Fast Scheduling of Robot Teams Performing Tasks with Temporospatial Constraints

Matthew C. Gombolay, Ronald J. Wilcox, and Julie A. Shah
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
gombolay@csail.mit.edu, rjwilcox@mit.edu, julie_a_shah@csail.mit.edu

***Abstract*—The application of robotics to traditionally manual manufacturing processes requires careful coordination between human and robotic agents in order to support safe and efficient coordinated work. Tasks must be allocated to agents and sequenced according to temporal and spatial constraints. Also, systems must be capable of responding on-the-fly to disturbances and people working in close physical proximity to robots. In this paper, we present a centralized algorithm, named "Tercio," that handles tightly intercoupled temporal and spatial constraints. Our key innovation is a fast, satisficing multi-agent task sequencer inspired by real-time processor scheduling techniques and adapted to leverage hierarchical problem structure. We use this sequencer in conjunction with a MILP solver and empirically demonstrate the ability to generate near-optimal schedules for real-world problems an order of magnitude larger than those reported in prior art. Finally, we demonstrate the use of our algorithm in a multi-robot hardware testbed.**

***Index Terms*—Scheduling, Human-Robot Teaming**

1. Introduction

Robotic systems are increasingly entering into domains previously occupied exclusively by humans. In the manufacturing field, for example, there is a strong economic motivation to enable human and robotic agents to cooperatively perform traditionally manual work. This integration requires a choreography of human and robotic work that meets upper- and lowerbound temporal deadlines for task completion (e.g., the assigned work must be completed within a single shift) and spatial restrictions on agent proximity (e.g., robots must maintain at least 4 meters of separation from other agents) in order to support safe and efficient human-robot cooperation. Multi-agent coordination problems with temporospatial constraints can be readily formulated as a mixed-integer linear program (MILP) (Brucker, 2001; Lenstra and Kan, 1978); however, the problem of optimally scheduling $n \geq 3$ tasks (each with a sequence of n_i subtasks) on a set of m machines is NP-Hard (Sotskov and Shakhlevich, 1995) and is computationally intractable for problems of interest to large-scale factory operations.

Various decentralized or distributed approaches have achieved fast computation and favorable scalability characteristics (Choi et al., 2009; Castañón and Wu, 2003; Curtis and Murphey, 2003; Ren et al., 2005; Tal Shima and Chandler, 2005). Rapid computation is desirable because it enables the system to respond on-the-fly to schedule disturbances, such as an inaccurately performed job, a machine breakdown, or changing temporal constraints (Ahmed et al., 2005; Castañón and Wu, 2003; Sariel and Balch, 2005). Some approaches boost computational performance by decomposing plan constraints and contributions to the objective function among agents (Choi et al., 2009). However, these methods break down when agents' schedules become tightly intercoupled, such as scenarios wherein multiple agents are maneuvering in close physical proximity to one another. While distributed coordination approaches are necessary for field operations in which environment and geography affect communication among agents, factory operations allow sufficient connectivity and bandwidth for either centralized or distributed approaches to task assignment and scheduling.

In this paper, we present Tercio[1], a centralized task assignment and scheduling algorithm that scales to multi-agent, factory-size problems and supports on-the-fly scheduling in the presence of temporal and spatial proximity constraints. We empirically demonstrate that this capability enables human and robotic agents to perform manufacturing tasks effectively and in close proximity to one another.

Tercio takes as input a set of tasks composed of precedence-related subtasks; a set of interval temporal constraints relating the start and finish times of subtasks; two-dimensional coordinates specifying the spatial locations where subtasks are performed; physical constraints restricting agent proximity; a set of agent capabilities specifying the tasks and subtasks each agent may perform and the minimum, maximum, and expected time for each agent to complete each task; and an objective function to optimize. Tercio provides a solution consisting of the assignment of agents (i.e., humans or robots) to tasks and a schedule for each agent's jobs (i.e., start and finish times for each job) such that all temporal and spatial proximity constraints are satisfied and the objective function is empirically within 10% of optimal the majority of the time (see Section 10).

Tercio is made efficient through a fast, satisficing multi-agent task sequencer that is inspired by real-time processor scheduling techniques but adapted to leverage hierarchical problem structure. Our task sequencer computes, in polyno-

Matthew Gombolay's present affiliation is with the Georgia Institute of Technology, Atlanta, GA 30332, USA. This work was supported by Boeing Research and Technology and by the National Science Foundation (NSF) Graduate Research Fellowship Program (GRFP) under grant number 2388357.

[1]"Tercio" is named for the Spanish military formation used during the Renaissance period, which consisted of several different types of troops, each with their own strengths, working together as a single unit.

mial time, a multi-agent schedule satisfying upper- and lower-bound temporal deadlines, as well as spatial restrictions of agent proximity. Although the sequencing algorithm is satisficing and incomplete, we empirically show that it produces task schedules within 10 percent of optimal. We also demonstrate that it produces solutions for nearly all problem instances solvable by a complete solution technique. We use this fast task sequencer in conjunction with a MILP solver, and indicate that Tercio is able to generate task assignments and schedules for up to 10 agents and 1,000 subtasks in approximately one minute, on average. In this regard, Tercio scales better than representative state-of-the-art benchmarks in heuristic (i.e., Nunes and Gini (2015)), meta-heuristic (i.e., Zhang and Wong (2015)), and exact (i.e., (Gurobi Optimization, 2016)) approaches. Tercio also returns flexible time windows for execution (Wilcox et al., 2012), enabling agents to adapt to small disturbances online without the need for a full re-computation of the schedule.

Tercio was first introduced by Gombolay et al. (2013)[2]. In this paper, we present two advancements for this work, proofs of correctness, and further analysis of the algorithm's performance. First, we generalize the types of task sets that Tercio can efficiently schedule. Second, we present an improved sequencer that produces less-conservative schedules than the method originally presented (Gombolay et al., 2013). Third, we include proofs of correctness for Tercio's subroutines. Finally, we expand the empirical evaluation of Tercio to demonstrate its significance over the closest available benchmarks in prior literature Gurobi Optimization (2016); Nunes and Gini (2015); Zhang and Wong (2015).

2. Related Work

There is a wealth of prior work related to task assignment and scheduling within the manufacturing field and other applications. Korsah *et al.* provided a comprehensive taxonomy (Korsah et al., 2013) for the multi-robot task allocation and scheduling problem. According to this taxonomy, our problem fits within the cross-schedule dependencies [XD] category, with single-task robots [ST], single-robot tasks [SR] and the time-extended allocation [TA] problem (XD [ST-SR-TA]). Cross-schedule dependencies exist when the utility of one agent is directly affected by the scheduling commitment of another. As noted by Korsah (2011), these dependencies occur in the form of various temporal constraints (e.g., precedence constraints) or of finite-capacity resources that must be shared by agents to complete their tasks. We consider both temporal and resource-based cross-schedule dependencies in this work. We note that prior works often seek to find a feasible schedule with the shortest duration, although other application-specific cost functions are sometimes considered.

While in many cases the problem of task allocation with cross-schedule dependencies can be readily formulated and solved as a MILP (Brucker, 2001; Lenstra and Kan, 1978), the complexity MILP-based solution techniques (e.g., branch-and-bound search) is exponential, leading to computational intractability for large-scale factory operations. Various algorithms have been proposed with the goal of achieving favorable scalability characteristics. Here, we survey solution techniques for scheduling problems similar to the problem we present in this work – including MILP formulations, auction- and market-based methods, and other hybrid approaches – and discuss the applicability of these prior techniques to our problem. Table 1 depicts a summary of the relevant approaches.

[2]Parts of this work have been published in Proc. of Robotics: Science and Systems (RSS 2013).

	ST vs. MT	SR vs. MR	TA	XD	Heterogeneous Agents	Precedence/Wait Constraints	Absolute Task Deadlines	Task-Task Deadlines	Shared Resources	Optimal	Complete
Tercio, 2017	ST	SR	✓	✓	✓	✓	✓	✓	✓		
McIntire et al., 2016	ST	SR	✓	✓	✓	✓	✓				
Nunes & Gini, 2015	ST	SR	✓		✓		✓				
Sung et al., 2015	ST	MR		✓		✓					
Zhang & Wong, 2015	ST	SR	✓	✓						Probabilistic	
Liu & Shell, 2013	ST	SR			✓					✓	✓
Erdem et al., 2013	MT	MR	✓	✓	✓	✓	✓		✓	✓	✓
Castro & Petrovic, 2012	ST	SR	✓	✓	✓	✓	✓	✓			
Ren & Tang, 2009	ST	SR	✓	✓	✓	✓	✓				
Choi et al., 2009b	ST	SR			✓					≤ 50%	Yes
Chen & Askin, 2009	MT	MR	✓	✓	✓	✓	✓				
Li & Womer, 2008	ST	MR	✓	✓	✓	✓	✓				
Zavlano et al., 2008	ST	SR	✓		✓					✓	✓
Mercier et al., 2005	MT	MR	✓				✓	✓	✓	✓	✓
Hooker, 2004	MT	SR	✓	✓	✓	✓	✓			✓	✓
Lemaire et al., 2004	ST	SR	✓	✓	✓	✓	✓	✓			
Tan, 2004	ST	SR	✓	✓	✓	✓	✓				
Cesta et al., 2002	ST	SR	✓				✓			Probabilistic	
Harujunkowski & Grossman, 2002	ST	SR	✓	✓	✓	✓	✓			✓	✓
Cordeau et al., 2001	ST	SR	✓	✓	✓	✓	✓	✓			
Chien et al., 2000	ST	SR	✓	✓		✓	✓	✓	✓		
Sycara et al., 1991	MT	MR	✓	✓	✓	✓	✓		✓		

Fig. 1: This table provides a summary of relevant prior work.

2.1. MILP/CP Solution Techniques

One promising approach to solving this class of problems has been the development of a hybrid algorithm incorporating MILP and constraint programming (CP) methods along with decomposition. Techniques based on Benders Decomposition (Benders, 1962; Geoffrion, 1972; Martin, 1999) are among the most widely used. Benders Decomposition works by iteratively updating a function, $f(\bar{y})$, which provides a lowerbound on the optimal solution for an optimization problem where y is a subset of the decision variables and $\bar{y}$ indicates a specific assignment to those decision variables. For a scheduling problem, y might represent the assignment of agents to jobs, and the remaining variables might represent potential sequences of those jobs. Through bifurcation of the decision variables, it is possible to improve the computation time for a solution to the optimization problem (Hooker, 2000).

Various scheduling applications have incorporated Benders Decomposition. Logic-Based Benders Decomposition in particular has been applied to solve job shop scheduling problems where n tasks must be scheduled at m facilities (Hooker, 2000). Benders Cuts (i.e., a lowerbound constraint on the optimal solution provided by $f(\bar{y})$) are generated by separating the allocation of tasks from the sequencing of those tasks. Reported results have demonstrated optimal solutions for problems involving approximately 10 facilities and 50 tasks.

Cordeau et al. (2001) and Mercier et al. (2005) employed Benders Decomposition for aircraft routing and crew scheduling (Cordeau et al., 2001). The authors heuristically decomposed aircraft routing from crew assignment and generated Benders Cuts for the resulting assignment sub-problem. They empirically demonstrated that this method produced more cost-efficient schedules than prior art. Rekik *et al.* similarly employed Benders Decomposition to schedule personnel shift-work (Rekik et al., 2004). The authors applied hand-tailored *forward* and *backward constraints* to cut the search space, and proved the correctness of these constraints. Rekik *et al.* showed that Benders Decomposition can be used to solve particularly challenging problems in which the *forward* and *backward constraints* do not sufficiently prune the search space.

While Benders Decomposition has served as the basis for many state-of-the-art scheduling algorithms, several alternative techniques have also successfully combined MILP and CP approaches. Jain and Grossmann (Jain and Grossmann, 2001) presented an iterative method that first solves a relaxed MILP formulation and then searches for the complete solution using CP. When applied to the scheduling of jobs for machines, the MILP relaxation solves for the assignment of jobs and the CP solves for the schedule. If a solution is identified, the algorithm returns the optimal solution; otherwise, the algorithm infers cuts based on the solution of the MILP relaxation and solves the new MILP relaxation using these cuts. The authors reported results for problems involving up to 20 tasks and 5 machines. Li and Womer later improved on this work by employing a hybrid Benders Decomposition algorithm with MILP and CP solver subroutines (Li and Womer, 2008), and reported that their method can solve problems involving 30 tasks and eight different agent types up to four times faster than a standard MILP. Ren and Tang (Ren and Tang, 2009) took a similar approach, but employed heuristic strategies to generate informative cuts in the event that the CP solver was unable to identify a feasible task sequence. A related method proposed by Harjunkoski and Grossman (2002) utilized an iterative approach to producing task assignments and schedules. Both Ren and Tang and Harjunkoski and Grossman empirically demonstrated that their algorithms can solve problems involving up to eight machines and 36 jobs; however, these works did not address problems associated with cross-schedule dependencies.

Although not mathematical programs, some prior works have incorporated PDDL-style problem formulations. For example, Erdem et al. (2012b, 2013) developed distributed and semi-distributed techniques for scheduling problems involving precedence and absolute temporal constraints, as well as other resource and spatial proximity constraints. These works utilize a formulation that supports causality-based reasoning. The Erdem et al. (2012b, 2013) technique yielded optimal solutions for problems with absolute deadlines and complex geometric constraints, and readily scaled to problems involving up to eight agents and 80 tasks.

2.2. Auction and Market-Based Solution Techniques

Auction methods and other market-based approaches to scheduling problems, such as those developed by Ponda et al., Choi *et al.*, and Liu and Shell, also frequently rely upon decomposition of the problem structure (Choi et al., 2009; Liu and Shell, 2013; Ponda et al., 2010). For example, Ponda et al. (2010) developed a decentralized, market-based solution technique for allocating tasks to agents. given that tasks are constrained by time windows. Ponda et al. (2010) employed the Consensus-Based Bundle Algorithm (CBBA) (Choi et al., 2009), in which the objective function and constraints are decomposed by agent so that each agent can quickly solve for the value of its bid on each task. Thus, while the work by Ponda et al. (2010) represents an added capability for CBBA, the scalability of the algorithm is strictly worse than CBBA.

Recently, Nunes and Gini (2015) developed the Temporal Sequential Single-Item (TeSSI) auction algorithm for decentralized scheduling, which has been shown to outperform CBBA. TeSSI takes as input a task set in the form of a simple temporal problem (STP); each task has an earliest start and latest finish time (absolute wait and deadline constraints). Constraints relating tasks are comprised of travel time constraints; resource constraints are not included. Nunes and Gini (2015) empirically demonstrated their approach yields improved performance compared with CBBA (Choi et al., 2009). Further, their approach has since been extended to incorporate precedence relations among tasks (McIntire et al., 2016). However, TeSSI and its variants (Nunes and Gini, 2015; McIntire et al., 2016) solve a narrower class of problem than Tercio, in that while they handle some cross-schedule dependencies in the form of precedence relations, they do not handle subtask-to-subtask deadlines (i.e., deadlines constraining the maximum time between the start time of one subtask and the finish time of the second) or resource capacity constraints.

Other techniques solve the task allocation problem efficiently (Sung et al., 2013; Liu and Shell, 2013; Bertsekas, 1990; Zavlanos et al., 2008; Beard et al., 2002), but do not address the sequencing problem. For example, Sung *et al.* addressed multi-robot task allocation where each agent maintains a queue of tasks and partial information about other agents' queues (Sung et al., 2013). During execution, agents communicate when possible and choose to exchange tasks using a heuristic approach. Sung *et al.* empirically demonstrated that their algorithm can solve problems involving up to six agents and up to 250 tasks; however, the problems did not involve cross-schedule dependencies or task deadlines.

Liu and Shell recently proposed a novel distributed method for task allocation via strategic pricing (Liu and Shell, 2013). Their work builds upon prior approaches to distributed auction algorithms (Bertsekas, 1990; Zavlanos et al., 2008), runs in polynomial time, and produces globally optimal solutions. However, this technique does not consider coupling constraints – for example, a problem in which one agent's assignment directly affects the domain of feasible assignments for other agents, as is the case when agents are performing tasks subject to temporal and resource constraints. Chien et al. proposed planning methods for a team of robotic rovers to accomplish a set of scientific objectives (Chien et al., 2000). The rovers needed to complete a set of tasks where each task required the use of shared, single-access resources (i.e., "shared resources"). The approach by Chien et al. (2000)

uses an iterative-repair centralized planner coupled with an auction algorithm to perform centralized goal allocation and decentralized route planning and goal sequencing. Chien *et al.* benchmarked against a set of randomized problems with three rovers and 12 goals; however, thorough empirical evaluation with an optimal benchmark was not reported.

Lemaire *et al.* approached the problem of allocating UAVs to tasks, represented by a bipartite graph (Lemaire et al., 2004). Here, one set of tasks (UAV navigation) was required to be completed before the second set (target sensing). The authors first presented a centralized auction solution and reported empirical results for a problem involving 50 tasks and four agents. Next, they described a distributed approach wherein an auctioneer agent assigns the first set of tasks to the multi-robot team, and then the second set of tasks is auctioned. This method supports rescheduling in light of dynamic disturbances occurring during task execution; however, the authors did not report empirical results for their distributed method. Sycara et al. (1991) explored the problem of task allocation and sequencing for a set of agents. In this work, agents were required to share a finite set of resources necessary for task execution. The authors formulation was distributed in nature and relied upon multiple heuristics to sequentially construct an effective schedule; however, their approach was suboptimal and did not consider deadline constraints relating tasks.

2.3. Hybrid Solution Techniques

Other hybrid approaches have integrated heuristic schedulers within the MILP solver to achieve better scalability characteristics. For example, Chen *et al.* incorporated depth-first search (DFS) with heuristic scheduling (Chen and Askin, 2009). In this approach, a DFS algorithm sequentially assigns tasks to agents, and a heuristic scheduling algorithm sequences the tasks according to a minimum slack priority. The algorithm also employs heuristics to guide the order in which tasks are assigned to resources during the search. Chen benchmarked their work on problems involving approximately 50 tasks and 10 resources (or agents) using a standard problem database (Patterson, 1984; Drexl et al., 2000).

Alternatively, Castro and Petrovic (2012) used a heuristic scheduler to seed a feasible schedule for the MILP with regard to patient procedures conducted within a hospital. This method incorporates a tiered approach to minimize a three-term objective function: First, a heuristic scheduling algorithm generates an initial feasible solution. Next, a MILP is solved for first term of the objective function using the heuristic solution as a seed schedule. The MILP is solved again to optimize the second objective function term, using the solution from the first MILP as a constraint. This process repeats, but for the third objective function term. The solution time is reduced by sequentially optimizing the objective function terms; however, this approach sacrifices global optimality.

Other approaches perform cooperative scheduling by incorporating Tabu Search within a MILP solver (Tan, 2004), or by applying heuristics to abstract the problem to groupings of agents (Kushleyev et al., 2013). These hybrid methods are able to solve scheduling problems involving 5 agents (or groups of agents) and 50 tasks in minutes or seconds, and address problems incorporating multiple resources, task precedence, and temporal constraints relating task start and end times to the plan epoch time. However, these approaches do not take more general, task-task temporal constraints into consideration.

Cesta et al. (2002) addressed the problem of project scheduling with time windows by formulating it as a constraint satisfaction problem. In their work, candidate (potentially infeasible) solutions are initially generated using heuristic methods; random and deterministic heuristic techniques are then used to iteratively repair any infeasibilities in the problem. Cesta et al. (2002) noted that randomization is essential to counteract the bias of greedy scheduling heuristics; however, they did not consider wait constraints that create cross-schedule dependencies, nor did they consider shared resources.

2.4. Meta-Heuristic Techniques

Successful meta-heuristic methods have included simulated annealing (SA) and genetic algorithms (GA). Davis (1985) produced one of the early works applying GAs to job shop scheduling, although many researchers have since followed suit (Falkenauer and Bouffouix, 1991; Fang et al., 1993; Godinho Filho et al., 2014; Zhang et al., 1997). Recently, Zhang and Wong (2015) developed a GA to perform process planning for single-task machines, single-machine tasks, and time-extended scheduling with precedence constraints; however, the formulation did not consider deadline constraints or shared-resource constraints.

Researchers have also sought to apply simulated annealing techniques to specific scheduling problems (Mousavi and Tavakkoli-Moghaddam, 2013; Osman and Potts, 1989; Van Laarhoven et al., 1992). Prior works have combined GAs and SA to further improve upon solution quality (Dai et al., 2013; Wang and Zheng, 2002). These techniques rely upon a random walk through the space of possible schedules; as the number of steps in the walk (i.e., algorithm iterations) approaches ∞, the optimal solution will be found.

These solution techniques are typically applied to job shop scheduling problems in which tasks are related through neither tightly intercoupled upper- and lowerbound temporal constraints nor shared resource constraints (Falkenauer and Bouffouix, 1991; Fang et al., 1993; Zhang and Wong, 2015; Zhang et al., 1997). As they are only probabilistically optimal and rely upon random search, meta-heuristics can require a large amount of computation time in order to identify and improve schedules when tight upperbound constraints exist.

2.5. Application to Task Assignment and Scheduling with Temporospatial Constraints

The problem of scheduling – allocating agents to tasks and sequencing those tasks – has been studied in a wide array of works incorporating various solution techniques. However, prior research does not address the need to quickly solve large-scale problems involving tight dependencies among agents' schedules, which can make decomposition problematic. Typically, allowing multiple robots to work closely within the same physical space produces dependencies among the agents'

temporal and spatial constraints, leading to uninformative decompositions. We are unaware of prior work that has yielded a solution technique for time-extended scheduling of heterogeneous agents in which each unit of work (i.e., subtask) is related through upper- and lowerbound temporal constraints without restriction on problem structure (e.g., only some subtasks can be related by certain constraints), where agents must share access to resources (e.g., physical locations) when performing their work.

While we are not aware of prior work focused on our problem definition, we are able to adapt heuristic, meta-heuristic, and exact solution techniques to provide an informative empirical validation. In particular, we benchmarked our solution technique, Tercio, against the following techniques:

1) We adapted the insertion-heuristic based TeSSI algorithm (Nunes and Gini, 2015) to accommodate our class of problems. Insertion heuristics are commonly used in vehicle routing and scheduling (Campbell and Savelsbergh, 2004; Fernandez-Viagas and Framinan, 2014; Solomon, 1987).
2) A state-of-the-art GA proposed by Zhang and Wong (2015) for integrated process planning and scheduling.
3) An exact, MILP-based solution technique, Gurobi (Gurobi Optimization, 2016), which uses state-of-the-art branch-and-bound search and cutting plane heuristics, for the mathematical formulation presented in Section 3.

In Section 10, we show that our technique has the singular ability to balance computational efficiency with solution quality to provide fast, empirically near-optimal solutions for large-scale scheduling problems.

3. Problem Statement

In this section, we formulate a task assignment and scheduling problem for multiple robots working within a shared physical space.

Tercio takes the following as input:

- $\boldsymbol{\tau}$: The set of all tasks to be performed. Each task $\tau_i \in \boldsymbol{\tau}$ is composed of m_i precedence-constrained subtasks, denoted τ_i^j, for $j = [1, m_i]$, where a subtask τ_i^{j+1} may not be released to execute until its predecessor, τ_i^j, finishes executing. Note: $|\boldsymbol{\tau}|$ is the number of subtasks in $\boldsymbol{\tau}$.
- $\boldsymbol{A}$: The set of all agents that perform subtasks.
- $\boldsymbol{TC}$: The set of **Interval Temporal Constraints** (Dechter et al., 1991) relating subtasks' start and finish times. Lowerbound "wait constraints" are denoted as $W_{\langle \tau_i^j, \tau_x^y \rangle}$, and specify that τ_x^y starts at least $W \geq 0$ time units after τ_i^j ends. Upperbound "deadline constraints" are denoted as $D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$, and specify that τ_x^y must finish within $D^{s2s} > 0$ time units of the start of τ_i^j. "Absolute" deadline constraints $D^{abs}_{\tau_i^j}$ limit the finish time of τ_i^j to within $D^{abs} > 0$ time units of the epoch start time.
- $\boldsymbol{\tau_R}$: The set of all subtask pairs separated by less than the allowable spatial proximity, derived from spatial locations on the factory floor where subtasks are performed (specified via two-dimensional Cartesian coordinates).
- $\boldsymbol{AC}$: The set of **agent capabilities** specifying the subtasks each agent $a \in \boldsymbol{A}$ may perform, as well as that agent's minimum $lb^a_{\tau_i^j}$, maximum $ub^a_{\tau_i^j}$, and expected completion time $C^a_{\tau_i^j}$ for each subtask τ_i^j.
- z: An **objective function** to minimize that includes the makespan and possibly other application-specific terms.

A solution to the problem consists of an assignment (i.e., $\forall \tau_i^j, \exists! a \in \boldsymbol{A}$ s.t. $A^a_{\tau_i^j} = 1$) of each subtask $\tau_i^j \in \boldsymbol{\tau}$ to one agent $a \in \boldsymbol{A}$ and the assignment of the start $s_i^j \in \mathbb{R}$ and finish time $f_i^j \in \mathbb{R}$ of each subtask such that all constraints (Equations 2-12) are satisfied and the objective function (Equation 1) is minimized.

Remark (On Complexity)**.** *Sotskov and Shakhlevich (1995) proved that the following problem (denoted SS'95) is NP-Hard: Find a minimum makespan schedule for a set of machines $M_l \in M$ (i.e., agents $a \in A$) completing a set of jobs $J_i \in J$ (i.e. tasks $\tau_i \in \boldsymbol{\tau}$) consisting of operations $l_i^j \in J_i$ (i.e., $\tau_i^j \in \boldsymbol{\tau}$) with duration t_i^j such that l_i^j must be completed before l_i^{j+1} (i.e., $\boldsymbol{TC} = \{W_{\langle \tau_i^j, \tau_i^{j+1} \rangle} | \tau_i^j, \tau_i^{j+1} \in \boldsymbol{\tau}\}$), and operation $l_{i,j}$ must be completed on machine M_j (i.e., a fixed agent assignment such that $\left(A^a_{\tau_i^j} = 1 | j \neq a\right) \rightarrow$ infeasible), where $|M|, |J| \geq 3$. In this paper, we consider a problem specification (denoted GWS'17) that is a superset of SS'95 by including general deadline, wait, and resource constraints, and a variable agent assignment. One can map in polynomial time a problem in SS'95 to one in GWS'17 by specifying the fixed agent assignment (i.e., $C^a_{\tau_i^j} \leftarrow \infty, \forall j \neq a$ and $C^a_{\tau_i^j} \leftarrow t_i^j$ otherwise) in $O(|\boldsymbol{\tau}||A|)$ time. TC consists only of the wait constraints between consecutive subtasks (i.e., $\boldsymbol{TC} = \{W_{\langle \tau_i^j, \tau_i^{j+1} \rangle} | \tau_i^j, \tau_i^{j+1} \in \boldsymbol{\tau}\}$) and $\boldsymbol{\tau_R} = \emptyset$. As such, there exists a mapping in P from a problem in SS'95 to one in GWS'17. Therefore, we can say that SS'95 $\leq_P$ GWS'17. As problems in SS'95 are NP-Hard, the problem we address are at least NP-Hard provided that P≠NP.*

The multi-agent coordination problem (with temporospatial constraints) can be readily formulated as a MILP as follows. In this formulation, $A^a_{\tau_i^j} \in \{0, 1\}$ is a binary decision variable for the assignment of agent a to subtask τ_i^j, $x_{\langle \tau_i^j, \tau_x^y \rangle} \in \{0, 1\}$ is a binary decision variable specifying whether τ_i^j comes after or before τ_x^y, and $s_i^j, f_i^j \in [0, \infty)$ are the start and finish times of τ_i^j. $\boldsymbol{A}$ is the set of all agents a, $\boldsymbol{\tau}$ is the set of all tasks and subtasks, and $\boldsymbol{\tau_R}$ is the set of all subtask pairs $\langle \tau_i^j, \tau_x^y \rangle$ that are separated by less than the allowable spatial proximity. M is an artificial variable set to a large positive number, and is used to encode conditional constraints.

Equation 2 ensures that each task is assigned to one agent. Equation 3 encodes the explicit ordering of subtasks according to the lowerbound temporal constraints. Equations 4 and 5 encode the minimum wait times and upperbound deadline constraints, respectively, between pairs of subtasks. Equation 6 enforces the absolute deadlines constraining subtask finish times. Equations 7 and 8 ensure that agents are not required to complete tasks any faster or slower than they are capable of. Note that human and robotic workers inherently have

varying capabilities. For example, a robot may be able to place composite material, whereas a human might be able to perform a more dexterous task, such as wire-laying. Also, two humans are likely to perform the same task at different rates. In this work, we assume that workers are heterogeneous agents, and that each subtask has an agent-specific completion time. Equations 9 and 10 sequence actions to ensure that agents maintain safe buffer distances from one another while performing tasks. Equations 11 and 12 ensure that each agent only performs one task at a time. Note that Equations 9 and 10 couple the variables relevant to sequencing and spatial proximity constraints and to task start and end times, and produce tight dependencies among agents' schedules.

$$\text{Objective: } \min(z) \tag{1}$$

subject to

$$\sum_{a \in \boldsymbol{A}} A^a_{\tau_i^j} = 1, \forall \tau_i^j \in \boldsymbol{\tau} \tag{2}$$

$$x_{\langle \tau_i^j, \tau_x^y \rangle} = 1, \forall W_{\langle \tau_i^j, \tau_x^y \rangle} \in \boldsymbol{TC} \tag{3}$$

$$W_{\langle \tau_i^j, \tau_x^y \rangle} \leq s_x^y - f_i^j, \forall W_{\langle \tau_i^j, \tau_x^y \rangle} \in \boldsymbol{TC} \tag{4}$$

$$D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle} \geq f_x^y - s_i^j, \forall D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle} \in \boldsymbol{TC} \tag{5}$$

$$D^{abs}_{\tau_i^j} \geq f_i^j, \forall D^{abs}_{\tau_i^j} \in \boldsymbol{TC} \tag{6}$$

$$f_i^j - s_i^j \geq lb^a_{\tau_i^j} - M\left(1 - A^a_{\tau_i^j}\right), \forall \tau_i^j \in \boldsymbol{\tau}, a \in \boldsymbol{A} \tag{7}$$

$$f_i^j - s_i^j \leq ub^a_{\tau_i^j} + M\left(1 - A^a_{\tau_i^j}\right), \forall \tau_i^j \in \boldsymbol{\tau}, a \in \boldsymbol{A} \tag{8}$$

$$s_x^y - f_i^j \geq M\left(x_{\langle \tau_i^j, \tau_x^y \rangle} - 1\right), \forall \left\langle \tau_x^y, \tau_i^j \right\rangle \in \boldsymbol{\tau_R} \tag{9}$$

$$s_i^j - f_x^y \geq -M x_{\langle \tau_i^j, \tau_x^y \rangle} \forall \left\langle \tau_i^j, \tau_x^y \right\rangle \in \boldsymbol{\tau_R} \tag{10}$$

$$s_x^y - f_i^j \geq M\left(1 + x_{\langle \tau_i^j, \tau_x^y \rangle} - A^a_{\tau_i^j} - A^a_{\tau_x^y}\right), \forall \tau_i^j, \tau_x^y \in \boldsymbol{\tau} \tag{11}$$

$$s_i^j - f_x^y \geq M\left(2 - x_{\langle \tau_i^j, \tau_x^y \rangle} - A^a_{\tau_i^j} - A^a_{\tau_x^y}\right), \forall \tau_i^j, \tau_x^y \in \boldsymbol{\tau} \tag{12}$$

Employing branch-and-bound search to identify the optimal solution in this MILP-based formulation requires $O\left(2^{|A||\boldsymbol{\tau}|)^3}\right)$ just in terms of the integer variables for allocation ($A^a_{\tau_i^j}$) and sequencing ($x_{\langle \tau_i^j, \tau_x^y \rangle}$). Note that the number of possible sequencing permutations is $O\left(|\boldsymbol{\tau}|!\right)$, while the number of sequencing variables in this formulation is $O\left(|\boldsymbol{\tau}|^2\right)$. Within the manufacturing settings of interest, the number of tasks and subtasks is typically much larger than the number of agents, so the computational bottleneck when solving for a schedule occurs within the sequencing sub-problem.

As noted in Section 2, other related works have included formulation of similar scheduling problems as MILPs: For example, Korsah et al. (2013) proposed a general formulation of the instantaneous assignment problem for the XD class. However, our formulation considers time-extended scheduling in which one must also determine how to sequence jobs. Pinto and Grossmann (1995) proposed a MILP formulation that bears similarities to ours: both consider assignments of agents to tasks (Equation 2), as well as the ordering of tasks for machines (or agents) as given by Equations 11 and 12. However, our MILP formulation includes shared resource constraints (Equation 9 and 10), as well as the unique capabilities or production rates of agents (Equations 7 and Equations 8).Other works have considered separate aspects of our problem, such as inclusion of heterogeneous agents (Topcuoglu et al. (1999)) or shared resources (Jones et al. (2011)). However, these works do not propose a mathematical formulation that addresses the XD [ST-SR-TA] problem variant with resource constraints.

Fig. 2: This figure depicts the Boeing 777 Fuselage Upright Autonomous Build Process. Courtesy: The Boeing Company.

Tercio approximately solves our problem by producing sub-optimal makespan solutions. Section 10 demonstrates through empirical evaluation that the produced makespans are within 10% of optimal for the range of problems evaluated. To preserve temporal flexibility at execution, the solution is then reformulated as a Simple Temporal Problem that is flexibly dispatched (Wilcox et al., 2012) (Section 4.1).

3.1. Real-World Motivation Example

Our problem statement is motivated by real-world applications, such as the Boeing 777 Fully Autonomous Upright Build (FAUB) project (pictured in Figure 2). This type of application requires the coordination of six to eight robots for successful assembly of an aerospace structure. Precedence constraints among the work packages must be respected to ensure structural integrity during the build process. Task-task temporal constraints are imposed by process requirements for the timed application of sealant. The robots' work must be sequenced to ensure mutually exclusive access to tools, utilities, and floor space; efficient solutions to the problem involve an intricate choreography of robot movements. A fast dynamic scheduling method is necessary to efficiently and effectively respond to schedule disruptions such as those caused by process time variation, re-work, and inspection.

4. TERCIO

In this section, we present Tercio, a centralized task assignment and scheduling algorithm that scales to multi-agent, factory-size problems and supports on-the-fly rescheduling with temporal and spatial proximity constraints. Figure 3 depicts the system architecture, and the pseudocode for the Tercio algorithm is presented in Figure 4. Tercio is made efficient through a fast, multi-agent task sequencer inspired by real-time processor scheduling techniques but adapted to leverage the hierarchical problem structure, wherein tasks are composed of precedence-related subtasks. Our approach decomposes the problem statement in Section 3 into sub-problems of task allocation and sequencing. We use a MILP-based solution method (Section 5) to allocate agents to tasks and a polynomial-time task sequencer to efficiently solve the task sequencing problem (Section 6). Although the task sequencer is satisficing and incomplete, it substantially improves

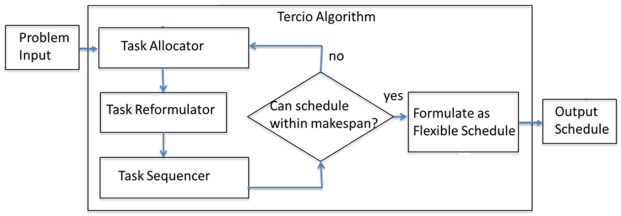

Fig. 3: This figure depicts the system architecture of Tercio.

worst-case time complexity. (We present time complexity analysis for each component of Tercio at the conclusion of their respective description below.)

4.1. Tercio Pseudocode

Tercio takes the inputs defined in Section 3, along with a user-specified makespan *cutoff* intended to terminate the optimization process. This *cutoff* can often be derived from the temporal constraints of the manufacturing process: for example, a user may specify that a provided task set must be completed within an 8-hour shift. Tercio works by iterating through agent allocations until a schedule can be identified that satisfies the maximum allowable makespan for the problem. However, Tercio can also run as an anytime heuristic, terminating once the allotted time has expired.

As shown in Figure 4, Tercio initializes the makespan (Line 1) and previous solution (Line 2), and then iterates Lines 3-7 to compute a schedule that meets this makespan. A third-party optimizer (Gurobi) solves the agent-allocation MILP (Line 4) and returns the agent allocation matrix, A. Interval temporal (TC) constraints are updated according to this matrix by tightening task time intervals (Line 5). For example, if a task is originally designated to take between 5 and 15 minutes, but the assigned robot can complete it in no fewer than 10 minutes, the interval tightens from $[5, 15]$ to $[10, 15]$.

The agent allocation, the capability-updated TCs and the spatial map of tasks are then provided as input to the Tercio multi-agent task sequencer (Line 6). The task sequencer (described further in Section 6) returns a tight upperbound on the optimal makespan for the given agent allocation, as well as a sequence of tasks for each agent. While this makespan is longer than *cutoff* (or while the algorithm's runtime has not exceeded the specified timeout), the algorithm iterates Lines 3-7, each time adding a constraint (Line 7) to exclude previously attempted agent allocations. Tercio terminates when either the returned makespan falls beneath *cutoff* or when no solution can be found after iterating through all feasible agent allocations. Note that, for each iteration, a search tree is generated for the agent allocation. We propose preserving the search tree across iterations in future work to reduce computation time, updating it with excluded allocations.

If the cutoff makespan is satisfied, agent and spatial resource sequencing constraints (interval form of $[0, \infty)$) are added to TC (Line 9). The resulting Simple Temporal Problem, composed of the interval temporal constraints, is compiled into a dispatchable form (Line 10) (Muscettola et al., 1998; Wilcox et al., 2012) which guarantees that, for any consistent

TERCIO(τ, A, TC, $\tau_{\mathbf{R}}$, AC, z, cutoff/timeout)

1: makespan $\leftarrow$ inf
2: exclusions $\leftarrow \emptyset$
3: **while** makespan $>$ cutoff or runtime $<$ timeout **do**
4: A $\leftarrow$ ALLOCATION(z, $\mathbf{A}$, τ, TC, AC, $\tau_{\mathbf{R}}$, exclusions)
5: TC $\leftarrow$ update agent capabilities
6: makespan,seq $\leftarrow$ SEQUENCER($\mathbf{A}, \mathbf{TC}, \tau_{\mathbf{R}}$)
7: exclusions $\leftarrow$ exclusions $\cup$ $\mathbf{A}$
8: **end while**
9: TC $\leftarrow$ add ordering constraints to enforce seq
10: STP $\leftarrow$ DISPATCHABLE(TC)
11: **return** STP

Fig. 4: Pseudocode for the Tercio Algorithm.

choice of a time point within a flexible window, there exists a solution that can be identified through one-step propagation of interval bounds. The dispatchable form maintains flexibility to increase robustness to disturbances, and has been shown to decrease the amount of time spent re-computing solutions in response to disturbances for randomly generated structured problems by up to 75% (Wilcox et al., 2012).

5. TERCIO: AGENT ALLOCATION

The Tercio Agent Allocator performs agent-task allocation by solving an MILP that includes Equations 2, 7, and 8, ensuring that each task is assigned to exactly one agent and that the allocation does not violate the upper- and lowerbound temporal constraints. In this work, we investigate the objective of minimizing schedule makespan. This corresponds to solving the MILP defined in Section 3 according to the objective function depicted in Equation 13, where we seek the optimal makespan (i.e., the overall process duration).

$$\min(z),\ z = g_{MILP}(A, \boldsymbol{\tau}) = \max_{\langle \tau_i^j, \tau_x^y \rangle} \left(f_i^j - s_x^y \right) \tag{13}$$

However, decomposition of task allocation and sequencing necessitates an objective function that guides the task allocation subroutine toward solutions that are likely to be favorable for the sequencing subroutine. As such, we developed the following objective function (Equation 14), comprised of three mixed-integer linear terms (Equations 15-16). Equation 15 minimizes the maximum work assigned to any one agent, which mitigates situations resulting in a single agent bottlenecking the schedule. Equation 16 minimizes the total amount of work time (i.e. "agent-hours") by selecting the most efficient agent for each subtask. As such, Tercio's agent-allocation subroutine maximizes Equation 14 subject to Equations 15 and 16. In our MILP-based task allocation subroutine, binary decision variable $A^a_{\tau_i^j}$ represents the assignment of agent a to subtask τ_i^j. The worst-case time complexity of assigning one of $a = |A|$ agents to each subtask in $\boldsymbol{\tau}$ via branch-and-bound search is given by $O\left(2^{|A||\boldsymbol{\tau}|}\right)$.

$$\min(z),\ z = \alpha_1 g_1(A, \boldsymbol{\tau}) + \alpha_2 g_2(A, \boldsymbol{\tau}) \tag{14}$$

$$g_1(A, \boldsymbol{\tau}) \geq \sum_{\tau_i^j \in \boldsymbol{\tau}} A^a_{\tau_i^j} \times C^a_{\tau_i^j}, \forall a \in \boldsymbol{A} \tag{15}$$

$$g_2(A, \boldsymbol{\tau}) \geq \sum_{a \in \boldsymbol{A}} \sum_{\tau_i^j \in \boldsymbol{\tau}} A^a_{\tau_i^j} \times C^a_{\tau_i^j} \tag{16}$$

6. TERCIO: MULTI-AGENT TASK SEQUENCER

The Tercio Task Sequencer takes the problem defined in Section 3 as input, along with a set of task assignments provided by the Tercio Agent Allocator described in Section 5. The Task Sequencer is satisficing, meaning the produced schedule merely satisfies the constraints of the problem (Equations 2 -12) and does not take an objective function as input. The Task Sequencer returns a valid task sequence if the algorithm can identify one. Tercio schedules tasks in simulation using the dynamic priority policy Earliest-Deadline First (EDF), as well as an online schedulability test that guarantees satisfaction of temporospatial constraints for any opportunistic scheduling policy (i.e., a policy that executes a task if one is available to execute). We formulate this schedulability test as a CP problem and determine whether a full, feasible schedule can be developed if subtask τ_i^j is scheduled at time t.

The Tercio Task Sequencer is inspired by real-time processor scheduling techniques and operates on the structure of a real-time processor scheduling model, called the "self-suspending task model." The sequencer performs a rapid variant of "edge-checking," (similar to that performed in Laborie (2003) and Vilím et al. (2005)), which we call the Multiprocessor Russian Dolls Test. To our knowledge, the Tercio Task Sequencer is the first real-time scheduling method for multiprocessor systems that tests the schedulability of non-preemptive, self-suspending tasks in scenarios where multiple tasks have more than one self-suspension and tasks are constrained by shared memory resources.

We now outline the steps taken by the Tercio Task Sequencer. In Section 7, we discuss the relationship of our problem to prior art in real-time processor scheduling and describe our problem as a real-time processor scheduling problem. We then present how the problem depicted in Section 3 is reformulated into a real-time processor scheduling problem. Finally, in Section 8, we present the Task Sequencer's priority policy and Multiprocessor Russian Dolls Test.

6.1. Multi-Agent Task Sequencer Walk-Through

Here, we address the mechanics of our task sequencing algorithm, as depicted in Figure 5. At Line 1, the task set is reformulated (if necessary) into a specific structure that makes the task sequencer operate more efficiently. The task sequencer requires that every subtask τ_i^j involved in an absolute deadline constraint $D^{abs}_{\tau_i^j}$ or subtask-to-subtask deadline constraint $D^{s2s}_{\langle \tau_a^b, \tau_i^j \rangle}$ have only one predecessor subtask τ_x^y. This means that either $\tau_x^y = \tau_i^{j-1}$, or else there exists a lowerbound wait constraint $W_{\langle \tau_x^y, \tau_i^j \rangle}$. In turn, every τ_x^y must have exactly one such predecessor subtask. This recurses until reaching either the epoch subtask τ_0^0 in the case of an absolute deadline constraint, or τ_a^b in the case of a subtask-to-subtask deadline.

In Section 7.3, we prove that the reformulation process preserves correctness – any schedule that satisfies the constraints of the reformulated task set will also satisfy the constraints of the original set. The ability to reformulate depends upon the laxity of the deadlines within the task set and the structure of the constraints. The tighter the deadlines and the more connected the constraint graph, the less able the algorithm will be to reformulate the problem into the structure our schedulability test requires. However, this more restricted structure enables the our schedulability test to compute an empirically tight schedulability test in polynomial time.

TERCIO-SEQUENCER($\boldsymbol{\tau}$)

1: reformulate $\boldsymbol{D^{s2s}}$ and $\boldsymbol{W}$
2: $t \leftarrow 0$ ▷ set simulation time to zero
3: ensure temporal feasibility of $\boldsymbol{D^{abs}}$
4: **while** true **do**
5: $availableTasks \leftarrow$ get and sort subtasks ready for execution at time t according to EDF
6: **for** all $\tau_i^j \in$availableTasks **do**
7: **if** executing τ_i^j at t will not violate temporospatial constraints **then**
8: $seq \leftarrow$ schedule τ_i^j
9: **end if**
10: **end for**
11: $t \leftarrow t + 1$ ▷ increment simulation time
12: **if** all tasks executed **then** break;
13: **end if**
14: **end while**
15: $makespan, seq \leftarrow$ extract multi-agent schedule
16: **return** $makespan, seq$

Fig. 5: Pseudocode for the Tercio Multi-agent Task Sequencer.

In Line 2, simulation time is initialized to zero. In Line 3, the algorithm determines whether the set of absolute deadlines $\boldsymbol{D^{abs}}$ in the reformulated task set is temporally consistent, meaning that the set of agents will be able to successfully schedule against those deadline constraints and their associated spatial constraints. This is determined using the Multiprocessor Russian Dolls Test, as described in Section 8. After ensuring schedule feasibility due to $\boldsymbol{D^{abs}}$, the algorithm begins to schedule all subtasks in $\boldsymbol{\tau}$ in simulation (Lines 4-14). In Line 5, the algorithm prioritizes the order in which it attempts to schedule available subtasks according to the Earliest-Deadline First (EDF) dynamic scheduling priority. EDF, commonly used in real-time systems (Harbour and Palencia, 2003; Zhang and Burns, 2009; Zhao et al., 2007) and multi-agent scheduling (Chen et al., 2014), attempts to execute the task τ_i^j with the earliest (smallest in magnitude) deadline d_i^j first.

Line 6 iterates over all available subtasks τ_i^j and applies the Multiprocessor Russian Dolls Test (Line 7) to determine whether a subtask can feasibly be scheduled at time t while satisfying the temporospatial constraints. If feasible, the subtask is scheduled at time t (Line 8) and an All-Pairs Shortest Path (APSP) computation (Johnson, 1954; Floyd, 1962) is performed to update the temporal constraints. The simulation time is incremented (Line 11) and the algorithm continues until all tasks have been scheduled (Line 12). Finally, the algorithm returns the makespan and subtask sequence (Line 15).

7. REAL-TIME PROCESSOR SCHEDULING ANALOGY FOR THE TASK SEQUENCER

The design of our informative, polynomial-time task sequencer was inspired by a processor scheduling analogy in which each agent is a computer processor able to perform one task at a time, and a physical location in discretized space is

modeled as a shared memory resource accessible by only one processor at a time. Wait constraints (lowerbounds on interval temporal constraints) are modeled as "self-suspensions" (Lakshmanan and Rajkumar, 2010; Ridouard and Richard, 2006) – times during which a task is blocked while another piece of hardware completes a different, time-durative task.

Typically, assembly manufacturing tasks have more structure (e.g., parallel and sequential subcomponents) and more complex temporal constraints than real-time processor scheduling problems. AI scheduling methods address complex temporal constraints and gain computational tractability by leveraging hierarchical structure within the plan (Smith et al., 2000). We bridged AI and real-time processor scheduling in order to develop a fast multi-agent task sequencer that satisfies tightly coupled upper- and lowerbound temporal deadlines and spatial proximity restrictions (shared resource constraints).

The Tercio Task Sequencer operates on an augmented *self-suspending task model* (defined in Section 7.2) and returns a valid task sequence if the algorithm can identify one. The task sequencer is satisficing and incomplete; however, we have empirically validated that it returns makespans within 10% of the optimal when integrated with the Tercio Agent Allocation algorithm (see Section 10).

Processor scheduling of *self-suspending task systems* has been the focus of much prior work due to the integration of relatively recent hardware and supporting software systems (e.g., GPUs, PPUs) that trigger the external blocking of tasks (Devi, 2003; Lakshmanan and Rajkumar, 2010; Ridouard and Richard, 2006). Self-suspensions can be thought of as lowerbound temporal constraints relating tasks: for example, a user might specify that a first coat of paint needs at least 30 minutes to dry before a second coat may be applied – this 30-minute wait time is a self-suspension of the painting task.

Prior work has computed the uniprocessor schedulability of a task set with single (Liu and Anderson, 2010; Ridouard and Richard, 2006) or multiple (Gombolay and Shah, 2012, 2015) self-suspensions. In our work, we compute the multiprocessor schedulability of a task set in which multiple tasks have more than one suspension and each subtask has a resource constraint. Our approach incorporates a scheduling policy that partially restricts the behavior of the scheduler in order to reduce incidence of multiprocessor schedule anomalies due to self-suspensions. Our approach is similar in spirit to prior art that restricted behavior to reduce anomalies that inherently arise from application of uniprocessor scheduling methods to self-suspending task sets (Gombolay and Shah, 2012, 2015; Rajkumar, 1991; Lakshmanan et al., 2010; Ridouard and Richard, 2006). We first introduce our task model in this section. Second, we describe how our task sequencer satisfies temporospatial constraints in Section 8.

7.1. Traditional Self-Suspending Task Model

The Tercio Task Sequencer relies upon a well-formed task model that captures hierarchical and precedence structure within the task network. The basis for our framework is the self-suspending task model, described in Equation 17:

$$\tau_i : (W_{\langle \tau_i^0, \tau_i^1 \rangle}, C_i^1, W_{\langle \tau_i^1, \tau_i^2 \rangle}, C_i^2, \ldots, C_i^m, T_i, D_i) \quad (17)$$

In this model, a set of tasks, $\boldsymbol{\tau}$, must be processed by the computer. An instance of each task τ_i is *released* (eligible to execute) at every period T_i. The execution of the first subtask of τ_i may be delayed from the epoch start as specified by a lowerbound wait constraint, $W_{\langle \tau_i^0, \tau_i^1 \rangle}$, called the "phase offset." For each task, there are m_i subtasks, with $m_i - 1$ self-suspension intervals for each task $\tau_i \in \boldsymbol{\tau}$. We use τ_i^j to denote the j^{th} subtask of τ_i. C_i^j is the expected duration (cost) of τ_i^j. A subtask τ_i^k is released once its predecessor subtask and preceding self-suspension have both finished executing. $W_{\langle \tau_i^{j-1}, \tau_i^j \rangle}$ is the lowerbound wait constraint (or self-suspension) interval relating the end of τ_i^{j-1} and the start of τ_i^j. T_i and D_i are the period and deadline of τ_i, respectively.

7.2. Augmented Self-Suspending Task Model

The standard self-suspending task model provides a solid basis for describing many real-world processor scheduling problems of interest (Devi, 2003; Lakshmanan and Rajkumar, 2010; Ridouard and Richard, 2006). Scheduling problems within the manufacturing domain inherently have strong, hierarchical structures that are captured well by the traditional self-suspending task model. However, self-suspending task systems generally assume that processors are homogeneous, and do not include more general temporal or resource constraints among tasks and subtasks (Liu and Anderson, 2010).

$$\tau_i : (W_{\langle \tau_i^0, \tau_i^1 \rangle}, (C^a_{\tau_i^1}, R_i^1), W_{\langle \tau_i^1, \tau_i^2 \rangle}, (C^a_{\tau_i^2}, R_i^2), \ldots (C^a_{\tau_i^m}, R_i^m), T_i, D_i^{s2s}, D_i^{abs}); \{A^a_{\tau_i^j}\}, \{W_{\langle \tau_i^j, \tau_x^y \rangle}\} \quad (18)$$

We first augmented the model to encode the assignment of processors (i.e., agents) to subtasks, denoted by the set of binary decision variables, $\{A^a_{\tau_i^j}\}$. The subtask cost $C^a_{\tau_i^j}$ is now dependent upon the capabilities of the agent a processing the subtask. The second augmentation enables a user to specify subtask-to-subtask deadlines, $D^{s2s}_{\langle \tau_i^j, \tau_i^k \rangle}$, between the start of τ_i^j and end of τ_i^k (Equation 19); as well as absolute deadlines, $D^{abs}_{\tau_i^j}$, for the absolute finish times of subtasks (Equation 20), as shown in Equations 19 and 20. Subtask-to-subtask deadlines are restricted to constrain subtasks within the same task, enabling fast edge-checking in our schedulability test (Section 8). In Section 7.3, we describe how general upperbound deadline constraints relating pairs of subtasks are reformulated into this augmented self-suspending task model.

$$D^{s2s}_{\langle \tau_i^j, \tau_i^k \rangle} \geq f_i^k - s_i^j \quad (19)$$

$$D^{abs}_{\tau_i^j} \geq f_i^j \quad (20)$$

We define d_i^j as the implicit deadline on the finish time f_i^j of subtask τ_i^j, which is implied by the absolute and subtask deadlines $\boldsymbol{D^{abs}} \cup \boldsymbol{D^{s2s}}$. These deadline constraints provide additional expressiveness to encode binary temporal constraints relating tasks. For instance, these constraints may be used to specify that a sequence of subtasks involving sealant application must be completed within 30 minutes of opening the sealant container. These types of constraints are commonly included in AI and operations research scheduling models, and are vital for modeling many real-world problems (Dechter

et al., 1991; Muscettola et al., 1998).

We augmented the model to express subtask-to-subtask wait constraints (Equation 21) provided that the subtasks meet certain criteria. To describe this restriction, we first introduce two categories of subtasks: *free* and *embedded*.

Definition 1. *A free subtask, $\tau_i^j \in \boldsymbol{\tau_{free}}$, is a subtask that does not share a deadline constraint with τ_i^{j-1}. In other words, a subtask τ_i^j is free* iff *for any deadline $D_{\langle \tau_i^a, \tau_i^b \rangle}$ associated with that task, $(j \leq a) \vee (b < j)$. We define τ_i^1 as free, since a preceding subtask does not exist.*

Definition 2. *An embedded subtask, $\tau_i^{j+1} \in \boldsymbol{\tau_{embedded}}$, is a subtask that shares a deadline constraint with τ_i^j (i.e. $\tau_i^{j+1} \notin \boldsymbol{\tau_{free}}$). $\boldsymbol{\tau_{free}} \cap \boldsymbol{\tau_{embedded}} = \emptyset$.*

$$W_{\langle \tau_i^j, \tau_x^y \rangle} \leq s_x^y - f_i^j \tag{21}$$

For our augmented self-suspending task model, a wait constraint can be applied to subtasks across different tasks $\langle \tau_i^j, \tau_x^y \rangle$ if and only if the τ_x^y is a *free* subtask. This restriction also enables fast edge-checking in our schedulability test, presented in Section 8. We address how general lowerbound wait constraints relating pairs of subtasks are reformulated into this model in Section 7.3.

We also extended the model to include shared memory resources. Each subtask τ_i^j requires that a set of shared memory resources, $R_i^j = \{R_{i,1}^j, \ldots, R_{i,k_i}^j\}$, be utilized to perform that subtask (e.g., for memory shared among multiple processors). In a manufacturing setting, a shared memory resource corresponds to a region of space on the factory floor that must be physically unoccupied in order for an agent to execute a subtask there. Shared memory resources encode hard spatial constraints that prohibit agents from working in dangerously close physical proximity to one another.

We made four simplifying assumptions about this augmented self-suspending task model. First, we set the implicit deadlines of the tasks equal to the period of the task set. This modification accurately models many assembly line manufacturing processes wherein the set of tasks at each location are repeated once every "pulse" (i.e., period, H) of the production line. In this scenario, the user allots a certain amount of time, T, for the set of tasks to be accomplished, and the set of tasks is repeated with a period of T.

We also assume that all subtasks are non-preemptable, meaning that the interruption of a subtask significantly degrades its quality, and that the cost of a subtask τ_i^j assigned to agent a is determined by the expected cost of agent a executing subtask τ_i^j. Finally, we assume that switching times between tasks (i.e., travel times on the factory floor) are small compared with subtask times and can be modeled as constant. We find this assumption suitable for many assembly manufacturing applications – for example, situations in which work is performed within a localized area and does not require lengthy traversals across the factory floor, or where calibration and set-up times at new work locations are long compared with travel times. This assumption would not be reasonable for applications involving the optimization of vehicle routes across a factory or satisfaction of more-complex geometric constraints. We leave the study of more general geometric and temporal logic constraints, such as those considered in Erdem et al. (2012a) and Plaku (2012), for future work.

SEQUENCER-REFORMULATE($\boldsymbol{\tau}$)

while $\exists D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$ s.t. there is no line of precedence from τ_x^y to τ_i^j **do**
  $D^{abs}_{\tau_x^y} \leftarrow D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle} - APSP_{\langle \tau_x^y, \tau_0^0 \rangle}$
  Replace $D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$ with $D^{abs}_{\tau_x^y}$
end while
while $\exists \tau_i^j$ with constrained start time and two predecessors, **do**
  for counter $= 2$ to number of τ_i^j's predecessors **do**
    $\tau_a^b \leftarrow \tau_i^j$'s (counter$-1)^{th}$ predecessor
    $\tau_x^y \leftarrow \tau_i^j$'s (counter)th predecessor
    feasible1 $\leftarrow$ whether it is feasible for τ_a^b to precede τ_x^y
    feasible2 $\leftarrow$ whether it is feasible for τ_x^y to precede τ_a^b
    if feasible1 $\wedge$ **feasible2 then**
      if $|APSP_{\langle \tau_a^b, \tau_i^j \rangle}| \leq |APSP_{\langle \tau_x^y, \tau_i^j \rangle}|$ **then**
        Replace $W_{\langle \tau_a^b, \tau_i^j \rangle}$ with $W_{\langle \tau_a^b, \tau_x^y \rangle}$ (Eq 22)
      else
        Replace $W_{\langle \tau_x^y, \tau_i^j \rangle}$ with $W_{\langle \tau_x^y, \tau_a^b \rangle}$ (Eq 22)
      end if
    else if feasible1 then
      Replace $W_{\langle \tau_a^b, \tau_i^j \rangle}$ with $W_{\langle \tau_a^b, \tau_x^y \rangle}$ (Eq 22)
    else if feasible2 then
      Replace $W_{\langle \tau_x^y, \tau_i^j \rangle}$ with $W_{\langle \tau_x^y, \tau_a^b \rangle}$ (Eq 22)
    end if
    if feasible1 $\vee$ **feasible2 then**
      continue
    else if counter = # predecessor subtasks **then return null**
    end if
  end for
end while
return $\boldsymbol{\tau}$

Fig. 6: Pseudocode for the Reformulation Subroutine.

7.3. Reformulating the Task Set

Tercio's Task Sequencer uses an online schedulability test to determine whether a given subtask τ_i^j can be scheduled at time t considering the deadline constraints within the task set. This schedulability test requires the specific structure of the augmented self-suspending task model in order for this inference to be efficient. In this section, we present our method for reformulating general upperbound deadlines, $D^{s2s}_{\langle \tau_a^b, \tau_i^j \rangle}$, and lowerbound wait constraints, $W_{\langle \tau_a^b, \tau_x^y \rangle}$, into this more-structured form.

The reformulation ensures that for every absolute deadline constraint $D^{abs}_{\tau_i^j}$ or subtask-to-subtask deadline constraint $D^{s2s}_{\langle \tau_a^b, \tau_i^j \rangle}$, subtask τ_i^j has only one predecessor subtask τ_x^y. This means that either $\tau_x^y = \tau_i^{j-1}$, or that there exists a self-suspension $W_{\langle \tau_x^y, \tau_i^j \rangle}$. In turn, τ_x^y must have exactly one such predecessor subtask. This process recurses until reaching either the epoch subtask τ_0^0, in the case of an absolute deadline constraint, or τ_a^b, in the case of a subtask-to-subtask deadline. The reformulation preserves correctness, meaning that any schedule that satisfies the constraints of the reformulated task set will also satisfy the constraints of the original task set. The reformulation rules are based on the *triangle rules* and

proofs of correctness developed by Tsamardinos *et al.* for the reformulation of temporal networks (Tsamardinos et al., 1998). However, note that the reformulation process for our algorithm is incomplete: it heuristically attempts to solve an NP-hard scheduling problem, does not explore the entire search space, and thus does not always return a sequence if one exists. The ability to reformulate depends upon the laxity of the deadlines in the task set and the structure of the constraints. The tighter the deadlines and the more connected the constraint graph, the less likely it is that the algorithm will be able to reformulate the problem into the structure our schedulability test requires. Figure 6 depicts **SEQUENCER-REFORMULATE**, which takes a task set as input and returns either a reformulated task set (if the algorithm can find a solution) or null (if no feasible reformulated task set can be identified).

7.4. Reformulation Pseudo-code

Line 1 checks whether any subtask-to-subtask deadlines $D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$ exist such that there is neither a wait constraint $W_{\langle \tau_i^j, \tau_x^y \rangle}$ nor a set of wait constraints linking τ_i^j and τ_x^y via one or more intermediary subtasks (e.g., $W_{\langle \tau_i^j, \tau_a^b \rangle}$ and $W_{\langle \tau_a^b, \tau_x^y \rangle}$). If there exists such a deadline $D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$, it is replaced with a new absolute deadline, $D^{abs}_{\tau_x^y}$, that implicitly enforces $D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$ (Line 2). Note in Line 2, $APSP_{\langle \tau_i^j, \tau_x^y \rangle}$ is the APSP temporal upperbound between τ_i^j and τ_x^y, and that the APSP temporal lowerbound between ordered tasks is non-positive.

For each subtask with its initiation constrained by a deadline and which involves more than one predecessor subtask, the algorithm iterates over the predecessor subtasks (Lines 5-6). Lines 7-8 consider two predecessors at a time, τ_a^b and τ_x^y, and determine whether the lowerbound constraints can be reformulated such that τ_a^b is the predecessor of τ_x^y, or vice-versa (Lines 9-10). For a subtask τ_i^j with two predecessors, τ_a^b and τ_x^y, the algorithm restructures the problem such that τ_a^b is the predecessor of τ_x^y by replacing $W_{\langle \tau_x^y, \tau_i^j \rangle}$ with $W_{\langle \tau_a^b, \tau_x^y \rangle}$, according to Equation 22:

$$W_{\langle \tau_a^b, \tau_x^y \rangle} = \max \left(W_{\langle \tau_a^b, \tau_i^j \rangle} - C_{\tau_x^y} - W_{\langle \tau_x^y, \tau_i^j \rangle}, W_{\langle \tau_a^b, \tau_x^y \rangle} \right) \quad (22)$$

In Line 11, in the event that both methods of reformulating the problem are feasible, the algorithm employs a "tie-breaker": If the temporal distance path $\tau_i^j \rightarrow \tau_x^y \rightarrow \tau_a^b$ is shorter than that between $\tau_i^j \rightarrow \tau_a^b \rightarrow \tau_x^y$, the path from τ_i^j to τ_x^y is removed and a temporal wait constraint is added from τ_a^b to τ_x^y – as a result, τ_x^y is now the predecessor of τ_a^b. If only one method of reformulation is feasible (Lines 17 and 19), the constraints are modified accordingly (Lines 18 and 20). Each time the constraint network is modified, the reformulation process restarts (Lines 22-23). The algorithm returns null if it is impossible to reorder τ_a^b and τ_x^y and if there are no more predecessor subtasks of τ_i^j to process (Line 24); otherwise, the algorithm returns the reformulated task set (Line 28).

7.5. Reformulation Example

In this section, we provide an example to illustrate the reformulation process. Figures 7 and 8 depict a task set

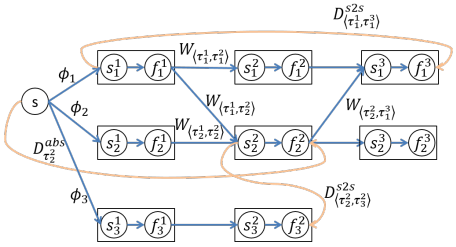

Fig. 7: This figure depicts a task set that must be reformulated to adhere to the augmented self-suspending task model.

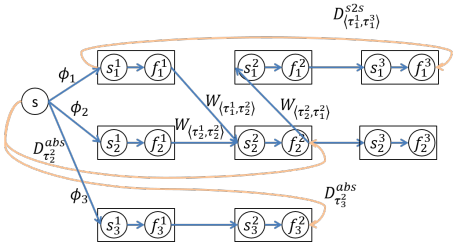

Fig. 8: This figure depicts the first two steps for the reformulation of the task set in Figure 7.

before and after reformulation, respectively. In these figures, subtask start and end times are denoted as nodes (black circles), and constraints are represented by edges. Blue edges indicate lowerbound temporal constraints (phase offsets, self-suspensions, or wait constraints), and orange edges represent upperbound temporal constraints (absolute or subtask-to-subtask deadline constraints). Recall that a task set must satisfy two conditions in order to be correctly sequenced by Tercio: First, for every subtask-to-subtask deadline $D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$, there must be a path of precedence from τ_i^j to τ_x^y. Second, each subtask must have exactly one predecessor subtask.

Consider the task set depicted in Figure 7. This example includes three temporal constraints that conflict with the above requirements: $D^{s2s}_{\langle \tau_2^2, \tau_3^2 \rangle}$ does not satisfy the requirement of a directed path of precedence from τ_2^2 to τ_3^2. Also, embedded subtask τ_2^2 has more than one precedence constraint delaying its initiation: ($W_{\langle \tau_1^1, \tau_2^2 \rangle}$ and $W_{\langle \tau_2^1, \tau_2^2 \rangle}$).

The first step to addressing these temporal constraints is to reformulate $D^{s2s}_{\langle \tau_2^2, \tau_3^2 \rangle}$ as an absolute deadline $D^{abs}_{\tau_3^2}$, as shown in Figure 8, such that $D^{abs}_{\tau_3^2}$ is sufficiently tight to guarantee that f_3^2 does not occur later than $s_2^2 + D^{s2s}_{\langle \tau_2^2, \tau_3^2 \rangle}$. To ensure consistency, the reformulation subroutine sets $D^{abs}_{\tau_3^2}$ equal to $-APSP_{\langle \tau_2^2, \tau_0^0 \rangle} + D^{s2s}_{\langle \tau_2^2, \tau_3^2 \rangle}$. The requirement of a path of precedence from τ_i^j to τ_x^y for every subtask-to-subtask deadline $D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$ is now satisfied.

Next, we address the wait constraints, $W_{\langle \tau_1^1, \tau_2^2 \rangle}$ and $W_{\langle \tau_2^1, \tau_2^2 \rangle}$. In order to do so, we must first determine whether to require for τ_2^1 to precede τ_1^1, or vice-versa. If neither option is possible, the reformulation algorithm cannot reformulate the task set. If only one option is possible, the algorithm selects and enforces that precedence constraint. If both options are temporally feasible, the algorithm assigns the ordering to reduce the delay of τ_2^2. In this example, ordering τ_2^1

before τ_1^1 is preferred, so the algorithm replaces $W_{\langle \tau_2^1, \tau_2^2 \rangle}$ with $W_{\langle \tau_2^1, \tau_1^1 \rangle} = \max\left(W_{\langle \tau_2^1, \tau_2^2 \rangle} - \left(W_{\langle \tau_1^1, \tau_2^2 \rangle} C_1^2\right), 0\right)$ to ensure satisfaction of the original constraint. The task set can now be correctly sequenced by the Tercio Task Sequencer algorithm.

7.6. Proofs of Termination and Correct Reformulation

In this section, we show that the reformulation algorithm (Figure 6) will terminate for finite task sets. We also demonstrate that the algorithm preserves correctness – that any schedule that satisfies the constraints of the reformulated task set will also satisfy the constraints of the original task set.

Theorem 7.1. *The Reformulation Subroutine preserves correctness, meaning that any schedule that satisfies the constraints of the reformulated task set will also satisfy the constraints of the original task set.*

Proof by Deduction: Here, we show that each lowerbound temporal constraint removed in Lines 13, 15, 18, and 20 is made redundant by the newly added temporal constraint. Consider an embedded subtask τ_i^j, with predecessors τ_a^b and τ_x^y, ordered as follows:

$$s_i^j \geq f_x^y + W_{\langle \tau_x^y, \tau_i^j \rangle} \tag{23}$$

and

$$s_i^j \geq f_a^b + W_{\langle \tau_a^b, \tau_i^j \rangle} \tag{24}$$

The reformulation process either orders τ_x^y before τ_a^b, or vice-versa. If ordering τ_x^y before τ_a^b is temporally feasible (Johnson, 1954; Floyd, 1962), the algorithm removes constraint $W_{\langle \tau_x^y, \tau_i^j \rangle}$ and adds constraint $W_{\langle \tau_x^y, \tau_a^b \rangle}$ according to Equation 22. (Note that the proof for ordering τ_a^b before τ_x^y is symmetric.) When $W_{\langle \tau_a^b, \tau_i^j \rangle}$ in Equation 22 is sufficiently large to contribute to the new wait constraint $W_{\langle \tau_x^y, \tau_a^b \rangle}$, Equation 24 and Equation 22 recover Equation 23:

$$\begin{aligned} s_i^j &\geq W_{\langle \tau_a^b, \tau_i^j \rangle} + f_a^b \geq W_{\langle \tau_a^b, \tau_i^j \rangle} + C_{\tau_a^b} + s_a^b \\ &\geq W_{\langle \tau_a^b, \tau_i^j \rangle} + C_{\tau_a^b} + W_{\langle \tau_x^y, \tau_a^b \rangle} + f_x^y \\ &\geq W_{\langle \tau_a^b, \tau_i^j \rangle} + C_{\tau_a^b} + W_{\langle \tau_x^y, \tau_i^j \rangle} - \left(C_{\tau_a^b} + W_{\langle \tau_a^b, \tau_i^j \rangle}\right) + f_x^y \\ &\geq f_x^y + W_{\langle \tau_x^y, \tau_i^j \rangle}. \end{aligned} \tag{25}$$

If there already exists a constraint $W_{\langle \tau_x^y, \tau_a^b \rangle}$ and $W_{\langle \tau_x^y, \tau_i^j \rangle} - \left(C_{\tau_a^b} + W_{\langle \tau_a^b, \tau_i^j \rangle}\right) \leq W_{\langle \tau_x^y, \tau_a^b \rangle}$, then ordering τ_x^y before τ_a^b inherently satisfies Equation 23. Likewise, if $W_{\langle \tau_x^y, \tau_a^b \rangle}$ does not exist and $W_{\langle \tau_x^y, \tau_i^j \rangle} - \left(C_{\tau_a^b} + W_{\langle \tau_a^b, \tau_i^j \rangle}\right) \leq 0$, then ordering τ_x^y before τ_a^b inherently satisfies Equation 23. Equation 24 remains satisfied because $W_{\langle \tau_a^b, \tau_i^j \rangle}$ remains unaltered. ■

Theorem 7.2. *The Reformulation Subroutine terminates.*

Proof by Deduction: Lines 1-4 process a finite number of subtask-to-subtask deadlines, replacing each with an absolute deadline. Lines 5-27 add temporal precedence constraints to the network. Each lowerbound temporal constraint removed in Lines 13, 15, 18, and 20 is redundant given the newly added temporal constraint, meaning that any schedule that satisfies the new temporal constraint also satisfies the removed constraint (see Theorem 7.1). Therefore, the size of the set of feasible schedules only decreases with each reformulation step, and the algorithm is ultimately guaranteed to terminate. ■

7.7. *SEQUENCER-REFORMULATE* Complexity Analysis

Here, we present the computational complexity of **SEQUENCER-REFORMULATE**. Lines 1-4 of **SEQUENCER-REFORMULATE** convert a subset of the subtask-to-subtask deadlines into absolute deadlines. This involves a maximum of $|\boldsymbol{\tau}|^2$ operations. Lines 5-27 reformulate the task graph to ensure that all subtasks τ_i^j with constrained start times have no more than one predecessor. At most, $|\boldsymbol{\tau}|^2$ wait constraints are modified, and one wait constraint is removed at each iteration (Lines 13, 15, 18, 20). The feasibility of each modification is assessed using Johnson's Algorithm (Johnson, 1954), which has a complexity of $O\left(|\boldsymbol{\tau}|^2 \log|\boldsymbol{\tau}| + |\boldsymbol{\tau}||TC|\right)$. Thus, the complexity of the reformulation algorithm is $O\left(|\boldsymbol{\tau}|^2 + |\boldsymbol{\tau}|^2\left(|\boldsymbol{\tau}|^2 \log|\boldsymbol{\tau}| + |\boldsymbol{\tau}||TC|\right)\right)$.

8. Multi-Agent Online Consistency Check

The Tercio Task Sequencer presented in Figure 5 uses a schedulability test to ensure feasibility while scheduling tasks against deadline constraints and shared memory (or spatial) resource constraints. Our schedulability test, the Multiprocessor Russian Dolls Test, works by determining whether the execution of one set of subtasks associated with a single deadline constraint can be nested within the laxity of a second set of subtasks associated with another deadline constraint. The test operates on the augmented self-suspending task model introduced in Section 7.2, and makes the explicit assumption that any subtask with a start time tightened by a deadline constraint will be released upon completion of its single predecessor and any wait or self-suspension time. This assumption enables the algorithm to compute an empirically tight schedulability test in polynomial time.

Our test is a variant of a resource edge-finding algorithm (Laborie, 2003; Vilím et al., 2005), the purpose of which is to determine whether an event must or may execute before or after a set of activities (Baptiste and Pape, 2000). To our knowledge, our approach is the first to leverage the structure of the self-suspending task model to perform fast edge-checking.

We incorporate the test in two ways: First, we use it to ensure that the sequencing algorithm can identify a feasible schedule against $\boldsymbol{D^{abs}}$ upon initiation of the scheduling simulation. Second, we use the test to ensure that scheduling a subtask τ_i^j at time t will not result in plan infeasibility due to $\boldsymbol{D^{abs}} \cup \boldsymbol{D^{s2s}}$.

To describe our test, we first define an *active deadline* (Definition 3) and an *active subtask* (Definition 4).

Definition 3. Active Deadline*: A subtask-to-subtask deadline* $D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$ *is considered* active *for* $t \in [s_i^j, f_x^y)$*, and an absolute deadline* $D^{abs}_{\tau_i^j}$ *is considered* active *for* $t \in [0, f_i^j)$.

Definition 4. Active Subtask*: A subtask is* active *at time* t *if it is unexecuted at time* t *and is directly constrained by an active deadline.*

8.1. Pseudocode Walkthrough

Figure 9 depicts the pseudocode for the Multiprocessor Russian Dolls Test. The test takes as input a subtask τ_i^j, the task set $\boldsymbol{\tau}$, the time t, and the *type* of test, and returns either the feasibility of the set of absolute deadlines (type $= 1$) or the feasibility of scheduling τ_i^j at time t (type $= 2$).

Our test operates on a specific problem structure: For every absolute deadline constraint $D^{abs}_{\tau_i^j}$ or subtask-to-subtask deadline constraint $D^{s2s}_{\langle \tau_a^b, \tau_i^j \rangle}$, subtask τ_i^j must have only one predecessor subtask τ_x^y. This means that either $\tau_x^y = \tau_i^{j-1}$, or else there exists a self-suspension $W_{\langle \tau_x^y, \tau_i^j \rangle}$. In turn, each τ_x^y must have exactly one such predecessor subtask. This recurses until reaching either the epoch subtask τ_0^0, in the case of an absolute deadline constraint, or τ_a^b, in the case of a subtask-to-subtask deadline. This structure allows us to assume that a subtask constrained by a deadline will be released as soon as its predecessor subtask has executed and the wait constraint has expired. In Section 7.3, we describe the mechanism for reformulating a general task set into this structure.

Before sequencing the task, the algorithm tests the feasibility of the absolute deadlines by calling the Russian Dolls Test with $\tau_i^j = \tau_0^0$ (the epoch), the task set $\boldsymbol{\tau}$, time $t = 0$, and type $= 1$ as input. The test determines whether the set of subtasks constrained by each active deadline can feasibly be executed within the laxity of the other active deadlines. Recall that initially, when $t = 0$, only the absolute deadlines are active. The algorithm collects the set of active tasks (Line 2) and iterates over all pairs of unique subtasks constrained by absolute deadlines (Lines 14-15). If a pair of subtasks, τ_i^k and τ_x^y, is allocated to the same agent (i.e., processor) or requires the same resource, there is potential for contention between these subtasks. Line 17 checks whether the execution of one subtask τ_i^k, which must occur before d_i^k, can be nested within the laxity, $d_x^y - C_{\tau_i^k}$, of the deadline of subtask τ_x^y, or vice-versa. This nesting is possible if proposition $\left(d_i^k \leq d_x^y - C_x^y\right) \vee \left(d_i^k - C_i^k \geq d_x^y\right)$ holds; if no such nesting is possible (i.e., if $\left(d_i^k > d_x^y - C_x^y\right) \wedge \left(d_i^k - C_i^k < d_x^y\right)$), the system returns that it cannot successfully schedule against the set of absolute deadline constraints (Line 17). If the system can perform this nesting for all such pairs of subtasks, it returns that it can successfully schedule τ_i^j at time t (Line 22).

If the task sequencer calls the Russian Dolls Test to determine the feasibility of scheduling τ_i^j at time t (if type $= 2$), it must first consider the direct effect of executing τ_i^j. The algorithm iterates over all active subtasks (Definition 4) not including τ_i^j (Line 4). If another active subtask τ_x^y is assigned to the same agent or requires the same resource as τ_i^j (Line 5), the test determines whether it is possible to nest the execution of τ_i^j within the laxity of τ_x^y's deadline (Line 6). If such nesting is impossible, the algorithm returns *false*, prohibiting the scheduling of τ_i^j at time t (Line 13). Otherwise, if scheduling τ_i^j at time t would activate a subtask-to-subtask deadline $D^{s2s}_{\langle \tau_i^j, \tau_i^z \rangle}$ (Definition 3), the algorithm must consider the indirect effects of activation on the other subtasks influenced by this deadline constraint. If such activation would occur, the test stores the set of those subtasks explicitly

multiprocessorRussianDollsTest(τ_i^j,$\boldsymbol{\tau}$,t,type)

1: **if** type $= 1$ **then**
2: $\quad \boldsymbol{\tau_{group}} \leftarrow \boldsymbol{\tau_{active}}$
3: **else if** (type $= 2$) $\wedge \left(\exists D^{s2s}_{\langle \tau_i^j, \tau_i^z \rangle}\right)$ **then**
4: $\quad$ **for** all $\tau_x^y \in \boldsymbol{\tau_{active}} \backslash \tau_i^j$ **do**
5: $\quad\quad$ **if** τ_x^y and τ_i^j use the same agent or resource **then**
6: $\quad\quad\quad$ **if** $t + C_i^j > d_x^y - C_x^y$ **then return** false
7: $\quad\quad\quad$ **end if**
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: $\quad \boldsymbol{\tau_{group}} \leftarrow \{\tau_i^k | j < k \leq z, \forall D^{s2s}_{\langle \tau_i^j, \tau_i^z \rangle}\}$
11: **else**
12: $\quad \boldsymbol{\tau_{group}} \leftarrow \emptyset$
13: **end if**
14: **for** all $\tau_i^k \in \boldsymbol{\tau_{group}}$ **do**
15: $\quad$ **for** all $\tau_x^y \in \boldsymbol{\tau_{active}} \backslash \tau_i^k$ **do**
16: $\quad\quad$ **if** τ_i^k and τ_x^y use the same agent or resource **then**
17: $\quad\quad\quad$ **if** $\left(d_i^k > d_x^y - C_x^y\right) \wedge \left(d_i^k - C_i^k < d_x^y\right)$ **then return** false
18: $\quad\quad\quad$ **end if**
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end for**
22: **return** true

Fig. 9: Pseudocode describing the Multiprocessor Russian Dolls Test.

constrained by $D^{s2s}_{\langle \tau_i^j, \tau_i^z \rangle}$, excluding τ_i^j (Line 10). (Note that the algorithm has already tested the feasibility of τ_i^j.)

Next, the algorithm iterates over the subtasks in this group and over the set of active subtasks (Lines 14-15). If an active subtask τ_x^y and a subtask τ_i^k constrained by $D^{s2s}_{\langle \tau_i^j, \tau_i^z \rangle}$ are assigned to the same agent or require the same resource (Line 16), the test determines whether it is possible to nest the execution of τ_i^k within the laxity of τ_x^y's deadline, or vice-versa (Line 17). The implicit deadline d_i^k for $\tau_i^k \in \boldsymbol{\tau_{group}}$ is calculated assuming τ_i^j is executed at time t for the purposes of the test. If nesting is not feasible, the algorithm returns *false*, indicating there is no guarantee that a feasible schedule exists if τ_i^j is executed at time t (Line 17). If nesting can be performed for all such pairs of subtasks, the algorithm returns *true*, indicating that τ_i^j can be scheduled at time t (Line 22).

8.2. Proof of Correctness of Multiprocessor Russian Dolls Test

The Multiprocessor Russian Dolls Test is polynomial in time complexity as it only performs pairwise comparisons between active subtasks and the tasks in τ_{group}, and is therefore not a complete schedulability test. However, we prove in this section that the algorithm is nonetheless correct, in that only pairwise comparisons of subtasks are necessary to ensure schedule feasibility.

The test leverages the problem structure inherent in the augmented self-suspending task model in order to perform efficient computation. Recall that the task model requires every subtask constrained by an absolute $D^{abs}_{\tau_i^j}$ or subtask-to-subtask $D^{s2s}_{\langle \tau_i^j, \tau_i^k \rangle}$ deadline to have exactly one predecessor. For absolute deadline $D^{abs}_{\tau_i^j}$, the precedence recurses from τ_i^j to

τ_i^{j-1} and so on through τ_i^1. Likewise, for subtask-to-subtask deadline $D^{s2s}_{\langle \tau_i^j, \tau_i^k \rangle}$, the precedence recurses from τ_i^k to τ_i^{k-1} through τ_i^j. This problem structure restricts the interaction of deadlines in the task set.

Lemma 8.1. *Scheduling τ_i^j does not change the implicit deadlines or the laxity of any subtask τ_x^y unless $x = i$ and a subtask-to-subtask deadline $D^{s2s}_{\langle \tau_i^j, \tau_i^z \rangle}$ with $i \leq y \leq z$ exists.*

Proof: Consider a set of intra-task deadlines for τ_i. If the problem is correctly structured as an augmented self-suspending task model, then each embedded subtask in the set $\boldsymbol{\tau_{group}} = \{\tau_i^{j+1}, \ldots, \tau_i^y, \tau_i^z\}$ has one predecessor that is either a member of $\boldsymbol{\tau_{group}}$ or is τ_i^j. Thus, when τ_i^j is scheduled, implicit deadlines δ_x^y for subtasks $\tau_x^y \notin \{\tau_i^j, \tau_i^{j+1}, \ldots, \tau_i^y, \tau_i^z\}$ are unchanged. ∎

Next, we use Lemma 8.1 to demonstrate the correctness of the Russian Dolls Test:

Theorem 8.2. *The Russian Dolls Test is correct in that it requires only pairwise comparisons between active subtasks $\boldsymbol{\tau_{active}}$ and subtasks in $\boldsymbol{\tau_{group}}$.*

Proof: When τ_i^j is scheduled, the implicit deadlines for active subtasks may be tightened. We show that this tightening will not result in temporal infeasibility. As in Lemma 8.1, scheduling τ_i^j does not change the implicit deadlines or slack times of any subtask τ_x^y unless $x = i$ and a subtask-to-subtask deadline $D^{s2s}_{\langle \tau_i^j, \tau_i^z \rangle}$ with $i \leq y \leq z$ exists. In the case where $x \neq i$ or there is no $D^{s2s}_{\langle \tau_i^j, \tau_i^z \rangle}$, if it is feasible to nest an active subtask τ_x^y within the laxity of every other active subtask τ_p^q at time t_o, this schedule commitment will reduce the slack available for future scheduling commitments but will not tighten the deadlines of other active subtasks. As such, a pairwise comparison between subtasks in $\boldsymbol{\tau_{group}}$ and subtasks in $\boldsymbol{\tau_{active}}$ is sufficient to produce of a feasible schedule. ∎

According to Theorem 8.2, it is sufficient to consider all pairwise comparisons between subtasks in $\boldsymbol{\tau_{active}}$ and $\boldsymbol{\tau_{group}}$ to ensure correctness of the Russian Dolls Test. However, these comparisons are not necessary for correctness. While the computational complexity of the test is polynomial in time, as discussed in Section 10, we seek to reduce the computational complexity further by leveraging the fact that these subtasks do not all utilize the same agents and resources.

Corollary 8.3. *The Russian Dolls Test is correct in that it requires only pairwise comparisons for each $\langle \tau_x^y, \tau_i^j \rangle$ where $\tau_x^y \in \boldsymbol{\tau_{active}}$ and $\tau_i^j \in \boldsymbol{\tau_{group}}$ (as in Theorem 8.2), such that $A^a_{\tau_i^j} = A^a_{\tau_x^j}$ and $\langle \tau_x^y, \tau_i^j \rangle \in \boldsymbol{\tau_R}$.*

Proof: Scheduling any subtask in $\tau_i^j \in \boldsymbol{\tau_{group}}$ can only result in infeasibility if the required agent or resource is not available to execute τ_i^j before d_i^j because the agent or resource is occupied by some $\tau_x^y \in \boldsymbol{\tau_{active}}$. Thus, to ensure feasibility in the Russian Dolls Test, it is only necessary to consider pairwise comparisons between subtasks in $\boldsymbol{\tau_{group}}$ and $\boldsymbol{\tau_{active}}$ that share the same agent or resource. ∎

8.3. Russian Dolls Test Complexity Analysis

Here, we analyze the computational complexity of the Russian Dolls Test. The test can be called under one of two conditions, Type 1 or Type 2, depending on how the test is being used. In either case, the complexity is the same. For Type 1, at most $|\boldsymbol{\tau}|$ are added to $\boldsymbol{\tau_{group}}$. Then, in Lines 14-21, every member of $\boldsymbol{\tau_{group}}$ is compared to each member of $\boldsymbol{\tau_{active}}$ in order to determine whether each $\tau_i^k \in \boldsymbol{\tau_{group}}$ can nest within the slack of $\tau_x^y \in \boldsymbol{\tau_{active}}$, or vice-versa (Line 17). As such, the complexity of the Russian Dolls Type 1 Test is $O\left(|\boldsymbol{\tau}|^2\right)$. For Type 2, the algorithm first checks the feasibility of scheduling τ_i^j at time t in Lines 4-9. Here, the test considers pairings of τ_i^j to each subtask in $\boldsymbol{\tau_{active}}$, resulting in a maximum of $|\boldsymbol{\tau}| - 1$ operations. Next, Lines 14-21 consider pairings of each τ_i^k in $\boldsymbol{\tau_{group}}$ to each subtask in $\boldsymbol{\tau_{active}}$. There are at most $\boldsymbol{\tau} - 1$ in both $\boldsymbol{\tau_{group}}$ and $\boldsymbol{\tau_{active}}$, resulting in $|\boldsymbol{\tau}|(|\boldsymbol{\tau}| - 1)$ evaluations. Overall, the Type 2 Russian Dolls Test is of complexity $O\left(|\boldsymbol{\tau}|(|\boldsymbol{\tau}| - 1) + (|\boldsymbol{\tau}| - 1)\right) = O\left(|\boldsymbol{\tau}|^2\right)$. Thus, the complexity of the Russian Dolls Test under Type 1 or 2 is given by $O\left(|\boldsymbol{\tau}|^2\right)$.

9. Complexity of Tercio

As presented in Section 5, the computational complexity of Tercio's agent allocation subroutine is given by $O\left(2^{|A||\boldsymbol{\tau}|}\right)$. Here, we derive the complexity of Tercio's sequencing subroutine (Figure 5). Tercio's sequencing subroutine, **TERCIO-SEQUENCER**, begins in Line 1 by calling the reformulation algorithm, **SEQUENCER-REFORMULATE** (Figure 6), to transform general task sets into the structured model required for the schedulability test. The complexity of this subroutine is given by $O\left(|\boldsymbol{\tau}|^2 + |\boldsymbol{\tau}|^2\left(|\boldsymbol{\tau}|^2 \log|\boldsymbol{\tau}| + |\boldsymbol{\tau}||TC|\right)\right)$. After reformulation, the sequencing subroutine uses the Russian Dolls Test (Figure 9, Type 1 test) to ensure the feasibility of $\boldsymbol{D^{abs}}$, which has a complexity of $O(|\boldsymbol{\tau}|)$ (Section 8.3).

Next, the sequencer iterates through time to sequence all tasks (Line 1, Figure 5). The sequencing algorithm iterates $f_{max} = \max_{\tau_i^j} f_i^j$ time steps at most, assuming a unit time step, and $\frac{f_{max}}{dt}$ otherwise. At each iteration, a maximum of n subtasks (Lines 5-6) are available for scheduling with satisfied precedence constraints. The priority queue of available subtasks is sorted with $O(n\log(n))$. Line 7 of the sequencer (Figure 5) then evaluates the feasibility of scheduling each subtask using the Russian Dolls Type 2 Test, the complexity of which is given by $O(|\boldsymbol{\tau}|)$ (Section 8.3). Finally, the sequencer performs an APSP computation (Line 8, Figure 5) for each subtask τ_i^j with a deadline constraint $D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$. This involves $O\left(|\boldsymbol{\tau}|^2 \log|\boldsymbol{\tau}| + |\boldsymbol{\tau}||TC|\right)$ operations. Thus, the sequencer's complexity is of $O\left(\text{Sequencer}\right) = O\left(|\boldsymbol{\tau}|^2 + |\tau^2| + \left(|\boldsymbol{\tau}|^2 + \ |\boldsymbol{\tau}|^2 n f_{max} n \log(n)\right)\left(|\boldsymbol{\tau}|^2 \log|\boldsymbol{\tau}| + |\boldsymbol{\tau}||TC|\right)\right) = O\left(\left(f_{max} n^2 \log(n)\right)\left(|\boldsymbol{\tau}|^4 \log|\boldsymbol{\tau}| + |\boldsymbol{\tau}|^3 |TC|\right)\right)$.

Given the above, Tercio – including task allocation, the sequencer, and all sequencing subroutines – is of complexity $O\left(2^{|\boldsymbol{A}||\boldsymbol{\tau}|}\left(f_{max} n^2 \log(n)\right)\left(|\boldsymbol{\tau}|^4 \log|\boldsymbol{\tau}| + |\boldsymbol{\tau}|^3 |TC|\right)\right)$.

10. Evaluation and Discussion

In this section, we empirically validate that Tercio quickly produces solutions within 10% of optimal for multi-agent

scheduling problems with temporospatial constraints. Results were generated using an off-the-shelf laptop with an Intel Core i7-2820QM CPU 3.20GHz (8 Cores) and 16 GB of RAM.

Tercio is composed of a Task Reformulator (Section 7.3, Figure 6), Task Allocator (Section 5) and Task (Section 6, Figure 5). The Reformulator is implemented in Java, the Task Allocation algorithm is solved using a Java interface to a third-party optimizer (Gurobi), and the Task Sequencer is in MATLAB. Tercio is an iterative algorithm; for this evaluation, we limited the number of iterations to 25. We found that solutions returned by Tercio did not improve significantly with additional iterations for the problems we were able to benchmark against the optimal solution. For agent allocation, we set $\alpha_1 = 2$ and $\alpha_2 = 1$, as this generally achieves helpful allocations for the sequencer, and terminated Gurobi once the incumbent solution was found to be within 0.1% of optimal. We compared our approach against three benchmarks: 1) TeSSI (Nunes and Gini, 2015), 2) OCGA (Zhang and Wong, 2015), and 3) an exact, MILP-based solution method. For all algorithms, a 60-minute timeout was applied.

We compared our approach to a technique based on TeSSI and its variants (Nunes and Gini, 2015; McIntire et al., 2016). At its core, TeSSI is an insertion heuristic. As discussed in (Gendreau et al., 1998), insertion heuristics function by deciding which subtask should next be inserted, where, and to which agent based on some prescribed criteria. TeSSI's criterion is the makespan of the agent to which the subtask is assigned. TeSSI operates on a more-restricted problem structure that involves neither upper- and lowerbound temporal constraints nor resource constraints. Thus, to compute each agent's makespan, TeSSI simply adds the duration of each task and the time spent traveling between tasks, which is linear in the number of subtasks assigned to the agent. In contrast, Tercio considers problems with upper- and lowerbound temporal constraints and resource constraints. In order to correctly schedule against upperbound temporal constraints, it is necessary to employ a temporal consistency check (e.g., Tercio's Russian Dolls Test) as commitments are made. A directed path consistency algorithm (Dechter et al., 1991) may also serve this purpose as an alternative to the Russian Dolls Test. For our comparison, we used Snowball (Planken et al., 2012), a state-of-the-art algorithm that achieves complexity of $O(n^2)$ for certain cases, such as graphs of constant tree width. We refer to this APSP-variant of TeSSI as TeSSI*.

OCGA is a state-of-the-art genetic algorithm developed for job shop scheduling problems (Zhang and Wong, 2015). We adopted the parameter prescribed by Zhang and Wong (2015) for OCGA as follows: population size N = 100, pairs of parents replicating per iteration $r = 7$, probability for crossover $p_c = 1$, probability for shifting genes' loci $p_e = 1$, probability for re-allocating agent for a given subtask p_m = 0.06, degeneration ratio $R = 0.09$, and number of iterations $N = 2,500$. As with TeSSI*, it was necessary to add a directed path consistency algorithm to evaluate the quality of each schedule; we also used Snowball for this purpose (Planken et al., 2012). In order to improve runtime, we used a hashing function, which stored the quality of previously computed schedules to reduce the number of calls to Snowball with successive iterations of OCGA. Genetic algorithms such as OCGA are generally probabilistically guaranteed to find the optimal solution as the number of iterations approaches infinity. Thus, given enough time, OCGA will identify a solution better than that generated by Tercio. We find that a more-helpful benchmark than comparing solution quality between OCGA and Tercio is to measure how much time is required for OCGA to find a solution comparable to or better than that identified by Tercio. Thus, in our results, we report the computation time required for OCGA to find such a solution. For instances in which Tercio cannot identify a solution, we report the total computation time required for all 2,500 iterations, as prescribed by Zhang and Wong (2015).

The exact, MILP-based formulation, corresponding to solving the MILP defined in Section 3 and objective function in Equation 13, was solved by calling Gurobi using its default settings, which sets the optimality threshold to 0.1%.

10.1. Generating Random Problems

We evaluated the performance of Tercio when applied to randomly generated problems. The lowerbound agent-task times were of the form $lb^a_{\tau_i^j}$ and were drawn from a uniform distribution over the interval $[1, 10]$. Expected agent-task times $C^a_{\tau_i^j}$ were generated from $C^a_{\tau_i^j} \sim U[lb^a_{\tau_i^j}, 10]$. We did not explicitly constrain the maximum amount of time an agent spent performing a subtask. Approximately 25% of the generated precedence constraints $W_{\langle \tau_x^{y-1}, \tau_x^y \rangle}$ were wait constraints with non-zero lowerbounds drawn from the interval $[1, 10]$. Approximately 25% of subtasks had a wait constraint, $W_{\langle \tau_i^j, \tau_x^y \rangle}$, in addition to their self-suspension constraint, $W_{\langle \tau_x^{y-1}, \tau_x^y \rangle}$.

In order to generate subtask-to-subtask deadlines, we randomly sampled two subtasks, τ_i^j and τ_x^y, such that $y \geq j$, and added a deadline constraint, $D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}$. To add absolute deadlines, we randomly sampled a subtask, τ_i^j, and added a deadline constraint, $D^{abs}_{\tau_i^j}$. We sought to generate problems of sufficient challenge for our validation. We established a metric, $\hat{D} = \sum_{D^{s2s}_{\langle \tau_i^j, \tau_x^y \rangle}} (y - j + 1) + \sum_{D^{abs}_{\tau_i^j}} j$, and set it to $\frac{1}{4}$ for our empirical evaluation depicted by Figure 10a-10l. The upperbound of each deadline constraint was drawn from a uniform distribution, with the lowerbound set to the tightest feasible deadline and the upperbound set to the sum over subtask costs and wait constraint times. The 1-D physical locations of subtasks were drawn from a uniform distribution $[1, |\boldsymbol{\tau}|]$. The number of subtasks m_i within each task τ_i was drawn from a uniform distribution in the interval $[1, 2n]$, where n is the number of tasks τ_i in $\boldsymbol{\tau}$. While an agent performed subtask τ_i^j at resource location (x), no other agent could work on a subtask at location (u) if $x - 1 \leq u \leq x + 1$.

10.2. Computation Speed and Scalability

Figures 10a, 10e, and 10i depict our evaluation of Tercios scalability and computational speed. We present the median and upper/lower quartiles of the computation time for 50 randomly generated problems incorporating 5, 10, or 100 agents (Figures 10a, 10e, and 10i, respectively) and between 4

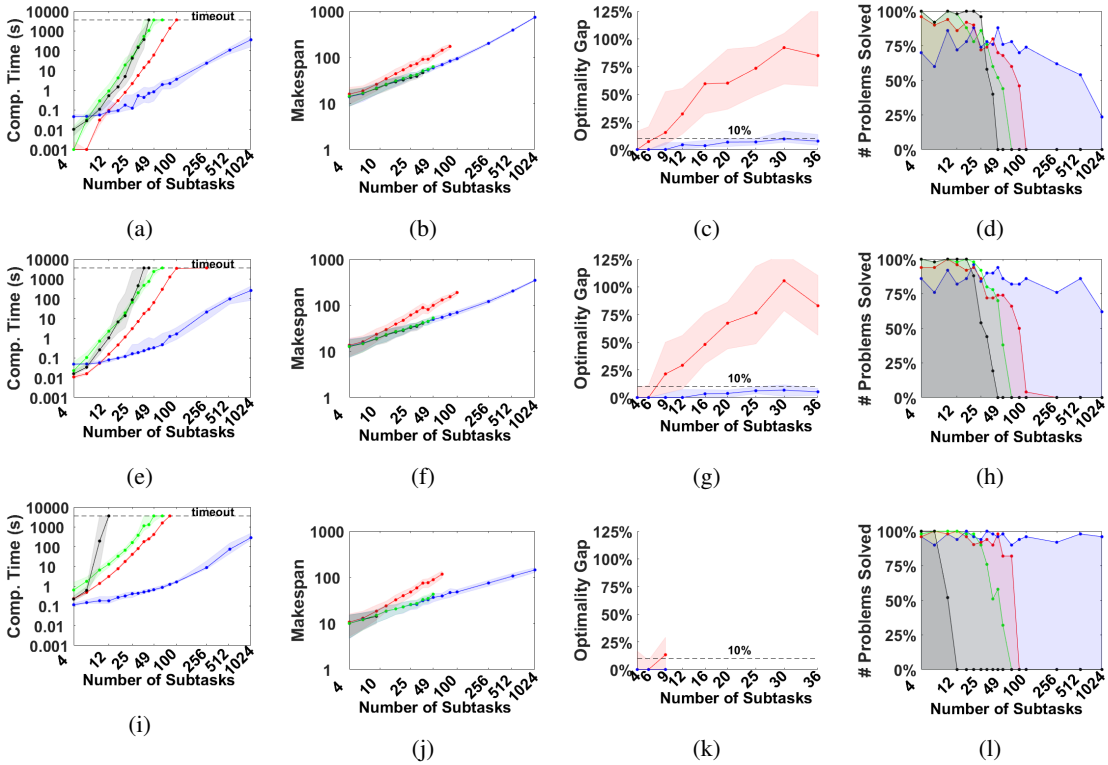

Fig. 10: This figure depicts the performance of Tercio relative to TeSSI*, OCGA, and the exact solution (where possible). Subfigures 10a-10d, 10e-10h, and 10i-10l present results for problems involving 5, 10, and 100 agents, respectively. When reporting computation time or solution quality (i.e., the makespan), the graphs depict the median and quartiles. Tercio is represented in blue, TeSSI* in red, OCGA in green, and the MILP-based solution technique in black. Note that OCGA is not depicted when reporting the optimality gap, as we terminated OCGA once it found a solution as good as that of Tercio.

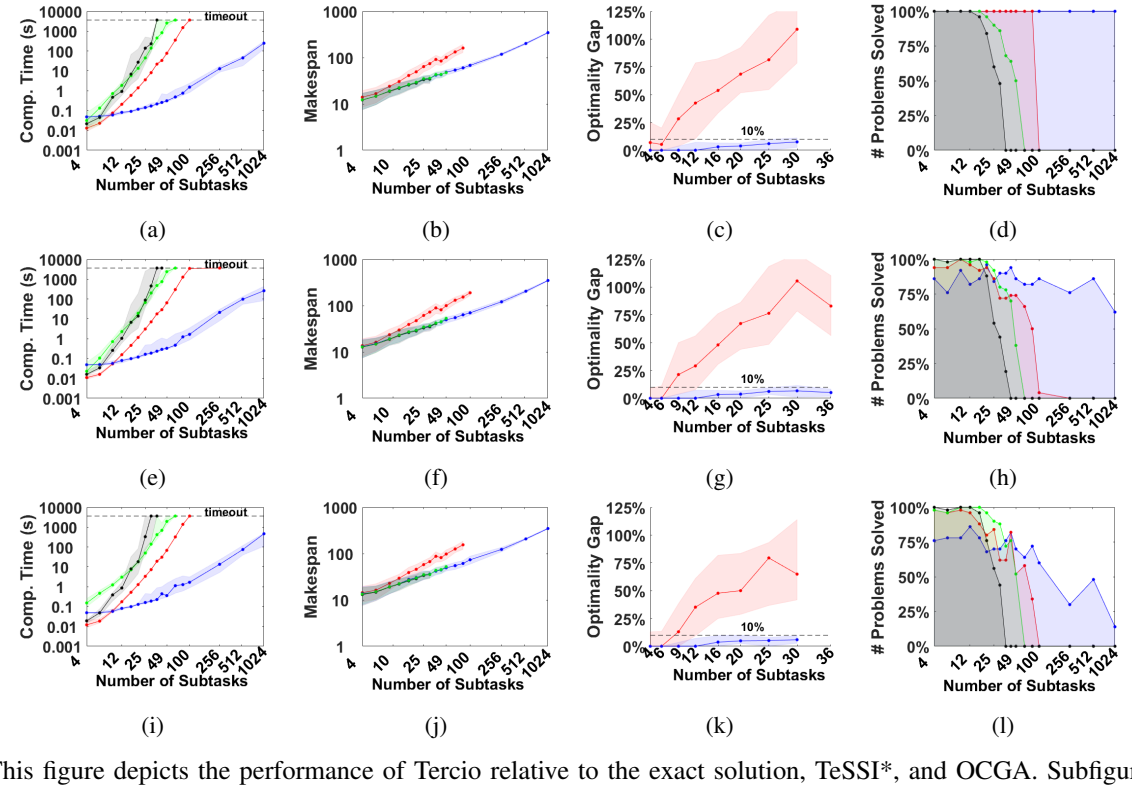

Fig. 11: This figure depicts the performance of Tercio relative to the exact solution, TeSSI*, and OCGA. Subfigures 11a-11d, 11e-11h, and 11i-11l present results for problems that involve 10 agents and are lightly, moderately, and severely constrained, respectively. Tercio is depicted in blue, TeSSI* in red, OCGA in green, and the MILP-based solution technique in black.

and $1,024$ subtasks. Where possible, we provide the computation time for the MILP-based solution method, the insertion algorithm, and the time required for OCGA to find a solution as good as that identified by Tercio. Note that the median is typically reported for such optimization problems because the distributions are often skewed and the mean is a less informative measure (Tsamardinos and Pollack, 2003).

Tercio can solve problems involving up to 100 agents and 1,000 subtasks in ∼120 seconds, a substantial improvement over our benchmarks. We note that TeSSI* and OCGA compute more slowly than previously published variants because of the need to include a temporospatial feasibility test for each assignment according to the upper- and lowerbound temporospatial constraints, requiring $O(n^2)$ step in the innermost loop of the algorithm. This feasibility test greatly increases computation time despite using the fastest technique available (Planken et al., 2012). Our findings underscore the benefit to computation time provided by our schedulability test.

10.3. Optimality

We empirically validate that Tercio produces solutions within 10% of optimal. Figures 10c, 10g, and 10k depict the median and upper/lower quartiles of the makespans produced by Tercio, along with the insertion algorithm for 50 randomly generated problems involving 5 and 10 agents and problem sizes spanning four to 42 tasks. The median deviation from optimal for Tercio was less than 10% for all testable problem sizes. However, the median deviation from optimal for the insertion algorithm increased up to 100% for larger problems.

10.4. Evaluating Completeness

Figures 10d, 10h, and 10l depict the proportion of problems solved by Tercio with 5, 10, and 100 agents, respectively. While TeSSI* and OCGA were able to solve slightly more smaller problems than Tercio, Tercio's ability overtook that of both TeSSI* and OCGA as problem size increased. Further, Tercio's completeness is proportional to the number of agents; conversely, OCGA and TeSSI* appear less able to identify satisfactory solutions as the number of agents increases.

10.5. Robustness

To test the robustness of our approach, we considered problems that are more- or less-constrained in Figures 11a-11d, 11e-11h, and 11i-11l. We set $\hat{D} = \{\frac{0}{4}, \frac{1}{4}, \frac{2}{4}\}$, respectively. Further, we varied the proportion of subtasks with wait constraints $W_{\langle \tau_i^j, \tau_x^y \rangle}$ such that $\tau_x^y \neq \tau_i^{j+1}$ in the set $\{\frac{0}{4}, \frac{1}{4}, \frac{2}{4}\}$, respectively. We found that, relative to OCGA and TeSSI*, Tercio's performance remained strong across a range of constraint settings. Tercio's completeness did degrade for the most-constrained problems (Figure 11l), but its scalability and solution quality remained strong relative to our benchmarks.

10.6. Robot Demonstration

Here, we demonstrate the use of Tercio in two hypothetical manufacturing scenarios. In both cases, a team (i.e., set) of robots worked to complete tasks on a simulated fuselage. The robots performed their tasks at specific locations on the factory floor, where there can be multiple subtasks at each location. In order to prevent collisions, each robot reserved the physical location for its subtasks, along with any immediately adjacent task locations; thus, no two workers could be present at the same location (or in neighboring locations) at the same time. For simplicity, only absolute deadlines D^{abs} were considered, although other constraints could be easily incorporated.

In the first evaluation, two KUKA Youbots simulated completion of drilling tasks on an aerospace fuselage, as shown in Figure 12 (video available at http://tiny.cc/t6wjxw). Initially, the robots planned to evenly split 12 identical tasks down the middle of the fuselage. After the robots completed their first subtasks, a worker then requested time to inspect the completed work along the left half of the fuselage; in the problem formulation, this corresponds to adding a resource reservation for the left half for a specified period of time. Tercio re-planned in response to the addition of this new constraint, and reallocated work in a reasonable manner to make productive use of both robots. For the purposes of this demonstration, the re-planning process involved calling Tercio while forcing a fixed assignment and ordering of all subtasks completed prior to the re-plan request. More-efficient methods could also be considered, such as pruning completed subtasks and reformulating appropriate temporal constraints.

Second, we demonstrated Tercio on a larger simulated problem, as shown in Figure 13 (video available at http://tiny.cc/jladjy). In this demonstration, which incorporated ABB's Robot Studio simulation environment, Tercio coordinated 5 robots to perform 110 identical tasks around an aerospace fuselage. Tercio was applied to re-plan in response to three disturbances: 1) a human workers request to enter the space to perform a quality-assurance (QA) inspection, 2) a robot breakdown, and 3) changing task deadlines. The duration of the QA request and the robot breakdown were known at the time of the disturbance. Tercio modeled the quality-assurance request as a resource reservation for the section of the fuselage to be inspected (i.e., no robots could occupy that space for a given duration). Tercio modeled the robot breakdown as an agent occupied by a repair subtask for a given duration.

These demonstrations indicate the applicability of the Tercio algorithm for solving real-world problems with application-specific features and complex objective functions. For example, localization errors among mobile robots must be accounted for during schedule generation. Intra-robot errors accumulate, which can result in gaps or overlaps in work. In order to reduce the number of opportunities for such errors to occur, we added a term to reduce the number of times when two different agents were used to perform adjacent work, as defined by $g_4(\boldsymbol{A}, \boldsymbol{\tau}) \geq \sum_{a \in \boldsymbol{A}} \sum_{\langle \tau_i^j, \tau_x^y \rangle \in \boldsymbol{\tau_R}} A^a_{\tau_i^j} \times (1 - A^a_{\tau_x^y})$. Also, schedule changes in response to disturbances must be interpretable by human workers. Small disturbances can potentially result in substantial changes to robot task assignments, and human workers may find it challenging to understand and anticipate the actions of the robots. In many cases, small schedule changes may be an acceptable trade-off for a marginally sub-optimal solution. As such, we included an additional term to minimize the number of assignment

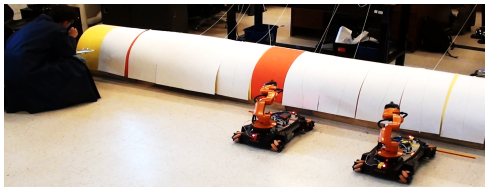

Fig. 12: A robotic team completes mock aerospace final-assembly tasks while maintaining a safe distance from human workers.

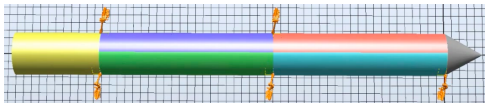

Fig. 13: A robotic team adapts performance of work related to a virtual fuselage in response to multiple disruptions.

changes from one schedule's allocation $P^a_{\tau_i^j}$ to the next $A^a_{\tau_i^j}$, as defined by $g_5(\boldsymbol{A}, \boldsymbol{P}, \boldsymbol{\tau}) \geq \sum_{a \in \boldsymbol{A}} \sum_{\tau_i^j \in \boldsymbol{\tau}} A^a_{\tau_i^j} \times (1 - P^a_{\tau_i^j})$. While practitioners might attempt to apply other scheduling technology (e.g., one of the benchmarks we employed), we believe Tercio provides a singular ability to rapidly and near-optimally reschedule robots in dynamic environments.

11. Conclusion

We developed Tercio, a scheduling algorithm made efficient through a fast, multi-agent task sequencer inspired by real-time processor scheduling techniques. We used the task sequencer in conjunction with a MILP solver to compute an integrated multi-agent task sequence that satisfies precedence and temporal and spatial-proximity constraints. Although Tercio is incomplete, we empirically show that the algorithm produces schedules within 10% of the optimal for real-world, structured problems. We also show that Tercio is able to solve more problems and achieve better-quality solutions for such problems than the prior state-of-the-art technique. Finally, we provide physical and virtual demonstrations of Tercio coordinating the activities of a robotic team within a human work environment.

References

A. Ahmed, A. Patel, T. Brown, M. Ham, M.-W. Jang, and G. Agha. Task assignment for a physical agent team via a dynamic forward/reverse auction mechanism. In *Proc. KIMAS*, April 18-21 2005.

P. Baptiste and C. L. Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1-2):119–139, 2000.

R. W. Beard, T. W. McLain, M. A. Goodrich, and E. P. Anderson. Coordinated target assignment and intercept for unmanned air vehicles. *T-RA*, 18(6):911–922, 2002.

J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

D. P. Bertsekas. Auction algorithms for network flow problems: A tut. intro. *Comp. Opt. and App.*, 1:7–66, 1990.

P. Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

A. M. Campbell and M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transport. Sci.*, 38(3):369–378, 2004.

D. A. Castañón and C. Wu. Distributed algorithms for dynamic reassignment. In *Proc. CDC*, volume 1, pages 13–18, 2003.

E. Castro and S. Petrovic. Combined mathematical programming and heuristics for a radiotherapy pre-treatment scheduling problem. *J. Scheduling*, 15(3):333–346, 2012.

A. Cesta, A. Oddi, and S. F. Smith. A constraint-based method for project scheduling with time windows. *J. Heuristics*, 8 (1):109–136, 2002.

F. Chen, K. Sekiyama, F. Cannella, and T. Fukuda. Optimal subtask allocation for human and robot collaboration within hybrid assembly system. *T-ASE*, 11(4):1065–1075, 2014.

J. Chen and R. G. Askin. Project selection, scheduling and resource allocation with time dependent returns. *EJOR*, 193: 23–34, 2009.

S. Chien, A. Barrett, T. Estlin, and G. Rabideau. A comparison of coordinated planning methods for cooperating rovers. In *Proc. ICAA*, pages 100–101, 2000.

H.-L. Choi, L. Brunet, and J. P. How. Consensus-based decentralized auctions for robust task allocation. *T-RO*, 25 (4):912–926, Aug 2009.

J.-F. Cordeau, G. Stojković, F. Soumis, and J. Desrosiers. Benders decomposition for simultaneous aicraft routing and crew scheduling. *Transport. Sci.*, 35(4):357–388, 2001.

J. W. Curtis and R. Murphey. Simultaneous area search and task assignment for a team of cooperative agents. In *Proc. GNC*, 2003.

M. Dai, D. Tang, A. Giret, M. A. Salido, and W. D. Li. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Rob. and Comp.-Intg. Manuf.*, 29(5):418–429, 2013.

L. Davis. Job shop scheduling with genetic algorithms. In *Proc. Int'l Conference on GA's*, 1985.

R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *AI*, 49(1), 1991.

U. C. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Proc. ECRTS*, 2003.

A. Drexl, R. Nissen, J. H. Patterson, and F. Salewski. Progen/πx an instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. *EJOR*, 125(1):59 – 72, 2000.

E. Erdem, E. Aker, and V. Patoglu. Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. *ISR*, 5(4):275–291, 2012a.

E. Erdem, K. Haspalamutgil, V. Patoglu, and T. Uras. Causality-based planning and diagnostic reasoning for cognitive factories. In *Proc. ETFA*, pages 1–8, 2012b.

E. Erdem, V. Patoglu, Z. G. Saribatur, P. Schüller, and T. Uras. Finding optimal plans for multiple teams of robots through a mediator: A logic-based approach. *Theory and Practice of Logic Programming (Sl)*, 13(4-5):831–846, 2013.

E. Falkenauer and S. Bouffouix. A genetic algorithm for job shop. In *Proc. ICRA*, pages 824–829. IEEE, 1991.

H.-L. Fang, P. Ross, and D. Corne. A promising genetic al-

gorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In *Proc. Int'l Conference on GA's*, pages 375–382, 1993.

V. Fernandez-Viagas and J. M. Framinan. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, 45:60–67, 2014.

R. W. Floyd. Algorithm 97: Shortest path. *Comm. ACM*, 5 (6):345–, June 1962.

M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *OR*, 46(3):330–335, 1998.

A. M. Geoffrion. Generalized benders decomposition. *J. Optimization Theory and Applications*, 10(4):237–260, 1972.

M. Godinho Filho, C. F. Barco, and R. F. T. Neto. Using genetic algorithms to solve scheduling problems on flexible manufacturing systems (fms): a literature survey, classification and analysis. *Flexible Services and Manufacturing Journal*, 26(3):408–431, 2014.

M. C. Gombolay and J. A. Shah. Uniprocessor scheduler for task sets with well-formed precedence relations, temporal deadlines, and wait constraints. In *Proc. AIAA Infotech@Aerospace*, 2012.

M. C. Gombolay and J. A. Shah. Schedulability analysis of task sets with upper- and lower-bound temporal constraints. *J. Aerospace Information Systems*, 11(12):821–841, 2015.

M. C. Gombolay, R. J. Wilcox, and J. A. Shah. Fast scheduling of multi-robot teams with temporospatial constrints. In *Proc. RSS*, Berlin, Germany, June 24-28, 2013.

I. Gurobi Optimization. Gurobi optimizer reference manual, 2016. URL http://www.gurobi.com.

M. G. Harbour and J. C. Palencia. Response time analysis for tasks scheduled under EDF within fixed priorities. In *Proc. RTSS*, 2003.

I. Harjunkoski and I. E. Grossman. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers & Chemical Engineering*, 26:1533–1552, 2002.

J. N. Hooker. *Logic-Based Benders Decomposition, in Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. John Wiley & Sons, Inc., 2000.

V. Jain and I. E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *J. Computing*, 13(4):258–276, 2001.

S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

E. Jones, M. Dias, and A. Stentz. Time-extended multi-robot coordination for domains with intra-path constraints. *AuRo*, 30(1):41–56, 2011.

G. A. Korsah. *Exploring bounded optimal coordination for heterogeneous teams with cross-schedule dependencies*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 2011.

G. A. Korsah, A. Stentz, and M. B. Dias. A comprehensive taxonomy for multi-robot task allocation. *IJRR*, 32(12): 1495–1512, 2013.

A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar. Towards a swarm of agile micro quadrotors. *AuRo*, 35(4): 287–300, 2013.

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

K. Lakshmanan and R. R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *Proc. RTAS*, 2010.

K. Lakshmanan, S. Kato, and R. R. Rajkumar. Open problems in scheduling self-suspending tasks. In *Proc. RTSOPS*, 2010.

T. Lemaire, R. Alami, and S. Lacroix. A distributed tasks allocation scheme in multi-uav context. In *Proc. ICRA*, pages 3622–3627, April 2004.

J. K. Lenstra and A. H. G. R. Kan. Complexity of scheduling under precedence constraints. *OR*, 26(1):22–35, 1978.

H. Li and K. Womer. Scheduling projects with multi-skill personnel by a hybrid MILP/CP benders decomposition algorithm. *J. Scheduling*, 12:281–298, 2008.

C. Liu and J. H. Anderson. Improving the schedulability of sporadic self-suspending soft real-time multiprocessor task systems. In *Proc. RTCSA*, 2010.

L. Liu and D. A. Shell. Optimal market-based multi-robot task allocation via strategic pricing. In *Proc. RSS*, 2013.

R. K. Martin. *Large Scale Linear and Integer Optimization: A unified Approach*. Kluwer Acadmic Publishers, 1999.

M. McIntire, E. Nunes, and M. Gini. Iterated multi-robot auctions for precedence-constrained task scheduling. In *Proc. AAMAS*, pages 1078–1086, 2016.

A. Mercier, J.-F. Cordeau, and F. Soumis. A computational study of benders decomposition for the integrated aircraft routing and crew scheduling problem. *Computers & Operations Research*, 32(6):1451 – 1476, 2005.

S. M. Mousavi and R. Tavakkoli-Moghaddam. A hybrid simulated annealing algorithm for location and routing scheduling problems with cross-docking in the supply chain. *J. Manufacturing Systems*, 32(2):335–347, 2013.

N. Muscettola, P. Morris, and I. Tsamardinos. Reformulating temporal plans for efficient execution. In *Proc. KR&R*, June 2-5, 1998.

E. Nunes and M. Gini. Multi-robot auctions for allocation of tasks with temporal constraints. In *Proc. AAAI*, pages 2110–2116, 2015.

I. H. Osman and C. Potts. Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551–557, 1989.

J. H. Patterson. A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Science*, 30(7):854 – 867, 1984.

J. M. Pinto and I. E. Grossmann. A continuous time MILP model for short term scheduling of multistage batch plants. *Industrial & Eng. Chem. Research*, 34(9):3037–3051, 1995.

E. Plaku. Planning in discrete and continuous spaces: From ltl tasks to robot motions. In *Adv. in Autonomous Robotics*, pages 331–342. Springer, 2012.

L. R. Planken, M. M. de Weerdt, and R. P. van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *JAIR*, pages 353–388, 2012.

S. Ponda, J. Redding, H.-L. Choi, J. P. How, M. Vavrina, and

J. Vian. Decentralized planning for complex missions with dynamic communication constraints. In *Proc. of ACC*, pages 3998–4003. IEEE, 2010.

R. R. Rajkumar. Dealing with self-suspending period tasks. Technical report, IBM, Armonk, New York, 1991.

M. Rekik, J.-F. Cordeau, and F. Soumis. Using benders decomposition to implicitly model tour scheduling. *Transport. Sci.*, 128:111–133, 2004.

H. Ren and L. Tang. An improved hybrid milp/cp algorithm framework for the job-shop scheduling. In *Proc. ICAL*, August 2009.

W. Ren, R. W. Beard, and T. W. McLain. Coordination variables, coordination functions, and cooperative timing missions. *GNC*, 28(1):150–161, 2005.

F. Ridouard and P. Richard. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Proc. RTNS*, May 30-31 2006.

S. Sariel and T. Balch. Real time auction based allocation of tasks for multi-robot exploration problem in dynamic environments. In *Proc. AAAI Workshop on Integrating Planning into Scheduling*, July 9 - 13 2005.

D. E. Smith, F. Frank, and A. Jónsson. Bridging the gap between planning and scheduling. *KER*, 15:47–83, 2000.

M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *OR*, 35(2):254–265, 1987.

Y. N. Sotskov and N. V. Shakhlevich. Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237–266, 1995.

C. Sung, N. Ayanian, and D. Rus. Improving the performance of multi-robot systems by task switching. In *Proc. ICRA*, 2013.

K. P. Sycara, S. F. Roth, N. Sadeh, and M. S. Fox. Resource allocation in distributed factory scheduling. *Expert*, 6(1): 29–40, 1991.

S. J. R. Tal Shima and P. Chandler. UAV team decision and control using efficient collaborative estimation. In *Proc. ACC*, pages 4107–4112, June 8-10 2005.

W. Tan. A linearized polynomial milp for the integration of process planning and scheduling. *JIM*, 15:593–605, 2004.

H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *Proc. Heterogeneous Computing Workshop*, pages 3–14. IEEE, 1999.

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

I. Tsamardinos, N. Muscettola, and P. Morris. Fast transformation of temporal plans for efficient execution. In *Proc. AAAI*, 1998.

P. J. Van Laarhoven, E. H. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *OR*, 40(1):113–125, 1992.

P. Vilím, R. Barták, and O. Čepek. Extension of $o(n\ log\ n)$ filtering algorithms for the unary resource constraint to optional activities. *Constraints*, 10(4):403–425, 2005.

L. Wang and D.-Z. Zheng. A modified genetic algorithm for job shop scheduling. *The Int'l J. Adv. Manufacturing Technology*, 20(1):72–76, 2002.

R. J. Wilcox, S. Nikolaidis, and J. A. Shah. Optimization of temporal dynamics for adaptive human-robot interaction in assembly manufacturing. In *Proc. RSS*, 2012.

M. M. Zavlanos, L. Spesivtsev, and G. J. Pappas. A distributed auction algorithm for the assignment problem. In *Proc. IEEE CDC*, 2008.

F. Zhang and A. Burns. Schedulability analysis for real-time systems with edf scheduling. *IEEE TC*, 58(9):1250–1258, 2009.

F. Zhang, Y. Zhang, and A. Y. C. Nee. Using genetic algorithms in process planning for job shop machining. *IEEE TEVC*, 1(4):278–289, 1997.

L. Zhang and T. Wong. An object-coding genetic algorithm for integrated process planning and scheduling. *EJOR*, 244 (2):434–444, 2015.

H. Zhao, L. George, and S. Midonnet. Worst case response time analysis of sporadic task graphs with edf non-preemptive scheduling on a uniprocessor. In *Proc. ICAS*, pages 22–22, 2007.

Matthew Gombolay will join the faculty in the School of Interactive Computing at the Georgia Institute of Technology as the Catherine M. and James E. Allchin Early-Career Associate Professor in Fall 2018, where he develops novel computational methods for dynamic, fluent human-robot teaming. Gombolay received his B.S. (2011) from the Mechanical Engineering Department at Johns Hopkins University, his S.M. (2013) from the Department of Aeronautics and Astronautics at MIT, and his Ph.D. (2017) in Autonomous Systems from MIT.

Ron Wilcox is currently an Associate at Oliver Wyman in New York City, New York. He received his B.A. (2011) in physics from William and Mary and his M.S. (2013) from the Mechanical Engineering Department at MIT.

Julie Shah is an Associate Professor of Aeronautics and Astronautics at MIT and director of the Interactive Robotics Group, which aims to imagine the future of work by designing collaborative robot teammates that enhance human capability. Before joining the faculty, she worked at Boeing Research and Technology on robotics applications for aerospace manufacturing. Shah received her SB (2004) and SM (2006) from the Department of Aeronautics and Astronautics at MIT, and her PhD (2010) in Autonomous Systems from MIT.