# Fast Scheduling of Multi-Robot Teams 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*— New uses of robotics in traditionally manual manufacturing processes require the careful choreography of human and robotic agents to support safe and efficient coordinated work. Tasks must be allocated among agents and scheduled to meet temporal deadlines and spatial restrictions on agent proximity. These systems must also be capable of replanning on-the-fly to adapt to disturbances in the schedule and to respond to people working in close physical proximity. In this paper, we present a centralized algorithm, named Tercio, that handles tightly intercoupled temporal and spatial constraints and scales to larger problem sizes than prior art. Our key innovation is a fast, satisficing multi-agent task sequencer that is inspired by real-time processor scheduling techniques but is adapted to leverage hierarchical problem structure. We use this fast task sequencer in conjunction with a MILP solver, and show that we are able to generate near-optimal task assignments and schedules for up to 10 agents and 500 tasks in less than 20 seconds on average. Finally, we demonstrate the algorithm in a multi-robot hardware testbed.**

## I. INTRODUCTION

Robotic systems are increasingly entering domains previously occupied exclusively by humans. In manufacturing, there is strong economic motivation to enable human and robotic agents to work in concert to perform traditionally manual work. This integration requires a choreography of human and robotic work that meets upperbound and lowerbound temporal deadlines on task completion (e.g. assigned work must be completed within one shift) and spatial restrictions on agent proximity (e.g. robots must maintain four meter separation from other agents), to support safe and efficient human-robot co-work. The multi-agent coordination problem with temporospatial constraints can be readily formulated as a mixed-integer linear program (MILP). However, the complexity of this approach is exponential and leads to computational intractability for problems of interest in large-scale factory operations [4].

Various decentralized or distributed approaches achieve fast computation and good scalability characteristics [5], [6], [9], [24], [31]. Fast computation is desirable because it provides the capability for on-the-fly replanning in response to schedule disturbances [2], [6], [27]. These works boost computational performance by decomposing plan constraints and contributions to the objective function among agents [5]. However, these methods break down when agents' schedules become tightly intercoupled, as they do when multiple agents are maneuvering in close physical proximity. While distributed approaches to coordination are necessary for field operations where environment and geography affect the communication among agents, factory operations allow for 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 replanning with temporal and spatial-proximity constraints. We demonstrate that this capability enables human and robotic agents to effectively work together in close proximity to perform manufacturing-relevant tasks.

Tercio is made efficient through a fast, satisficing multi-agent task sequencer that is inspired by real-time processor scheduling techniques but is adapted to leverage hierarchical problem structure. Our task sequencer computes in polynomial time a multi-agent schedule that satisfies upperbound and lowerbound temporal deadlines as well as spatial restrictions on agent proximity. Although the sequencing algorithm is satisficing, we show that it is *tight*, meaning it produces near-optimal task sequences for real-world, structured problems. We use this fast task sequencer in conjunction with a MILP solver, and show that we are able to generate near-optimal task assignments and schedules for up to 10 agents and 500 tasks in less than 20 seconds on average. In this regard, Tercio scales better than previous approaches to hybrid task assignment and scheduling [7], [8], [13], [14], [15], [32]. An additional feature of Tercio is that it returns flexible time windows for execution [22], [34], which enable the agents to adapt to small disturbances online without a full re-computation of the schedule.

In Section II we place our work in the context of prior art. In Section III we formally define the problem we solve and outline our technical approach. Section IV presents the Tercio algorithm including pseudocode, and Section V describes the fast multi-agent task sequencer, called as a subroutine within Tercio. Section VI presents the empirical evaluation, and describes the application of the algorithm in a multi-robot hardware testbed.

## II. RELATED WORK

There is a wealth of prior work in task assignment and scheduling for manufacturing and other applications. While

*These authors contributed equally to this work. **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]Our method is named Tercio 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.

the problem in many cases may be readily formulated and solved as a mixed-integer linear program (MILP), the complexity of this approach is exponential and leads to computational intractability for problems of interest in large-scale factory operations [4]. To achieve good scalability characteristics, various hybrid algorithms have been proposed. A brief survey of these methods follows.

One of the most promising approaches has been to combine MILP and constraint programming (CP) methods into a hybrid algorithm using decomposition (e.g. Benders Decomposition) [13], [14], [15]. This formulation is able to gain orders of magnitude in computation time by using a CP to prune the domain of a relaxed formulation of the MILP. However, if the CP is unable to make meaningful cuts from the search space, this hybrid approach is rendered nearly equivalent to a non-hybrid formulation of the problem. Auction methods (e.g. [5]) also rely on decomposition of problem structure and treat the optimization of each agent's schedule as independent of the other agents' schedules. These techniques preclude explicit coupling in each agent's contribution to the MILP objective function. While the CP and auction-based methods support upperbound and lowerbound temporal deadlines among tasks, they do not handle spatial proximity constraints, as these produce tight dependencies among agents' schedules that make decomposition problematic.

Other hybrid approaches integrate heuristic schedulers within the MILP solver to achieve better scalability characteristics. For example, Chen *et al.* incorporate depth-first search (DFS) with heuristic scheduling [8], and Tan incorporates Tabu Search [32] within the MILP solver. Castro *et al.* use a heuristic scheduler to seed a feasible schedule for the MILP [7], and Kushleyev *et al.* [16] apply heuristics for abstracting the problem to groupings of agents. These methods solve scheduling problems with 5 agents (or groups of agents) and 50 tasks in seconds or minutes, and address problems with multiple agents and resources, precedence among tasks, and temporal constraints relating task start and end times to the plan epoch time. However, more general task-task temporal constraints are not considered.

In the next section we describe our approach, which solves task assignment and scheduling problems with the full set of features: multiple agents, precedence and temporal constraints among tasks, and spatial proximity constraints.

## III. Our Approach

### A. Problem Statement

In this section, we formulate the task assignment and scheduling problem for multiple robots moving and working in the same physical space as a mixed-integer linear program (MILP). Problem inputs include:

- a **Simple Temporal Problem (STP)** [10] describing the interval temporal constraints (lowerbound $lb$ and upperbound $ub$) relating tasks (e.g. "the first coat of paint requires 30 minutes to dry before the second coat may be applied" maps to interval constraint $secondCoatStart - firstCoatFinish \in [30, \infty)$, and the constraint that "the entire task set must be finished within the 120 minutes" maps to $finishTaskset - startTaskSet \in [0, 120]$),
- **two-dimensional (x,y) positions** specifying the floor spatial locations where tasks are performed,
- **agent capabilities** specifying the tasks each agent may perform and the agent's expected time to complete each task, and
- **allowable spatial proximity** between agents.

A solution to the problem consists of an assignment of tasks to agents and a schedule for each agent's tasks such that all constraints are satisfied and the objective function is minimized. The mathematical formulation of the problem is presented below:

$$min\ f(A, P, J, S, R, \gamma) \tag{1}$$

subject to

$$\sum_{a \in Ag} A_{a,j} = 1, \forall j \in \gamma \tag{2}$$

$$lb_i \leq t_m - t_n \leq ub_i, \forall i \in T, n, m \in \gamma \tag{3}$$

$$t_k^E - t_k^S \geq lb_{a,k} - M(1 - A_{a,k}), \forall k \in \gamma, a \in Ag \tag{4}$$

$$t_k^E - t_k^S \leq ub_{a,k} + M(1 - A_{a,k}), \forall k \in \gamma, a \in Ag \tag{5}$$

$$t_j^S - t_i^E \geq M(1 - J_{i,j}), \forall i, j \in R \tag{6}$$

$$t_i^S - t_j^E \geq MJ_{i,j}, \forall i, j \in R \tag{7}$$

$$t_j^S - t_i^E \geq M(1 - J_{i,j}) + M(2 - A_{a,i} - A_{a,j}) \quad \forall i, j \in \gamma \tag{8}$$

$$t_i^S - t_j^E \geq MJ_{i,j} + M(2 - A_{a,i} - A_{a,j}) \quad \forall i, j \in \gamma \tag{9}$$

where $A_{a,j} \in \{0, 1\}$ is a binary decision variable for the assignment of agent $a$ to task $j$. $J_{i,j}$ is a binary decision variable specifying the relative sequencing of two tasks $i$ and $j$ ($J_{i,j}$ = 1 implies task $i$ occurs before $j$). $T$ is the set of all interval temporal constraints relating tasks, and is equivalently encoded and referred to as the Simple Temporal Problem (STP) [10]. $R$ is the set of task pairs $(i, j)$ that are separated by less than the allowable spatial proximity. $Ag$ is the set of all agents, $\gamma$ is the set of all tasks, and $t_i^S$ and $t_i^E$ represent the start and end times of task $i$, respectively. Finally, $P_{a,i}$ is the variable $A_{a,i}$ from a previous agent allocation, used to avoid oscillation between equivalent solutions. $M$ is a large positive number, and is used to encode conditional constraints. Figure 1 visually depicts a problem instance of this MILP, with two robots and six tasks (depicted as six stripes on the workpiece).

Equation 2 ensures that each task is assigned to one agent. Equation 3 ensures that the temporal constraints relating tasks are met. Equations 4 & 5 ensure that agents are not required to complete tasks faster or slower than they are capable. Equations 6 & 7 sequence actions to ensure that agents performing tasks maintain safe buffer distances from one another. Equations 8 & 9 ensure that each agent

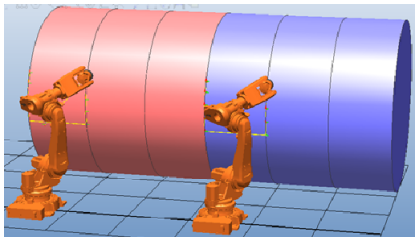

Fig. 1. Example of a team of robots assigned to tasks on a cylindrical structure.

only performs one task at a time. Note Equations 6 and 7 couple the variables relating sequencing constraints, spatial locations, and task start and end times, resulting in tight dependencies among agents' schedules.

The objective function $f(A, P, J, S, R, \gamma)$ is application specific. In our empirical evaluation in Section VI we use an objective function that includes three equally weighted terms. The first term minimizes $g(A, P, \gamma)$,the difference between the previous agent assignment and the returned agent assignment. Minimizing this quantity helps to avoid oscillation among solutions with equivalent quality during replanning. The second term $h(A, R)$ minimizes the number of spatial interfaces between tasks performed by different robots. Inter-robot accuracy is challenging for multi-robot systems of standard industrial robots. In robot painting, this can lead to gaps or overlaps at interfaces between work done by two different robots, and so we seek a task assignment with the fewest interfaces possible. In Figure 1 the agent allocation results in one interface between the red work assigned to the left robot and the blue work assigned to the right robot. The third term $q(A, J, S, E, \gamma)$ minimizes the time to complete the entire process (i.e. the makespan).

### B. Technical Approach

Next we outline our technical approach for efficiently solving this MILP. Tercio is made efficient through a fast, satisficing, incomplete multi-agent task sequencer that is inspired by real-time processor scheduling techniques, but adapted to leverage hierarchical problem structure. We decompose the MILP into a task allocation and a task sequencing problem. We modify the MILP to support this decomposition, and use the task sequencer to efficiently solve for the subproblem involving Equations 3-9, and objective term $q(A, J, S, E, \gamma)$. We demonstrate that this approach is able to generate near-optimal schedules for up to 10 agents and 500 work packages in less than 20 seconds.

*Real-Time Processor Scheduling Analogy:* We use a processor scheduling analogy to inspire the design of an informative, polynomial-time task sequencer. In this analogy, each agent is a computer processor that can perform one task at a time. A physical location in discretized space is considered a shared memory resource that may be accessed by up to one processor at a time. Wait constraints (lowerbounds on interval temporal constraints) are modeled as "self-suspensions," [19], [26] times during which a task is blocking while another piece of hardware completes a time-durative task.

**TERCIO**($STP, P_{a,i}, Ag, \gamma, R$, *cutoff*)

1: $makespan \leftarrow \inf$
2: **while** $makespan \geq$ *cutoff* **do**
3: $A \leftarrow$ exclude previous allocation $P_{a,i}$ from agent capabilities
4: $A \leftarrow$ TERCIO-ALLOCATION($\gamma, STP, Ag$)
5: $STP \leftarrow$ update agent capabilities
6: $makespan, seq \leftarrow$ TERCIO-SEQUENCER($\boldsymbol{\tau}$,*cutoff*)
7: **end while**
8: $STP \leftarrow$ add ordering constraints to enforce $seq$
9: $STP \leftarrow$ DISPATCHABLE($STP$)
10: **return** STP

Fig. 2. Psuedo-code for the Tercio Algorithm.

Typically, assembly manufacturing tasks have more structure (e.g., parallel and sequential subcomponents), as well as more complex temporal constraints than real-time processor scheduling problems. AI scheduling methods handle complex temporal constraints and gain computational tractability by leveraging hierarchical structure in the plan [28]. We bridge the approaches in AI scheduling and real-time processor scheduling to provide a fast multi-agent task sequencer that satisfies tightly coupled upperbound and lowerbound temporal deadlines and spatial proximity restrictions (shared resource constraints). While our method relies on a plan structure composed of parallel and sequential elements, we nonetheless find this structural limitation sufficient to represent many real-world factory scheduling problems.

## IV. Tercio Algorithm

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 replanning with temporal and spatial-proximity constraints. Pseudo-code for the Tercio algorithm is presented in Figure 2.

The inputs to Tercio are as described in Section III-A. Tercio also takes as input a user-specified makespan *cutoff* (Line 2) to terminate the optimization process. This can often be derived from the temporal constraints of the manufacturing process. For example, a user may specify that the provided task set must be completed within an eight-hour shift. Tercio then iterates (Lines 3-7) to compute an agent allocation and schedule that meets this makespan. Because Tercio uses a satisficing and incomplete sequencer, it is not guaranteed to find an optimal solution, or even a satisficing solution if one exists. In practice, we show (Section VI) Tercio produces makespans within about $10\%$ of the optimal minimum makespan, for real-world structured problems.

### A. Tercio Agent Allocation

Tercio performs agent-task allocation by solving a simplified version of the MILP from Section III. The objective function for the agent allocation MILP is formulated as:

$$Objective = min\ g(A, P, \gamma) + h(A, R) + v, \qquad (10)$$

where, recall $g$ minimizes the difference between the previous agent assignment and the returned agent assignment to help avoid oscillations between equivalent quality solutions during replanning, and $h$ minimizes the number of spatial interfaces between tasks performed by different robots.

We introduce a proxy variable $v$ into the objective function to perform work-balancing and guide the optimization towards agent allocations that yield a low makespan. The variable $v$ encodes the maximum total task time that all agents would complete their tasks, if those tasks had no deadline or delay dependencies and is defined as:

$$v \geq \sum_j c_j \times A_{a,j} \forall a \tag{11}$$

where $c_j$ is a constant representing the expected time of each task. We find in practice the addition of this objective term and constraint guides the solution to more efficient agent allocations. The agent allocation MILP must also include Equations 2 and 3 ensuring each task is assigned to exactly one agent and that the agent-task allocation does not violate the STP constraints.

### B. Tercio Pseudocode

A third-party optimizer [1] solves the simplified agent-allocation MILP (Line 4) and returns the agent allocation matrix $A$. Interval temporal (STP) constraints are updated based on this agent allocation matrix by tightening task time intervals (Line 5). For example, if a task is originally designated to take between five and fifteen minutes but the assigned robot can complete it no faster than ten minutes, we tighten the interval from $[5, 15]$ to $[10, 15]$.

The agent allocation matrix, the capability-updated STP, 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 V) 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*, the algorithm iterates (Lines 3-7), each time adding a constraint (Line 3) to exclude the agent allocations tried previously:

$$\sum_{a,i|L_{a,i}=0} A_{a,i} + \sum_{a,i|L_{a,i}=1} (1 - A_{a,i}) > 0 \tag{12}$$

where $L_{a,i}$ is the solution from the last loop iteration.

Tercio terminates when the returned makespan falls beneath *cutoff*, or else when no solution can be found after iterating through all feasible agent allocations.

If the cutoff makespan is satisfied, agent sequencing constraints (interval form of $[0, \infty)$) are added to the STP constraints (Line 8). Finally the resulting Simple Temporal Problem is compiled to a dispatchable form (Line 9) [22], [34], which guarantees that for any consistent choice of a timepoint within a flexible window, there exists a solution that can be found in the future 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 recomputing solutions in response to disturbances by up to 75% for randomly generated structured problems [34].

## V. Multi-agent Task Sequencer

The key to increasing the computational speed of Tercio is our hybrid approach to task sequencing. Tercio takes as input a set of agent-task assignments and a well-formed self-suspending task model (defined below) and returns a valid task sequence if one can be found by the algorithm. The task sequencer is merely satisficing and not complete; however, we empirically validate that it returns near-optimal makespans when integrated with the Tercio Agent Allocation algorithm (See Section VI).

Our sequencing algorithm is inspired by prior art in the field of real-time systems research; we model the multi-agent coordination problem using the self-suspending task model. Scheduling of self-suspending task systems has been the focus of much recent work due to the relatively recent integration of new hardware and supporting software systems (e.g., GPUs, PPUs) that trigger external blocking of tasks [11], [19], [26]. Self-suspensions can alternatively be thought of as lowerbound temporal constraints relating tasks. For example, one might specify that a first coat of paint needs at least 30 minutes to dry before the second coat may be applied. This 30 minute wait time is a self-suspension of the painting task.

Prior work computes the uniprocessor schedulability of a task set with a single suspension [20], [21], [26]. We compute the multiprocessor schedulability of a task set where multiple tasks have more than one suspension. Our approach leverages the use of a scheduling policy that partially restricts the behavior of the scheduler to reduce multi-processor schedule anomalies due to self-suspensions. Our approach is similar in spirit to prior work [23], [30] that restricts the behavior to reduce anomalies that inherently arise when applying uniprocessor scheduling methods to self-suspending task sets [18], [26].

In this section, we first introduce our task model, which is inspired by work in real-time processor scheduling. Second, we describe how our fast task sequencer works to satisfy temporospatial constraints.

### A. Well-Formed Task Model

The Tercio Task Sequencer relies on a well-formed task model that captures hierarchical and precedence structure in the task network. The basis for our framework is the self-suspending task model [19], described in Equation 13.

$$\tau_i : ((C_i^1, E_i^1, C_i^2, E_i^2, \ldots, E_i^{m_{i-1}}, C_i^{m_i}), T_i, D_i). \tag{13}$$

In this model, that is a task set $\boldsymbol{\tau} = \{\tau_i | i \in \{1, 2, \ldots, n\}\}$ with $n$ tasks $\tau_i$ that must be processed by the computer. For each task, there are $m_i$ subtasks with $m_i - 1$ self-suspension intervals. We use $\tau_i^j$ to denote the $j^{th}$ subtask of $\tau_i$, $C_i^j$ is the expected duration (cost) of $\tau_i^j$. $E_i^j$ is the expected duration of the $j^{th}$ self-suspension interval of $\tau_i$. $T_i$ and $D_i$ are the period and deadline of $\tau_i$, respectively. For our application

we enforce that all tasks are non-preemptable, meaning the interruption of a subtask significantly degrades its quality.

While this self-suspending task model provides a solid basis for describing many real-world processor scheduling problems of interest, we augment this model to better capture problem structure inherent in the manufacturing environment. First, we set the period and deadline of each task $\tau_i$ equal to a constant, $T$ (i.e., $T_i = D_i = T, \forall i$) This modification models many assembly line manufacturing processes where the set of tasks at one location is repeated once every "pulse" 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 $T$.

The second adaptation we make is to allow phase offsets $\phi$ for each task $\tau_i$, where a phase offset is a delay between the epoch time and the release of the given task. This adaptation allows a user the ability to require that an agent wait a specified time interval before starting the first subtask of a task.

The third change we make is to enable the user to specify a intra-task and subtask deadlines. An intra-task deadline $D^{rel}_{(i,a),(i,b)}$ constrains the start, $s^a_i$ and finish time $f^b_i$ of two subtasks $\tau^a_i$ and $\tau^b_i$ for a given task $\tau_i$ by duration $d^{rel}_{(i,a),(i,b)}$, as shown in Equation 14. A subtask deadline $D^{abs}_{(i,j)}$ upperbounds the duration between the epoch and the finish time $f^j_i$ of subtask $\tau^j_i$ by duration $d^{abs}_{(i,j)}$, as shown in Equation 15.

$$D^{rel}_{(i,a),(i,b)} : \left(f^b_i - s^a_i \leq d^{rel}_{(i,a),(i,b)}\right) \tag{14}$$

$$D^{abs}_{(i,j)} : \left(f^j_i \leq d^{abs}_{(i,j)}\right) \tag{15}$$

These deadline constraints provide additional expressiveness to encode binary temporal constraints relating tasks in the manufacturing process. For example, these constraints may be used to specify that a sequence of subtasks related to sealant application must be completed within a half hour after opening the sealant container. These types of constraints are commonly included in AI and operations research scheduling models [4], [10], [22], [34].

We also extend the model to include shared memory resources. Each subtask $\tau^j_i$ requires that a set of shared memory resources $R^j_i$ be utilized in performing that subtask (e.g. for memory shared among multiple processors), where $\boldsymbol{R}$ is the set of all shared memory resources. In the manufacturing analogy, a resource $r \in R^j_i$ corresponds to a region of space in the factory that must physically be unoccupied for an agent to execute a subtask in that location. These shared memory resources are used to encode hard spatial constraints that prohibit agents from working too closely in physical proximity.

Next we describe how the Tercio Task Sequencer leverages the structure of the well-formed task model to compute a schedule that satisfies upperbound and lowerbound temporal constraints, as well as spatial-proximity restrictions. To our knowledge, this is the first real-time scheduling method for multi-processor systems that tests the schedulability of non-preemptive, self-suspending tasks where multiple tasks have more than one self-suspension with shared memory resource constraints [18], [19], [25].

### *B. Multi-agent Task Sequencer Pseudocode*

The Tercio multi-agent task sequencer pseudo-code is presented in Figure 3. The algorithm takes as input the user-specified makespan *cutoff* and a task set $\boldsymbol{\tau} = \{\tau_i | i \in \{1, \ldots, n\}\}$ that encodes the agent assignments $A^j_i$, shared memory resource constraints $R^j_i$, and deadline constraints. The algorithm returns a valid task sequence for each agent, if one can be found, and an upperbound on the time to complete all tasks. This upperbound on completion time is compared to *cutoff* to test schedulability of a well-formed task model.

The real-time systems community has developed analytic schedulability tests as well as methods for testing schedulability through simulation, of both fixed priority (e.g.Rate-Monotonic [19]) and dynamic priority (e.g. Earliest Deadline First [29]) algorithms. Tercio schedules through simulation using a dynamic priority method. Our key innovation leverages the use of a scheduling policy that guides the behavior of the scheduler to reduce schedule anomalies due to self-suspensions. We first describe the dynamic priority heuristics used to guide scheduling behavior, and then describe the online temporal consistency check that enables dynamic scheduling of tasks, while guaranteeing inter-subtask temporal constraints are satisfied.

*1) Guiding the Behavior of the Scheduler:* Our approach to scheduling tasks bears resemblance to prior art that uses heuristic methods to alter the behavior of self-suspending systems to tightly bound the makespan for the task set [19], [23], [30]. We introduce four heuristics for the scheduling of subtasks (applied in Line 3) that address the types of schedule bottlenecks that hinder efficient execution in the well-formed task model augmented with shared memory resources.

**Dynamic Priority Heuristics.** First, we want as many agents as possible to work concurrently; however, the scheduling of one agent restricts the available subtask options for other agents. As such, we introduce a heuristic function, $\pi_A(\tau^j_i)$, that prioritizes a subtask $\tau^j_i$ assigned to agent $A^j_i$ in inverse proportion to the number of available subtasks assigned to $A^j_i$.

Second, when subtasks share a resource (i.e. are located in close, physical proximity) agents assigned to those tasks may have to idle due to resource contention. We introduce a heuristic function, $\pi_R(\tau^j_i)$, that eases resource contention by prioritizing subtasks that require higher-demand resources over subtasks that require lower-demand resources.

Third, consider a scenario where two subtasks, $\tau^j_i$ and $\tau^{j+k}_i$ with agent assignments $A^j_i$ and $A^{j+k}_i$. If $A^j_i$ does not execute $\tau^j_i$ in a timely manner, then $A^{j+k}_i$ will idle. We introduce a heuristic function, $\pi_P(\tau^j_i)$, to ease bottlenecks due to inter-agent precedence constraints. The heuristic prioritizes the execution of a subtask $\tau^j_i$ in proportion to the number

of following subtasks in task $\tau_i$ that are assigned to other agents.

Fourth, we observe that humans working alongside robots are distracted by robots that move large distances quicklyand frequently. The worker may also distrust or fear a robotic-system that behaves in this erratic manner. For this reason, we introduce a heuristic function, $\pi_D(\tau_i^j)$, that prioritizes a subtask according to the spatial distance between the subtask's physical resource and the current location of the agent assigned to that subtask.

These four heuristics can be combined in various formulations depending on *a priori* knowledge of what kind of bottlenecks govern the system. For the results we present in this paper, we use a multi-tiered sort with $\pi_A(\tau_i^j)$ first, $\pi_P(\tau_i^j)$ second, $\pi_R(\tau_i^j)$ third, and $\pi_D(\tau_i^j)$ fourth. We find this setup performs well against our model of real-world assembly manufacturing problems. Finally, our scheduling policy requires that an agent not idle if there is an available subtask, unless executing that subtask will violate a deadline constraint (Line 5). This condition is checked via an online consistency test that we describe next.

*2) Multi-Agent Online Consistency Check:* During the scheduling simulation, we perform an online consistency check, which we call the Multiprocessor Russian Dolls Test, that ensures that scheduling of $\tau_i^j$ at time $t$ will not cause $\tau_x^y$ to violate a temporal or shared memory resource constraint. This work extends our single-agent online consistency test [12] to handle multiple agents and shared memory resource constraints. Our online consistency test is a variant of resource edge-finding [17], [33]. The purpose of edge-finding is to determine whether an event must or may execute before or after a set of activities [3]. We develop an analytical, polynomial-time approach that we use online to determine whether a subtask $\tau_i^j$ can feasibly execute before a set of other subtasks given the deadline constraints. To our knowledge, our approach is the first to leverage the structure of the self-suspending task model to perform fast edge checking in polynomial time.

Our well-formed self-suspending task model includes absolute deadlines, $D_{(i,b)}^{abs}$ relating a subtask $\tau_i^b$ to the plan epoch time, and inter-subtask deadline, $D_{(i,j),(i,b)}$, from $\tau_i^j$ to $\tau_i^b$. We introduce a definition for the an *active deadline*, which we use to describe our online consistency test.

*Definition 1: Active Deadline* - Consider an intra-task deadline $D_{(i,j),(i,b)}^{rel}$, or an absolute deadline $D_{i,j}^{abs}$. An intra-task deadline is considered *active* between $0 \leq t \leq \min\left(f_i^j, D_{(i,j),(i,b)}^{rel}\right)$, and an absolute deadline is considered *active* between $0 \leq t \leq \min\left(f_i^j, D_{i,j}^{abs}\right)$. $\boldsymbol{D^{active}}$ is the set of all active deadlines.

We readily formulate our online consistency test as a constraint satisfaction problem, as shown in Equations 16-18, where $\boldsymbol{D^*}$ is union of all active deadlines and the deadline we are considering activating. Equation 16 determines whether a subtask executed by an agent $a$ and using resource $r$ can be scheduled without resulting in the immediate or eventual violation of an active deadline. $\xi_a(i,j,k,a)$ and $\beta_a(i,j,k,a)$ refer respectively the next and last subtask to which agent $a$ is assigned to in $\{\tau_i^j, \ldots, \tau_i^k\}$. $\xi_r(i,j,k,r)$ and $\beta_r(i,j,k,r)$ refer respectively the next and last subtask in $\{\tau_i^j, \ldots, \tau_i^k\}$ that require resource $r$. $d_i^{\beta_a(i,j,k,a)}$ and $d_i^{\beta_r(i,j,k,r)}$ are the absolute deadlines of the last subtasks assigned to agent $a$ and that require resource $r$, respectively.

$$\begin{aligned}&\left(\delta_{(i,j),(i,k)}^{a} \geq d_x^{\beta_a(x,y,z,a)} - t \vee \delta_{x,y:z}^{a} \geq d_x^{\beta_a(x,y,z,a)} - t\right)\\ \wedge &\left(\delta_{(i,j),(i,k)}^{r} \geq d_x^{\beta_r(x,y,z,r)} - t \vee \delta_{x,y:z}^{r} \geq d_x^{\beta_r(x,y,z,r)} - t\right),\\ &\forall D_{(i,j),(i,k)}^{rel} \in \boldsymbol{D^*}, \forall D_{(i,k)}^{abs} \in \boldsymbol{D^*}, \forall a, \forall r\end{aligned} \tag{16}$$

$$\begin{aligned}\delta_{(i,j),(i,k)}^{a} = d_i^{\beta_a(i,j,k,a)} - t - \Bigg(&C_i^{\beta_a(i,j,k,a)}\\ &+ \sum_{\psi=\xi_a(i,j,k,a)}^{\beta_a(i,j,k,a)-1}\left(C_i^{\psi} + E_i^{\psi}\right)\Bigg)\end{aligned} \tag{17}$$

$$\begin{aligned}\delta_{(i,j),(i,k)}^{r} = d_i^{\beta_r(i,j,k,r)} - t - \Bigg(&C_i^{\beta_r(i,j,k,r)}\\ &+ \sum_{\psi=\xi_r(i,j,k,r)}^{\beta_r(i,j,k,r)-1}\left(C_i^{\psi} + E_i^{\psi}\right)\Bigg)\end{aligned} \tag{18}$$

$\delta_{(i,j),(i,k)}^{a}$ (Equation 17) and $\delta_{(i,j),(i,k)}^{r}$ (Equation 18) are the *slack time* for agent $a$ and resource $r$, respectively, in relation to an active deadline $D_{(i,j),(i,k)}^{rel}$ or $D_{(i,k)}^{abs}$. An agent's or resource's slack time for a deadline is a bound on the amount of time that the agent or resource may feasibly commit to the execution of subtasks not associated with that deadline.

Temporal feasibility is ensured if, for each deadline, we can nest the time before one deadline within the slack of another (or vice versa) for all agents and resources associated with that deadline. Our multi-agent sequencer uses a polynomial-time version of this online consistency test to evaluate the feasibility of scheduling subtasks. The complexity of this consistency check is $O(n(a+r))$ where $n$ is the number of tasks, $a$ is the number of agents, and $r$ is the number of resources.

## VI. Evaluation and Discussion

In this section, we empirically validate that Tercio is fast and produces near-optimal solutions for the multi-agent task assignment and scheduling problem with temporal and spatial-proximity constraints. Results are generated on an Intel Core i7-2820QM CPU 2.30GHz.

### A. Generating Random Problems

We evaluate the performance of Tercio on randomly generated problems that simulate multi-agent construction of a large structural workpiece, such as an airplane fuselage or wing. Task times are generated from a uniform distribution in the interval $[1, 10]$. Approximately $\frac{1}{4}$ of the subtasks are related via a nonzero wait duration (lowebound constraint) drawn from the interval $[1, 10]$, and approximately $\frac{1}{4}$ of the subtasks are related via an upperbound temporal deadline generated randomly to another subtask. The upperbound of each intra-task and subtask deadline constraint is drawn from

**TERCIO-SEQUENCER**($\boldsymbol{\tau}$, *cutoff*)

$t \leftarrow 0$ ▷ set simulation time to zero
**while** true **do**
  $availableTasks \leftarrow$ get and sort subtasks ready for execution at time $t$ according to dynamic priority heuristics
  **for** $k \leftarrow 1$ to the number of $availableTasks$ **do**
    **if** executing $k^{th}$ available subtask at $t$ will not violate temporospatial constraints **then**
      $seq \leftarrow$ schedule $k^{th}$ available subtask
    **end if**
  **end for**
  $t \leftarrow t + 1$ ▷ increment simulation time
  **if** all tasks executed **then** break;
  **end if**
**end while**
$makespan, seq \leftarrow$ extract multiagent schedule
**if** $makespan \leq cutoff$ **then return** $makespan, seq$
**end if**
**return** false

Fig. 3. Pseudo-code for the Tercio Multi-agent Task Sequencer.

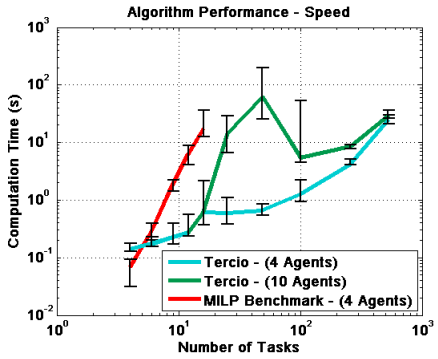


Fig. 4. Computation speed as a function of the number of work packages and the number of agents. Tercio can solve problems up to hundreds of tasks in seconds versus the dozen tasks solvable by the MILP benchmark before it times out.

a normal distribution with mean set to the tightest possible bound. For values less than the mean, we simply redraw a new deadline value. We vary the number of subtasks $m_i$ within each task $\tau_i$ from a uniform distribution in the interval $\left[\frac{n}{4}, \frac{5n}{4}\right]$. Lastly, physical locations of a subtask are drawn from a uniform distribution in $[1, \sum_{i=1}^{n} m_i]$ where $m_i$ is the number of subtasks in $\tau_i$ and $\tau_i$ is the number of tasks in $\boldsymbol{\tau}$.

### B. Computation Speeds

In Fig. 4 we evaluate scalability and computational speed of Tercio. We show the median and quartiles of computation time for 25 randomly generated problems, spanning 4 and 10 agents, and 5 to 500 tasks (referred to as subtasks in the well-formed model). For comparison, we show computation time for solving the full MILP formulation of the problem, described in Section III. Tercio is able to generate flexible schedules for 10 agents and 500 tasks in seconds. This is a significant improvement over prior work [7], [8], [32], [16], which report solving up to 5 agents (or agent groups) and 50 tasks in seconds or minutes.

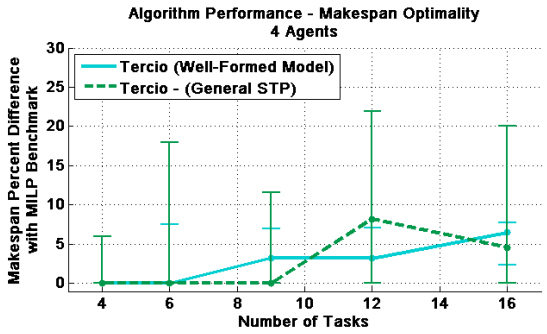


Fig. 5. Tercio suboptimality in makespan for problems with 4 agents. Tercio achieves less than a 10% overestimate in makespan on average for problem sizes at which true optimal can be computed.

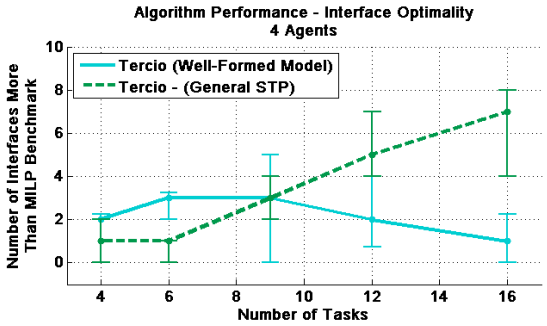


Fig. 6. Tercio suboptimality in number of interfaces for problems with 4 agents. Tercio solutions contain an acceptable number of additional interfaces for problem sizes at which true optimal can be computed.

### C. Optimality Levels

Our fast computation relies on the known structure of our well-formed task model, but it is desirable to be able to take as input general sets of temporal (STP) constraints. General STPs can be reformulated into well-formed task models by adding and tightening well-formed temporal constraints to make the constraints that violate the well-formed model redundant. We present results with both random problems that are well-formed and problems that are general but have been reformulated into a well-formed task model.

In Figures 5-6 we show that Tercio is often able to achieve makespans within 10% of the optimal makespan for both well-formed models and general STPs; Tercio is able to produce less than four additional interfaces when compared to the optimal task allocation for well-formed models and less than eight additional interfaces for general models. We are unable to measure the suboptimality gap for larger problems due to the computational intractability of the full MILP. The purpose of Tercio is to solve the problem of scheduling with tens of agents and hundreds of tasks; as we can see in Figure 5, Tercio tightly tracks the optimal solution for problems tested.

### D. Robot Demonstration

We demonstrate the use of Tercio to plan the work of two KUKA Youbots. Video can be found at `http://tiny.`

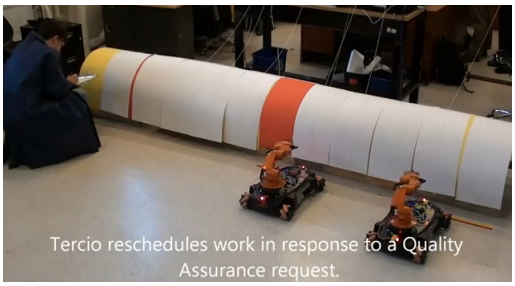


Fig. 7. Hardware demonstration of Tercio where two KUKA Youbots build a mock airplane fuselage alongside a human quality assurance agent.

cc/2aytrw. The two robots are working to assemble a mock airplane fuselage. The robots perform their subtasks at specific locations on the factory floor. To prevent collisions, each robot reserves both the physical location for its subtask, as well as the immediately adjacent subtask locations. Initially, the robots plan to split twelve identical tasks in half down the middle of the fuselage. After the robots finish their first subtasks, a person requests time to inspect the work completed on the left half of the fuselage. In the problem formulation, this corresponds to adding a resource reservation for the left half of the fuselage for a specified period of time. Tercio replans in response to the addition of this new constraint, and reallocates the work among the robots in a near-optimal manner to make productive use of both robots and to keep the number of interfaces low.

## VII. Conclusion

We present Tercio, a task assignment and scheduling algorithm that is made efficient through a fast, satisficing, incomplete multi-agent task sequencer inspired by real-time processor scheduling techniques. We use the fast task sequencer in conjunction with a MILP solver to compute an integrated multi-agent task sequence that satisfies precedence, temporal and spatial-proximity constraints. We demonstrate that Tercio generates near-optimal schedules for up to 10 agents and 500 tasks in less than 20 seconds.

## References


[1] Gurobi optimizer version 5.0, 2012.

[2] 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. International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, 2005.

[3] P. Baptiste and C. L. Pape. Edge-finding consraint propagation algorithms for disjunctive and cumulative scheduling. In *In Proc. Workshop of the U.K. Planning and Special Interest Group*, 1996.

[4] D. Bertsimas and R. Weismantel. *Optimization over Integers*. Dynamic Ideas, 2005.

[5] L. Brunet, H.-L. Choi, and J. P. How. Consensus-based auction approaches for decentralized task assignment. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2008 (AIAA-2008-6839).

[6] D. A. Castañón and C. Wu. Distributed algorithms for dynamic reassignment. In *IEEE Conference on Decision and Control (CDC)*, volume 1, pages 13–18, December 2003.

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

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

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

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

[11] U. C. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Euromicro Technical Committee on Real-Time Systems*, 2003.

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

[13] J. N. Hooker. A hybrid method for planning and scheduling. In *Proc. Carnegie Mellon University Research Showcase*, 2004.

[14] J. N. Hooker. An improved hybrid MILP/CP algorithm framework for the job-shop scheduling. In *Proc. IEEE International Conference on Automation and Logistics*, 2009.

[15] V. Jain and I. E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. 13, 2001.

[16] A. Kushleyev, D. Mellinger, and V. Kumar. Towards a swarm of agile micro quadrotors. *Robotics: Science and Systems (RSS)*, July 2012.

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

[18] K. Lakshmanan, S. Kato, and R. R. Rajkumar. Open problems in scheduling self-suspending tasks. In *Proc. Real-Time Scheduling Open Problems Seminar*, 2010.

[19] K. Lakshmanan and R. R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2010.

[20] C. Liu and J. H. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *Proc. Real-Time Systems Symposium (RTSS)*, 2009.

[21] C. Liu and J. H. Anderson. Improving the schedulability of sporadic self-suspending soft real-time multiprocessor task systems. In *Proc. Conference on Real-Time Computing Systems and Applications (RTCSA)*, 2010.

[22] N. Muscettola, P. Morris, and I. Tsamardinos. Reformulating temporal plans for efficient execution. In *Proc. Principles of Knowledge Representation and Reasoning (KR&R)*, 1998.

[23] R. R. Rajkumar. Dealing with self-suspending period tasks. Technical report, IBM Thomas J. Watson Research Center, 1991.

[24] W. Ren, R. W. Beard, and T. W. McLain. Coordination variables, coordination functions, and cooperative timing missions. *AIAA Journal on Guidance, Control, and Dynamics*, 28(1):150–161, 2005.

[25] P. Richard. On the complexity of scheduling real-time tasks with self-suspensions on one processor. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, 2003.

[26] F. Ridouard and P. Richard. Negative results for scheduling independent hard real-time tasks with self-suspensions. 2006.

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

[28] D. E. Smith, F. Frank, and A. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15, 2000.

[29] J. Stankovic, M. Supri, M. D. Natale, and G. Buttazzo. Implications of classical scheduling resuts for real-time systems. 1995.

[30] J. Sun and J. Liu. Synchronization protocols in distributed real-time systems. In *Proc. International Conference on Distributed Computing Systems*, 1996.

[31] S. J. R. Tal Shima and P. Chandler. UAV team decision and control using efficient collaborative estimation. In *Proc. American Control Conference (ACC)*, volume 6, pages 4107–4112, June 2005.

[32] W. Tan. Integration of process planning and scheduling - a review. 11:51–63, 2000.

[33] 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, Oct. 2005.

[34] R. J. Wilcox, S. Nikolaidis, and J. A. Shah. Optimization of temporal dynamics for adaptive human-robot interaction in assembly manufacturing. In *Proc. Robotics: Science and Systems (RSS)*, 2012.