), approaches that work out interdependencies between agents first and then build local plans to fit these commitments (e.g., ter Mors, Valk, & Witteveen, 2004), and approaches that blur this dichotomy by interleaving planning and coordination (e.g., Clement, Durfee, & Barrett, 2007 does this by establishing multiple levels of abstraction). Third, both planning and scheduling involve ordering events and checking for cycles (to ensure goals/conditions are not clobbered in planning and to ensure consistency in scheduling), which is particularly challenging when cycles are potentially distributed across multiple agents (Cox, Durfee, & Bartold, 2005).

6.5 Resource and Task Allocation Problems

Allocating tasks or resources to multiple agents has been studied in a variety of settings (e.g., Goldberg, Cicirello, Dias, Simmons, Smith, & Stentz, 2003; Nair, Ito, Tambe, & Marsella, 2002; Sandholm, 1993; Simmons, Apfelbaum, Fox, Goldman, Haigh, Musliner, Pelican, & Thrun, 2000; Wellman, Walsh, Wurman, & MacKie-Mason, 2001; Zlot & Stentz, 2006). Typically these problems involve either assigning a set of tasks to a limited number of agents that can perform them or, alternatively, assigning scarce resources to agents that require these resources. A common approach for handling such task allocation is a market-based approach (e.g., Nair et al., 2002), where agents place bids on tasks (or subsets of tasks in combinatorial auctions) according to their calculated costs for performing the tasks, with tasks then being assigned to the lowest bidder. Other market-based approaches allow agents to locally exchange tasks in order to quickly respond to a dynamic environment (e.g., Sandholm, 1993). While more recent approaches (Goldberg et al., 2003; Zlot & Stentz, 2006) allow agents to negotiate at various levels of task abstraction/decomposition, the primary temporal reasoning occurs within an agent, which uses scheduling information to estimate costs for its bid. Similarly, before bidding on them, agents can append temporal constraints to tasks, such as time-windows, to help capture/enforce relevant precedence constraints between tasks of different agents.

Our work assumes that the task and resource allocation problems have already been solved, or, if necessary, constraints are in place to prevent concurrent, overlapping use of a resource or duplication of a redundant activity. Additionally, whereas task/resource allocation is cast as an assignment problem, our work deals largely with reasoning over bounds so as to support flexible times representations. Finally as noted in greater detail by Zlot and Stentz (2006), while optimal, centralized approaches for solving this problem exist, the NP-hard nature of the problem, coupled with the uncertain or dynamic environment, leads to most recent approaches being both distributed as well as heuristic or approximate in nature. In contrast, our work assumes deterministic and complete approaches, but does not explicitly model the relative costs or values of various schedules. Instead, our agents provide their users with the autonomy to make their own cost/value judgments, and in turn, provide advice about the implications of their scheduling decisions.

6.6 Operations Research

Our MaSTP representations are capable of representing a particular class of scheduling problems. The Operations Research (OR) community is also interested in solving (often NP-hard) scheduling problems. In her overview comparison of the two fields, Gomes (2000, 2001) classifies OR problems as optimization problems, where the utility function tends to play an important role. Additionally, Gomes notes OR tends to represent problems using mathematical modeling languages and linear and non-linear inequalities, and uses tools such as linear programming, mixed-integer linear programming, and non-linear models. Typically, OR approaches gain traction by exploiting problem structuring. For example approaches such as timetabling or edge-finding (Laborie, 2003) are both OR techniques for tightening the bounds over when possible activities can occur. While a full review of the many OR models (e.g., Traveling Salesperson Problem, Job Shop Scheduling Problem, Resource Constrained Project Scheduling Problem, Timetabling, etc.) is beyond the scope of this paper, it is worth pointing out that synergy between the AI scheduling and OR communities is growing (Baptiste, Le Pape, & Nuijten, 1995; Barták, 1999; Laborie, 2003; Baptiste, Laborie, Le Pape, & Nuijten, 2006; Oddi, Rasconi, & Cesta, 2010).

7. Conclusion

The work we presented in this paper builds foundational algorithms for scheduling agents that assist people in managing their activities, despite distributed information, implicit constraints, costs in sharing information among agents (e.g., delays, privacy, autonomy, etc.), and the possibility of new constraints dynamically arising. Agents flexibly combine shared reasoning about interactions between their schedules with independent reasoning about their local problems. They do so by externalizing constraints that compactly summarize how their local subproblems affect each other, and internalizing new local constraints that decouple their problems from one another. This approach is most advantageous for problems where interactions between the agents are sparse compared to the complexity of agents' individual scheduling problems, where our algorithms achieve significant computational speedup over the current art.

More particularly, we contributed a formal definition of the Multiagent Simple Temporal Problem and analyzed the benefits that a distributed representation affords. We presented the D$\triangle$PPC algorithm, which is the first distributed algorithm for calculating a decentralized representation of the complete space of joint solutions, and proved that it correctly establishes partial-path consistency with a worst-case runtime complexity that is equal to the previous state-of-the-art. We empirically demonstrated that D$\triangle$PPC scales to problems containing more agents better than its state-of-the-art centralized counterpart, and exhibits a steady margin of speedup. Exactly how effectively it load-balances computational effort depends on the number of external constraints coupling agents' problems, where D$\triangle$PPC achieves up to a 22-fold reduction in nonconcurrent computational effort over $\triangle$PPC for problems with 25 agents in the problems we studied. Additionally, we formally defined the Multiagent Temporal Decoupling Problem along with the first distributed algorithm for solving it. We proved that the MaTD algorithm is correct, and demonstrated both analytically and empirically that it calculates a temporal decoupling upwards of four orders-of-magnitude faster than previous approaches, exploiting sparse structure and parallelism when it exists. Additionally, we introduced the Relaxation algorithm for relaxing the bounds of existing decoupling constraints to form a *minimal* temporal decoupling, and empirically showed that this algorithm can decrease rigidity by upwards of 50% (within 15-21% of the state-of-the-art centralized approach) while increasing computational effort by as little as 20% in the problems we studied. Overall, we have shown that the combination of the MaTD and MaTDR algorithms can calculate a temporal decoupling faster than the D$\triangle$PPC algorithm, and four orders-of-magnitude faster than the previous state-of-the-art centralized TDP algorithm.

This work leads to many interesting ongoing and future research directions. First, the use of an MaSTN to monitor plan execution and dispatch scheduling advice to users in a *distributed manner* (as was done for centralized dispatching in, e.g., Drake, see Conrad & Williams, 2011) will rely on an ability to efficiently maintain MaSTNs (Boerkoel et al., 2013) and a on comprehensive empirical understanding of the inherent trade-offs between the costs of maintaining PPC *versus* the loss of flexibility in temporal decouplings. Second, the MaSTP formulation could be adapted to include models of uncertainty or utility, as it has for the centralized STP (Rossi, Venable, & Yorke-Smith, 2006). Optimally solving the temporal decoupling problem for general models of utility is an NP-hard problem, even in the centralized case (Planken, de Weerdt, & Witteveen, 2010a). Thus, designing mechanisms that can augment our distributed temporal decoupling algorithm and encourage agents to solve the MaTDP in both a distributed and *globally optimal* (rather than just locally minimal) manner is an open challenge that spans both multiagent scheduling and algorithmic game theory. Finally, the ideas presented here for MaSTP can be extended to more-general, disjunctive scheduling problems (Boerkoel & Durfee, 2012, 2013). The combinatorics of disjunctive problems leads to significant challenges—finding solutions is NP-hard, and the structure of the underlying temporal network, which for the inherently binary MaSTP is known *a priori*, can become conflated or tangled by the combinatorial number of possible disjunctive choices.

Acknowledgments

This work was supported, in part, by the NSF under grants IIS-0534280 and IIS-0964512, the AFOSR under Contract No. FA9550-07-1-0262, and the 2011 Rackham Predoctoral Fellowship. We would like to thank Léon Planken, Michael Wellman, Martha Pollack, Amy Cohn, Stefan Witwicki, Jason Sleight, Elizabeth Boerkoel, and Julie Shah for their helpful suggestions.

References

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

Armstrong, A., & Durfee, E. (1997). Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 620–625.

Baptiste, P., Laborie, P., Le Pape, C., & Nuijten, W. (2006). Constraint-based scheduling and planning. *Foundations of Artificial Intelligence*, *2*, 761–799.

Baptiste, P., Le Pape, C., & Nuijten, W. (1995). Incorporating efficient operations research algorithms in constraint-based scheduling. In *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research*.

Barbulescu, L., Rubinstein, Z. B., Smith, S. F., & Zimmerman, T. L. (2010). Distributed coordination of mobile agent teams: The advantage of planning ahead. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pp. 1331–1338.

Barták, R. (1999). Constraint programming: In pursuit of the holy grail. In *Proceedings of the Week of Doctoral Students (WDS99 -invited lecture)*, pp. 555–564.

Bellman, R. (1966). Dynamic programming. *Science*, *153*(3731), 34–37.

Bernstein, D., Zilberstein, S., & Immerman, N. (2000). The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 32–37.

Bliek, C., & Sam-Haroud, D. (1999). Path consistency on triangulated constraint graphs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, Vol. 16, pp. 456–461.

Boerkoel, J. (2012). *Distributed Approaches for Solving Constraint-based Multiagent Scheduling Problems*. Ph.D. thesis, University of Michigan.

Boerkoel, J., & Durfee, E. (2009). Evaluating hybrid constraint tightening for scheduling agents. In *Proceedings of The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pp. 673–680.

Boerkoel, J., & Durfee, E. (2010). A comparison of algorithms for solving the multiagent simple temporal problem. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pp. 26–33.

Boerkoel, J., & Durfee, E. (2011). Distributed algorithms for solving the multiagent temporal decoupling problem. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pp. 141–148.

Boerkoel, J., & Durfee, E. (2012). A distributed approach to summarizing spaces of multiagent schedules. *In Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, 1742–1748.

Boerkoel, J., & Durfee, E. (2013). Decoupling the multiagent disjunctive temporal problem. *In Proceedings of the Twenty-Seventh Conference on Artificial Intelligence (AAAI-13)*, To appear.

Boerkoel, J., Planken, L., Wilcox, R., & Shah, J. (2013). Distributed algorithms for incrementally maintaining multiagent simple temporal networks. *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, To appear.

Bresina, J., Jónsson, A. K., Morris, P., & Rajan, K. (2005). Activity planning for the Mars exploration rovers. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, pp. 40–49.

Buzing, P., & Witteveen, C. (2004). Distributed (re) planning with preference information. In *Proceedings of the Sixteenth Belgium-Netherlands Conference on Artificial Intelligence*, pp. 155–162.

Castillo, L., Fernández-Olivares, J., & O. García-Pérez, F. P. (2006). Efficiently handling temporal knowledge in an HTN planner. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pp. 63–72.

Cesta, A., & Oddi, A. (1996). Gaining efficiency and flexibility in the simple temporal problem. In *Proceedings of the 3rd Workshop on Temporal Representation and Reasoning (TIME'96)*, pp. 45–50.

Cesta, A., Oddi, A., & Smith, S. (2002). A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, *8*(1), 109–136.

Clement, B., Durfee, E., & Barrett, A. (2007). Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research*, *28*(1), 453–515.

Conrad, P., & Williams, B. (2011). Drake: An efficient executive for temporal plans with choice. *Journal of Artificial Intelligence Research*, *42*(1), 607–659.

Cox, J., Durfee, E., & Bartold, T. (2005). A distributed framework for solving the multiagent plan coordination problem. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pp. 821–827.

de Weerdt, M., & Clement, B. (2009). Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, *5*(4), 345–355.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. In *Knowledge Representation*, Vol. 49, pp. 61–95.

Dechter, R., & Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, *34*(1), 1–38.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, *113*(1-2), 41–85.

desJardins, M., & Wolverton, M. (1999). Coordinating a distributed planning system. *AI Magazine*, *20*(4), 45–53.

desJardins, M. E., Durfee, E. H., Ortiz Jr, C. L., & Wolverton, M. J. (1999). A survey of research in distributed, continual planning. *AI Magazine*, *20*(4), 13–22.

Erol, K., Hendler, J., & Nau, D. S. (1994). Semantics for hierarchical task-network planning. Tech. rep., University of Maryland at College Park, College Park, MD, USA.

Fikes, R., & Nilsson, N. (1972). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, *2*(3-4), 189–208.

Floyd, R. (1962). Algorithm 97: Shortest path. *Communications of the ACM*, *5*(6), 345.

Fukunaga, A., Rabideau, G., Chien, S., & Yan, D. (1997). Aspen: A framework for automated planning and scheduling of spacecraft control and operations. In *Proceedings International Symposium on AI, Robotics and Automation in Space*.

Garrido, A., & Barber, F. (2001). Integrating planning and scheduling. *Applied Artificial Intelligence*, *15*(5), 471–491.

Georgeff, M. (1983). Communication and interaction in multi-agent planning. In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*, pp. 125–129.

Goldberg, D., Cicirello, V., Dias, M., Simmons, R., Smith, S., & Stentz, A. (2003). Market-based multi-robot planning in a distributed layered architecture. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the 2003 International Workshop on Multi-Robot Systems*, Vol. 2, pp. 27–38.

Golumbic, M. (1980). *Algorithmic graph theory and perfect graphs*. Academic Press.

Gomes, C. (2000). Artificial intelligence and operations research: Challenges and opportunities in planning and scheduling. *The Knowledge Engineering Review*, *15*(1), 1–10.

Gomes, C. (2001). On the intersection of AI and OR. *The Knowledge Engineering Review*, *16*(1), 1–4.

Halsey, K., Long, D., & Fox, M. (2004). Crikey-a temporal planner looking at the integration of scheduling and planning. In *ICAPS Workshop on Integrating Planning into Scheduling*, pp. 46–52.

Hirayama, K., Yokoo, M., & Sycara, K. (2004). An easy-hard-easy cost profile in distributed constraint satisfaction. *Transactions of Information Processing Society of Japan*, *45*, 2217–2225.

Hunsberger, L. (2002). Algorithms for a temporal decoupling problem in multi-agent planning. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pp. 468–475.

Hunsberger, L. (2009). Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *Procedings of 16th*

International Symposium on Temporal Representation and Reasoning(TIME 2009), pp. 155–162.

Kjaerulff, U. (1990). Triangulation of graphs - algorithms giving small total state space. Tech. rep., Aalborg University.

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

Laborie, P., & Ghallab, M. (1995). Planning with sharable resource constraints. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1643–1649.

Lau, H. C., Li, J., & Yap, R. H. (2006). Robust controllability of temporal constraint networks under uncertainty. In *Tools with Artificial Intelligence, 2006. ICTAI'06. 18th IEEE International Conference on*, pp. 288–296.

Mailler, R., & Lesser, V. (2006). Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *Journal of Artificial Intelligence Research*, *25*(1), 529–576.

Mailler, R., & Lesser, V. (2004). Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pp. 438–445.

McVey, C., Atkins, E., Durfee, E., & Shin, K. (1997). Development of iterative real-time scheduler to planner feedback. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1267–1272.

Meisels, A., & Zivan, R. (2007). Asynchronous forward-checking for DisCSPs. *Constraints*, *12*(1), 131–150.

Modi, P., Shen, W., Tambe, M., & Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1-2), 149–180.

Morris, P., & Muscettola, N. (2005). Temporal dynamic controllability revisited. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pp. 1193–1198.

Morris, P., Muscettola, N., & Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 494–502.

Muscettola, N., Morris, P., & Tsamardinos, I. (1998). Reformulating temporal plans for efficient execution. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR 1998)*, pp. 444–452.

Myers, K., & Smith, S. (1999). Issues in the integration of planning and scheduling for enterprise control. *Proceedings of the DARPA Symposium on Advances in Enterprise Control.*, 217–223.

Nair, R., Ito, T., Tambe, M., & Marsella, S. (2002). Task allocation in the robocup rescue simulation domain: A short note. In *Proceedings of the RoboCup 2001: Robot Soccer World Cup V*, pp. 751–754.

Oddi, A., Rasconi, R., & Cesta, A. (2010). Casting project scheduling with time windows as a DTP. In *Proceedings of the ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2010)*, pp. 42–49.

Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *International Joint Conference on Artificial Intelligence (IJCAI-05)*, Vol. 19, pp. 266–271.

Planken, L. (2008). Incrementally solving the STP by enforcing partial path consistency. In *Proceedings of the Twenty-seventh Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2008)*, pp. 87–94.

Planken, L., de Weerdt, M., & van der Krogt, R. (2008). P^3C: A new algorithm for the simple temporal problem. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pp. 256–263.

Planken, L. R., de Weerdt, M. M., & Witteveen, C. (2010a). Optimal temporal decoupling in multiagent systems. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pp. 789–796.

Planken, L. R., de Weerdt, M. M., & Yorke-Smith, N. (2010b). Incrementally solving STNs by enforcing partial path consistency. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pp. 129–136.

Rossi, F., Venable, K., & Yorke-Smith, N. (2006). Uncertainty in soft temporal constraint problems: A general framework and controllability algorithms for the fuzzy case. *Journal of Artificial Intelligence Research*, *27*(1), 617–674.

Sandholm, T. (1993). An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-93)*, pp. 256–256.

Shah, J., & Williams, B. (2008). Fast dynamic scheduling of disjunctive temporal constraint networks through incremental compilation. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pp. 322–329.

Shah, J. A., Conrad, P. R., & Williams, B. C. (2009). Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pp. 289–296.

Silaghi, M., & Faltings, B. (2005). Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence*, *161*(1-2), 25–53.

Simmons, R., Apfelbaum, D., Fox, D., Goldman, R., Haigh, K., Musliner, D., Pelican, M., & Thrun, S. (2000). Coordinated deployment of multiple, heterogeneous robots. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pp. 2254–2260.

Smith, D., Frank, J., & Jónsson, A. (2000). Bridging the gap between planning and scheduling. *The Knowledge Engineering Review*, *15*(1), 47–83.

Smith, S., Gallagher, A., Zimmerman, T., Barbulescu, L., & Rubinstein, Z. (2007). Distributed management of flexible times schedules. In *Proceedings of the Sixth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, pp. 472–479.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*, Vol. 1. Cambridge Univ Press.

ter Mors, A. W., Valk, J. M., & Witteveen, C. (2004). Coordinating autonomous planners. In *Proceedings of the International Conference on Artificial Intelligence (ICAI'04)*, pp. 795–801.

Tsamardinos, I., Pollack, M., & Ganchev, P. (2001). Flexible dispatch of disjunctive plans. In *Proceedings of the Sixth European Conference in Planning (ECP-06)*, pp. 417–422.

Vidal, T. (2000). Controllability characterization and checking in contingent temporal constraint networks. In *Principles of Knowledge Representation and Reasoning-International Conference*, pp. 559–570.

Vidal, T., & Fargier, H. (1999). Handling contingency in temporal constraint networks: From consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, *11*(1), 23–45.

Vidal, T., & Ghallab, M. (1996). Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI-96)*, pp. 48–54.

Wellman, M., Walsh, W., Wurman, P., & MacKie-Mason, J. (2001). Auction protocols for decentralized scheduling. *Games and Economic Behavior*, *35*(1/2), 271–303.

Witwicki, S. J., & Durfee, E. H. (2010). Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pp. 185–192.

Xu, L., & Choueiry, B. (2003). A new efficient algorithm for solving the simple temporal problem. In *Proceedings of the Tenth International Symposium on Temporal Representation and Reasoning, 2003 and Fourth International Conference on Temporal Logic (TIME-ICTL 03)*, pp. 212–222.

Yeoh, W., Felner, A., & Koenig, S. (2010). BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, *38*, 85–133.

Yokoo, M., Durfee, E., Ishida, T., & Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, *10*(5), 673–685.

Zlot, R., & Stentz, A. (2006). Market-based multirobot coordination for complex tasks. *The International Journal of Robotics Research*, *25*(1), 73–101.