MIT Open Access Articles

Optimization of Temporal Dynamics for Adaptive Human-Robot Interaction in Assembly Manufacturing

The MIT Faculty has made this article openly available. ***Please share*** how this access benefits you. Your story matters.

Citation: Wilcox, Ronald, Stefanos Nikolaidis, and Julie Shah. "Optimization of Temporal Dynamics for Adaptive Human-Robot Interaction in Assembly Manufacturing." Robotics: Science and Systems VIII (July 9, 2012).

As Published: http://dx.doi.org/10.15607/RSS.2012.VIII.056

Publisher: Robotics: Science and Systems Foundation

Persistent URL: http://hdl.handle.net/1721.1/116089

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of use: Creative Commons Attribution-Noncommercial-Share Alike

Massachusetts Institute of Technology

Optimization of Temporal Dynamics for Adaptive Human-Robot Interaction in Assembly Manufacturing

Ronald Wilcox, Stefanos Nikolaidis, and Julie Shah
Massachusetts Institute of Technology
Cambridge, Masachusetts 02139
Email: rjwilcox@mit.edu; snikol@mit.edu; julie_a_shah@csail.mit.edu

***Abstract*—Human-robot collaboration presents an opportunity to improve the efficiency of manufacturing and assembly processes, particularly for aerospace manufacturing where tight integration and variability in the build process make physical isolation of robotic-only work challenging. In this paper, we develop a robotic scheduling and control capability that adapts to the changing preferences of a human co-worker or supervisor while providing strong guarantees for synchronization and timing of activities. This innovation is realized through dynamic execution of a flexible optimal scheduling policy that accommodates temporal disturbance. We describe the Adaptive Preferences Algorithm that computes the flexible scheduling policy and show empirically that execution is fast, robust, and adaptable to changing preferences for workflow. We achieve satisfactory computation times, on the order of seconds for moderately-sized problems, and demonstrate the capability for human-robot teaming using a small industrial robot.**

I. Introduction

Traditionally, industrial robots in manufacturing and assembly perform work in isolation from people. When this is not possible, the work is done manually. We envision a new class of manufacturing processes that achieve significant economic and ergonomic benefit through robotic assistance in manual processes. For example, mechanics in aircraft assembly spend a significant portion of their time retrieving and staging tools and parts for each job. A robotic assistant can provide productivity benefit by performing these non-value-added tasks for the worker. Other concepts for human and robot co-work envision large industrial robotic systems (examples in Fig. 1) that operate in the same physical space as human mechanics.

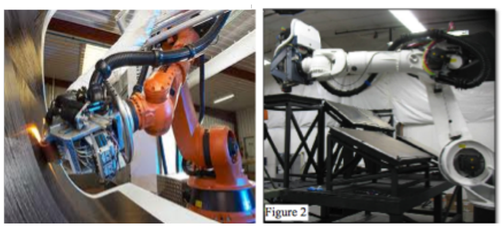

Fig. 1. (Left) Coriolis Composite Placement Robot; (Right) ElectroImpact Robotic Drilling Arm

This work is supported by Boeing Research and Technology and ABB, Inc. Mechatronics US Corporate Research.

In this paper, we present a robotic scheduling and control capability for human-robot collaborative work that addresses two key challenges in the manufacturing environment. First, preferences about task completion are prone to change since the ordering and timing of activities in many manual processes are left to the discretion of the human workers. A high level of adaptability and robustness must therefore be built into any robotic system that works in close collaboration with people.

Second, human and robotic work in manufacturing and assembly must meet hard scheduling constraints, including pulse rates between build stations and flow rates for end-to-end assembly. The changing preferences of a human co-worker or supervisor must be accommodated while preserving strong guarantees for synchronization and timing of activities.

Our approach generalizes from dynamic scheduling methods [2, 6, 13] first developed to perform scheduling onboard a deep space satellite [6]. Dynamic scheduling is domain independent and has been successfully applied to scheduling within the avionics processor of commercial aircraft [13], autonomous air vehicles [12], robot walking [3], and recently, human-robot teaming [8, 11]. We leverage prior art that addresses efficient real-time scheduling of plans whose temporal constraints are described as Simple Temporal Problems (STPs) [2, 6, 13]. STPs compactly encode the set of feasible scheduling policies for plan events that are related through simple interval temporal constraints. Temporal flexibility in the STP provides robustness to disturbances at execution.

We make use of this simple yet powerful framework to model joint human-robot work as a Simple Temporal Problem with soft constraints (called preferences). The preferences encode person-specific workflow patterns and human operator input for suggested workflow. Simple Temporal Problems with Preferences (STPPs) have been studied previously [4, 5, 7] for weakest-link optimization criteria, but these solution techniques do not generalize to optimization criteria relevant to manufacturing applications. Alternatively, an STPP with arbitrary objective function may be formulated and solved as a non-linear program (NLP), where the solution is an assignment of execution times to each event in the plan. This approach results in brittle solutions; any disturbance in execution time requires time-consuming re-calculation of the schedule.

In this work, we describe a robotic scheduling capability that leverages the strengths of STP and NLP solution methods: flexibility in execution and optimization of arbitrary objective functions, respectively. We present the Adaptive Preferences Algorithm (APA) that uses the output of a non-linear program solver to compute a flexible optimal scheduling policy that accommodates temporal disturbance. The algorithm also supports on-the-fly optimization in response to changing preferences. In Section II we discuss examples of human-robot interaction that motivate our work. Next, in Section III we briefly review STP dynamic scheduling and optimization of preferences, since this prior art forms the basis for our approach. In Sections IV and V we present the Adaptive Preferences Algorithm (APA) and its evaluation. We show empirical results indicating both that APA provides significant robustness to temporal disturbance and that a robot using APA can adaptively schedule its actions over a horizon of 20 activities with sub-second speed. Finally in Section VI, we apply APA to perform human-robot teaming using a small industrial robot.

II. Motivating Examples

In this section, we discuss two types of applications that motivate our work: one-on-one robotic assistance for a worker, and single-operator orchestration of robot teams.

A. Robotic assistant to assembly mechanic

We aim to develop a capability that supports efficient and productive interaction between a worker and a robotic assistant, such as the FRIDA robot shown in Fig. 2. Although important aspects like tolerances and completion times are well defined, many details of assembly tasks are left largely up to the mechanic.

Fig. 2. ABB FRIDA robot acting as a robotic assistant

Assembly of airplane spars is one example of a manual process where mechanics develop highly individualized styles for performing the task. Fig. 3 shows a mechanic assembling a spar. The spar is composed of two pieces that must be physically manipulated into alignment. After alignment, wet sealed bolts are hammered into pre-drilled holes and are fastened with collars. Excess sealant is removed, and the collars are re-torqued to final specifications. Sequencing of these tasks is flexible, subject to the constraint that the sealant is applied within a specified amount of time after opening it.

A robot such as FRIDA can assist a mechanic by picking bolts and fasteners from a singulator, rotating them in front of a stationary sealant end-effector, and inserting them into the bores. This would allow the mechanic to focus on wiping

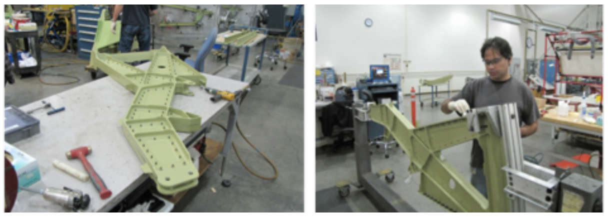

Fig. 3. Spar assembly is a manual process that could be improved by a robotic assistant (image courtesy of Boeing Research and Technology)

sealant, hammering the bolts, and placing and torquing the collars. This division of labor would provide productivity benefit through parallelization of tasks.

Our aim is to enable a robotic assistant to adapt to person-specific workflow patterns. If most mechanics like to hammer all bolts before torquing collars, the robot will support this approach by placing all bolts in a pattern that anticipates the mechanic's actions. When the robot is paired with a mechanic that instead prefers to hammer and torque the collar for each bolt as it is placed, the robot will quickly perceive this difference and reoptimize its schedule to converge on a turn-taking pattern with the mechanic. The robot will adapt according to the mechanic's preferences, subject to the constraint that the sealant is utilized within the specified window.

B. Robotic Team Orchestration

We also aim for our capability to enable a single operator to direct a team of robots, while ensuring that hard scheduling deadlines such as mandated flow rates are met. Work will be shifted according to operator preferences through fast re-computation of the robots' schedule, while preserving guarantees that assembly will finish within specified deadlines.

Unscheduled maintenance is frequently required for new, specialized robots that perform traditionally manual work, including drilling and composite lay-down. Current practices require all robots halt while one robot is repaired, or while a quality assurance agent inspects the work. These slowdowns and subsequent workflow recalculations cost the facilities hours of productivity that can be avoided with the quick re-computation and flexible schedules provided by our approach.

III. Background

The robotic scheduling and control capability we develop builds on previous work in STP dynamic scheduling and optimization of STPs with Preferences (STPPs).

A. Dynamic Scheduling of STPs

A Simple Temporal Problem (STP) [2] consists of a set of executable events, X. These events are connected via binary temporal constraints (intervals) b_{ij} that indicate a range for the temporal distance between events X_i and X_j. Fig. 4 (left) presents the *constraint form* graphical depiction of a binary temporal constraint. Events are represented as nodes, and the temporal constraint is depicted with an arrow and assigned interval.

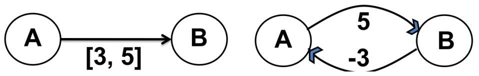

Fig. 4. **Left:** Constraint form representation; indicates that event B must occur at least 3 time units after event A but no more than 5 time units after it, or $3 < X_B - X_A < 5$, **Right:** Distance graph representation; indicates the same interval as the constraint, but yields two equivalent inequalities, $X_B - X_A < 5$ and $X_A - X_B < -3$

The STP constraint form may be mapped to an equivalent *distance graph form* to support efficient inference [2]. Fig. 4 (right) presents the distance graph form of the temporal constraint. The interval upperbound is mapped to a positive arc from X_A to X_B, and the lowerbound is mapped to a negative arc from X_B to X_A.

A *solution* to an STP is a time assignment to each event such that all constraints are satisfied. An STP is said to be *consistent* if at least one solution exists. Checking an STP for consistency can be cast as an all-pairs shortest path problem. The STP is consistent if and only if there are no negative cycles in the all-pairs distance graph. This check can be performed in $O(n^3)$ time by applying the Floyd-Warshall algorithm [2].

The all-pairs shortest path graph of a consistent STP is also a *dispatchable form* of the STP, enabling flexible real-time scheduling [6]. The dispatchable STP provides a compact representation of the set of feasible schedules. Dynamic scheduling of the dispatchable STP provides a strategy that schedules events online just before they are executed, with a guarantee that the resulting schedule satisfies the temporal constraints of the plan. Scheduling events on-the-fly allows the robot to adapt to temporal disturbance associated with past events through fast linear-time constraint propagation. More formally, a network is *dispatchable* if for each variable X_A it is possible to arbitrarily pick a time t within its timebounds and find feasible execution windows in the future for other variables through one-step constraint propagation of the X_A temporal commitment. Dispatching of STPs is described in more detail with examples in Section IV.

B. Prior art in STP Preference Optimization

An STP with Preferences (STPP) [5] is a Simple Temporal Problem with the addition of soft binary constraints, or preference functions, $f_{b_{ij}}(t)$ relating the temporal durations between events. The global preference function, F, of an STPP represents the objective function derived from the preference values based on a time assignment to each event. An optimal solution to the STPP is consistent with the temporal constraints b_{ij} and optimizes the global preference function F.

Preferences provide an expressive and natural framework for encoding human input. A supervisor may apply preference functions to specify the most effective timing for an activity without providing hard constraints that lead to schedule brittleness. For example, a supervisor may specify the desire for painting to take four hours, but allow any time up to six hours as acceptable.

Preference functions may also be applied to encode statistical information about likely execution times for human actions, so as to drive the robot schedule to conform to human behavior. Data mining of typical human workflows can provide the statistical information necessary to infer preference functions. In addition, preference functions may be used to model the effect of implicit communications; recent studies indicate that gestures induce preferences over execution sequence and timing in human teams [9]. This effect may be reproduced in human-robot teams using preference functions.

STPPs were originally developed to perform scheduling for Earth observation satellites [4]. Scientists were asked to provide preferences indicating the most effective times for them to access the satellite. The STPP framework was applied to solve the scheduling problem, using an objective function that maximized the preferences of the least satisfied scientist. Solution methods, including a slow constraint propagation technique and fast binary chop method [7], have been designed for this weakest link optimization criterion but do not readily generalize to other objective functions.

Fairness is not a concern in the optimization of a manufacturing process. It is acceptable to sacrifice one interval's preference value to improve the preference values for many other intervals (e.g. slow down one robot so that it does not block the path for the other robots). For example, for many manufacturing applications, an approach that optimizes the STPP with respect to the sum of preference values, $\sum_{b_{ij}} f_{b_{ij}}(t)$, is more appropriate. An STPP with arbitrary objective function may be formulated and solved as a nonlinear program (NLP), where the solution is an assignment of execution times to each event in the plan. However, this approach results in brittle solutions; any disturbance in execution time requires time-consuming re-calculation of the schedule. In the next sections, we present a method for computing a temporally flexible optimal scheduling policy that leverages the strengths of STP and NLP solution methods. The Adaptive Preferences Algorithm computes a flexible optimal scheduling policy that accommodates fluctuations in execution time and supports robust online optimization in response to changing preferences.

IV. Problem Formulation

The Adaptive Preferences Algorithm (APA) takes as input a Simple Temporal Problem with Preferences (STPP), composed of

- a set of variables, $X_1, ...X_n$, representing executable events,
- a set of binary temporal constraints of the form b_{ij} encoding activity durations and qualitative and quantitative temporal relations between events X_i and X_j,
- a set of preferences functions of the form $f_{b_{ij}}(t)$ encoding preference values over the temporal interval b_{ij} , and
- a global objective criterion F defined as a function of the preferences functions $f_{b_{ij}}(t)$. We use $F = \sum_{b_{ij}} f_{b_{ij}}(t)$ for prototyping of the described manufacturing applications, although note APA generalizes to other forms of the objective function.

The output of the algorithm is a *dispatchably optimal* (DO) form of the STPP that supports fast dynamic scheduling. We define an STPP as dispatchably optimal if it is possible to maximize the global preference function F through the following procedure: for each variable X_i it is possible to arbitrarily pick a time t within the DO form's timebounds and find feasible execution windows in the future for other variables through fast one-step constraint propagation of the X_i temporal commitment.

Notice that the proposed problem may be formulated as a non-linear optimization problem to solve for event execution times. This approach provides a solution that is brittle to disturbance, requiring recomputation when an event does not execute at precisely the specified time. In contrast, our approach compiles a temporally flexible optimal scheduling policy that accommodates fluctuations in execution time. This method leverages the insight that there are many potential schedules that are consistent with an optimal time assignment to preference functions. Section IV-A presents the compilation algorithm that computes the DO form for the STPP. Section IV-B presents the dispatcher algorithm that generates a schedule using the STPP DO form, and supports robust online reoptimization in response to changing preferences.

A. Compiler for STPP DO Form

The Compiler takes as input a STPP composed of events X_i, constraints b_{ij}, and preference functions $f_{b_{ij}}(t)$. It then reformulates and optimizes the STPP as a non-linear program. The resulting optimal timestamps are used to modify the network so that intervals with preference functions are tightened to the values returned by the optimizer; intervals without preferences retain their flexibility. After an all-pairs-shortest-path computation, the resulting output is a DO plan, which encodes a flexible scheduling policy that maximizes the global preference function F subject to the given binary temporal constraints b_{ij}.

Pseudocode for the compilation algorithm is provided in Fig. 5. The first step (Line 1) of **APAcompilePlan** is to compute the all-pairs-shortest-path form of the STP using the Floyd-Warshall algorithm [1]. This process exposes implicit constraints and is necessary to ensure events are scheduled in the proper order with requisite temporal distances between events. The result of the APSP computation is a fully-connected network, with binary constraints relating each pair of events. Many of the added constraints are redundant and can be removed from the problem (Line 2) without loss of information [6]. Our empirical investigations indicate that pruning of redundant constraints reduces the total number of constraints by 40-50%. The resulting network is the most compact representation of the binary temporal constraints that still contains all feasible solutions present in the original problem [6].

In Line 3, we use the resulting representation as input to a standard, third-party optimization solver [1]. The STPP is formulated as a nonlinear program as follows. Events are encoded as variables with ranges that span the possible

```
function APAcompilePlan(STPP plan)
1.  STP compiled_plan = perform APSP( plan)
2.  compiled_plan = prune redundant edges(compiled_plan)
3.  optimal_execution_times = new NLP Solver(compiled_plan)
4.  given_prefs = gather constraints with preferences (plan);
5.  for(each interval b'{ij} in compiled_plan)
6.    if( there exists a constraint relating events Xi and Xj in given_prefs)
7.      set b'{ij} to difference in optimal_execution_times[Xj-Xi];
8.    end if
9.  end for
10. perform APSP (compiled_plan);
11. return compiled_plan;
```

Fig. 5. Pseudocode for the compilation algorithm

execution times computed by the APSP computation. Binary constraints are formulated as linear inequality constraints relating the variables. For the manufacturing applications we are interested in, the objective function is defined as $\sum_{b_{ij}} f_{b_{ij}}(t)$, the sum of the preference values evaluated across each binary interval constraint. The preference functions are permitted to be nonlinear, resulting in the nonlinear formulation. The solver returns an assignment of event execution times that optimizes the global preference value F subject to the given constraints b_{ij}.

Note that we do not use the output of the nonlinear optimizer directly to set the schedule, as this will provide no robustness to uncertainty and disturbance in the execution. Instead, we use the output as follows to reformulate the STPP and compute a temporally flexible, optimal scheduling policy.

In Line 4, the algorithm iterates through all constraints in the original STPP and makes a list *given_prefs* of those that have preference functions associated with them. Line 5 searches through each constraint b'_{ij} in the partially compiled plan. If b'_{ij} also exists in *given_prefs*, then b'_{ij} is updated, setting both the upper and lower bounds of the constraint to the optimized time of execution (with a small tolerance built in). Finally, in Line 10, the APSP network is computed to expose implicit constraints of the tightened network. The result (Line 11) is a *DO form of the STPP* that preserves temporal flexibility in the network where there is no impact on the time assignments to preference values.

We now walk through an illustrative example for applying the compilation algorithm. Consider the STPP shown in Fig. 6. This network is an all-pairs-shortest path graph (Line 1), with all implicit constraints exposed, and does not contain any redundant constraints (Line 2). Line 3 generates a list containing the following constraints with preference functions: b_{AD} and b_{BC}.

Line 4 creates a solver with variables for each event: A, B, C, D. All six intervals act as inequality constraints (e.g. for interval AC, we have $1 < CA < 5$). The objective function is given by

$$f_{global} = -(D-A)^2 + 15(D-A) - 50 - (B-C)^2 + 2(B-C). \tag{1}$$

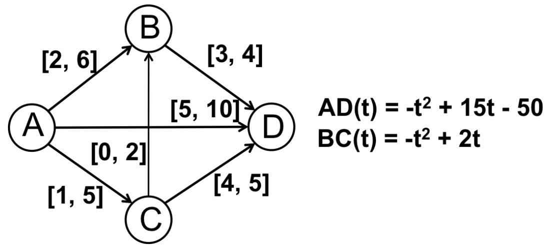

Fig. 6. Example STPP to illustrate compilation

The non-linear program is solved, and yields optimal execution times of $A = 0, B = 3, C = 4, D = 7.5$. Next, we create a new copy of the plan and replace intervals $AD[5, 10]$ with $AD[7.5, 7.5]$ and $BC[0, 2]$ with $BC[1, 1]$. Performing Floyd Warshall on this new network then produces the DO form of the STPP, given in Fig. 7. Any choice of times satisfying the constraints in Fig. 7 produces a solution that maximizes the global preference value f_{global} .

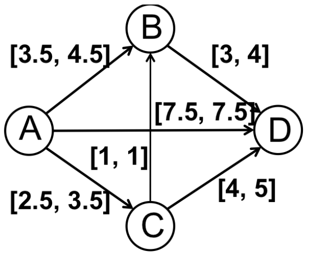

Fig. 7. DO Form of the STPP in Fig. 6

Next, we provide a proof that the STPP-DO form computed by **APAcompilePlan** encodes all feasible solutions in the original STPP that are consistent with a given optimal time assignment to preference functions. In the next section, we discuss the method for dispatching the DO form of the STPP.

Lemma (STPP-DO Form): Given an STPP with an optimal time assignment $t_{b_{ij}} \to f_{b_{ij}}(t)$ to each preference function,

(i) the STPP-DO form encodes all feasible solutions in the STPP that are consistent with $t_{b_{ij}} \to f_{b_{ij}}(t)$, and

(ii) the STPP-DO form supports dispatchable scheduling.

Proof: (i) Lines 5-8 in **APAcompilePlan** tighten constraints in the original STPP, ensuring that any solution satisfies $t_{b_{ij}} \to f_{b_{ij}}(t)$ and achieves the optimal global preference value. Line 10 computes the all-pairs-shortest-path form of the resulting STP, which by definition contains all feasible solutions present in the original problem [6] that also satisfy $t_{b_{ij}} \to f_{b_{ij}}(t)$.

(ii) The resulting STPP-DO form returned at Line 11 is an all-pairs-shortest-path STP, which by definition is also a dispatchable STP [6].

B. Dispatcher

In this section, we present a dispatcher algorithm that supports two functions: the dispatcher (**function1**) generates a schedule using the STPP DO form, and (**function2**) supports robust online reoptimization in response to changing preferences. The dispatcher takes as input an STP *compiled_plan* that encodes the DO form of an STPP S. The dispatcher schedules events on-the-fly just before they are executed while guaranteeing that the resulting schedule satisfies the temporal constraints of the plan. This guarantee is achieved through constraint propagation of temporal commitments to executed events. The output of the dispatcher is an assignment of event execution times that optimizes the STPP S global preference value F, subject to the given temporal constraints of S.

The dispatcher also supports robust online replanning of the DO form, in response to changing preference functions and disturbances in the optimal execution. In these situations, the DO form must be recompiled by calling the algorithm **APAcompilePlan** with the modified STPP S'. As discussed in Section V, this recompilation takes on the order of seconds for moderately-sized real-world problems.

Function1 of the dispatcher is achieved using the standard STP dispatching algorithm [6]. **Function2** is achieved by augmenting the STP dispatching algorithm with two additional methods. The first method triggers recompilation for changing preference functions or deviations from the optimal schedule; the second method runs concurrently to ensure the dispatcher makes progress during recompilation and that the execution schedule satisfies the hard constraints of the STPP S.

Fig. 8 presents the STPP dispatching algorithm. Augmentations to the standard STP dispatching algorithm are highlighted. We walk through the dispatch of the DO plan in Fig. 7 to illustrate the algorithm.

First, in Line 1, all events without predecessors are added to the *Enabled* list. In our example from Fig. 7, event A is initially added to the *Enabled* list. In Line 2, the current time is set to zero. Line 3 contains the first major change to the standard dispatching algorithm. Here a concurrent thread is started to shadow dispatch the STP associated with the *orig_plan*. This thread is used to ensure the dispatcher makes progress during recompilation and that the execution schedule satisfies the hard constraints of *orig_plan*.

Dispatching continues until there are no unexecuted events in the plan (Line 5). If new preference functions are made available or the execution deviates from the optimal scheduling policy, recompilation is triggered (Line 6). Execution control is switched to the STPdispatch thread (Line 7). The *orig_plan* is updated with execution commitments (Line 8) and is compiled (Line 9). Next, execution control is transferred back to STPPdispatch (Line 9), and execution proceeds in Lines 11-25 according to the standard STP dispatching algorithm.

The dispatcher listens for notice of successful event executions from the robot (Line 13). Executed events are recorded in the *Executed* list and removed from the *Enabled* list (Lines 14-17). In Lines 18-21, the dispatcher commands an event to be executed if it is both *enabled*, meaning all predecessors have been executed, and is *alive*, meaning the current time is within the event's feasible window of execution. In our example, at $t = 0$ Event A is enabled and alive, and is executed.

If an event is executed (Line 24), the *Enabled* list is updated (Line 26), and the commitment is propagated through the network *compiled_plan* to update liveness windows for

```
function STPPdispatch(STP compiled_plan, STPP orig_plan)
1.  Enabled = {first event}; Executed = {}
2.  current_time = 0
3.  new thread STPdispatch(orig_plan)
4.  while(size of Executed < number of events)
5.    if(new preferences or deviation from optimal schedule)
6         switch execution control to STPdispatch thread
7.        orig_plan' = replace past intervals with rigid links(orig_plan)
8.        compiled_plan = compilePlan(orig_plan')
9         switch execution control to STPPdispatch
10.   end if
11.   for(each event e in plan)
12.     if(Executed does not contain e)
13.       if( robot signals event has been performed)
14.         add event and execution time to Executed
15.         remove event from Enabled
16.         event_executed = true
17.       end if
18.       if(event e is in Enabled)
19.         Interval bounds = extract 'liveness' bounds for e
20.         if( bounds lowerbound < current _time < bounds upperbound)
21.           signal robot to execute event e
22.         end if
23.       end if
24.       if(event_executed)
25.         event_executed = false;
26.         Enabled = gather enabled events
27.         propagate event commitment to compute liveness windows
28.         wait for next live event or until robot signals an executed event
29.       end if
30.     end if
31.   end for
```

Fig. 8. Pseudocode for the dispatching algorithm

all connected unexecuted events. With event A successfully executed, the liveness windows for events B, C, and D are updated to $B : [3.5, 4.5]$, $C : [2.5, 3.5]$, $D : [7.5, 7.5]$. Once A executes, event C is added to the *Enabled* list. Event C is alive when the current time is between 2.5 seconds and 3.5 seconds. Executing event C at 2.5 seconds then leads to the situation shown in Fig. 9; executed events are shown with squares around letters.

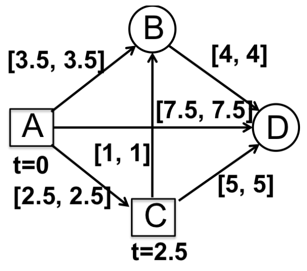

Fig. 9. Dispatching & propagation status after event A has been executed at $t = 0$ and event C has been executed at $t = 2.5$

With events A and C in the *Executed* list, event B becomes enabled and is executed at $t = 3.5$. This commitment is propagated forward, and event D is executed at $t = 7.5$. The resulting schedule maximizes the global preference value and satisfies the temporal constraints of the problem.

The signal-and-response structure (signal in Line 21 and robot response in Line 13) provides robustness in execution by allowing for situations that prevents the robot from completing the task at precisely the specified time. For example, consider event C is commanded at $t = 2.5$ but is delayed at execution till $t = 3.0$. The STPP DO form accommodates this disturbance on-the-fly through one-step constraint propagation. The liveness windows for events B and D are updated to $B : [4.0, 4.0], D : [7.5, 7.5]$.

The compiler and dispatcher presented in Figs. 5 and 8 have been implemented and tested successfully. Section V presents an empirical evaluation of APA, and Section VI describes a robot demonstration applying APA to one-to-one human-robot teaming.

V. Empirical Evaluation

The STPP DO form is designed to be temporally flexible, reducing the impact of disturbance on the schedule. In this section, we empirically investigate the benefit of this flexibility in two ways and compare the results to the non-linear programming (NLP) solution. We also present computation times for on-demand recompilation of the plan, showing that a robot using APA can quickly adapt its schedule in response to changing preferences.

Empirical results are produced using a random problem generator that creates structured problems in the same manner as prior art [10, 14]. The generator takes as input the number n of events, the number of user-specified constraints c, and the set P of preference functions. Each temporal constraint relating plan events is generated by randomly selecting two events from an array and connecting them with a binary interval constraint. Constraint upper and lower bounds are set randomly and then scaled by the difference in array indices between the two events. This creates a network that has a natural structure, with more distant events related through longer temporal durations than local events. Each preference function in P is assigned to a binary constraint in the order the constraints are generated. Following the precedence of previous work in STPPs [7], we consider preference functions of constant, linear, and quadratic form only. Only positive-valued preference functions are permitted. A randomized multiplier is applied to distinguish relative importance among preference functions. The output of the generator is an STPP, which is provided as input to the compiler. The APA compiler, dispatcher, and random problem generator are implemented in Java, and non-linear (here, quadratic) programs are solved using the Java implementation of Gurobi [1]. Results are generated using an Intel Core i7-2620M 2.70 GHz Processor.

First we run simulations to evaluate the cumulative time a robot spends re-computing the schedule in response to frequent small disturbances, for example, from a human co-worker that does not precisely follow the optimal scheduling policy. This measure represents the total execution time the robot spends unresponsive to the human co-worker's preferences for workflow. Fig. 10 presents results showing the worst-case cumulative compilation time for randomly-generated structured problems, in response to frequent small disturbances in the optimal schedule. Each data point signifies the average and standard deviation across fifty randomly generated problems. Results were computed for problem sizes ranging from 25 to 250 events. The number of preference functions was set at 20% the number of events, based on the observation that real-world problems typically have many fewer preference functions than events. Cumulative compilation time for the inflexible NLP approach scales with the number of events in the plan, whereas the STPP DO approach scales with the number of preference functions. The result is that the STPP DO form provides on average an 80% reduction in cumulative compilation time.

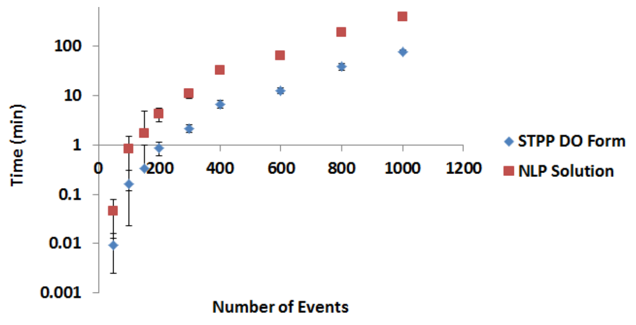

Fig. 10. Cumulative Compilation Time as a function of the number of events in the plan

Next, we compute a comparative measure of the temporal flexibility between both the STPP-DO form and the NLP solution and the original STPP. We compiled 50 random problems and compared the resulting interval durations to the original problems' interval durations. This ratio then represents the percentage of flexibility retained from the original problem; higher values of this ratio correspond to an increased robustness to disturbances during execution. We compare this to the flexibility ratio for the NLP-specified schedule on the same 50 problems; Fig. 11 presents the results. The DO form captures on average more than 70% of the temporal flexibility in the original plan, whereas the NLP solution captures less than 1%. The DO form provides a marked improvement in robustness to disturbance over the NLP solution while achieving global optimization of the schedule.

Number of Events	DO Form Flexibility Ratio	NLP Solution Flexibility Ratio
50	74.7% $\pm$ 3.3%	1.0% $\pm$ 0.3%
100	75.4% $\pm$ 3.2%	0.5% $\pm$ 0.1%
150	72.2% $\pm$ 3.1%	0.4% $\pm$ 0.1%
200	71.7% $\pm$ 2.0%	0.2% $\pm$ 0.05%

Fig. 11. Plan Flexibility of DO Form and NLP Solution

Finally, we present the computation times for single on-demand recompilation of the plan. These results simulate the execution latency associated with operator-specified changes to the workflow. Fig. 12 presents the compilation time results for randomly-generated structured problems ranging in size from 50 to 1000 events. The number of preference functions is set at 20% the number of events. We empirically analyzed the impact of the number of preference functions, ranging from 20% to 80% of the number of events, and found no significant effect on performance. Instead, the number of temporal constraints appears to be the primary driver of computation time.

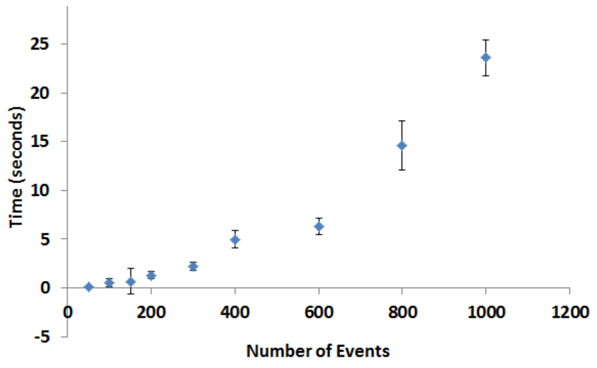

Fig. 12. Compilation Time as a function of the number of events in the plan

The results show satisfactory compilation times on the order of seconds for problems with hundreds of events. Compilation time is less than five seconds for problems with 400 events and less than 1 second for 150 events or less. These results provide sufficient capability for one-to-one human-robot collaboration, indicating a robot can adaptively schedule its actions over a horizon of approximately 75 activities with sub-second speed.

Multi-robot orchestration problems involve problems of one to two orders of magnitude larger, with thousands to tens of thousands of events. The current non-linear programming approach does not scale to these sizes, and we are exploring techniques to improve the efficiency of the compilation. The bottleneck in the algorithm's speed lies in the non-linear optimizer. We are currently investigating the reformulation of the non-linear program as an approximate linear program. This approach is feasible for non-linear programs where the non-linear terms are found only in the objective function, and the objective function is composed of separable, concave terms. Both of these conditions hold for the described approach, if the preferences functions strictly relate event execution times to the plan start.

VI. Robotic Demonstration

We have applied the Adaptive Preferences Algorithm to perform human-robot teaming using a small ABB IRB 120 industrial robot (set-up shown in 13). This demonstration is based on the spar building application described in Section II. The robot's job is to apply sealant to each hole, and the mechanic places and torques the fasteners. The mechanic and robot must work together to ensure that each fastener is placed within three seconds of sealant application. This requires that the robot adapt to the timing of the mechanic's actions to avoid applying the sealant too early. One set of workers, group A, likes to place all fasteners before torqueing them. The other set, group B, likes to place and torque each fastener before moving on to the next. The robot uses APA to adaptively schedule its actions based on the type of worker it is paired with; worker-type is inferred from the timing of the mechanic's actions. Specifically, APA tracks the two different sets of preference functions and switches to the set that achieves the maximum possible global preference value. The STPP representation of this joint human-robot plan is shown in Figure 14. Video of the demonstration can be found at http://tinyurl.com/7n439eg.

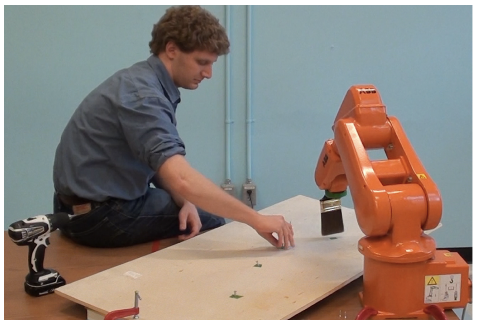

Fig. 13. Demonstration Set-up

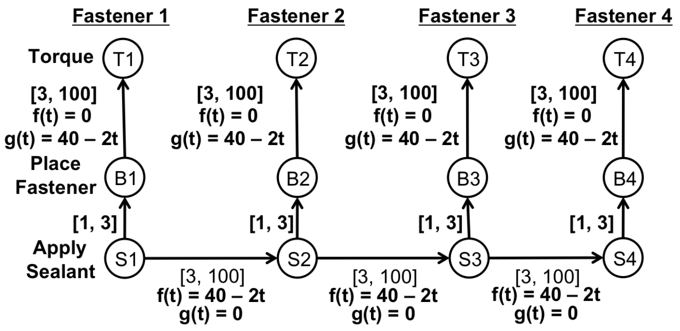

Fig. 14. STPP for the Robotic Demonstration; the f preference functions correspond to group A workers, while preferences, g, correspond to group B workers

Trials of human-robot teaming demonstrated that the robot was successfully able to adapt its schedule to both types of workers. When a group A mechanic performed the assembly task, the robot applied the sealant in regular intervals every 3 seconds to keep just ahead of the mechanic, allowing the mechanic to place the fasteners in the holes before the sealant dried. When a group B mechanic performed the task, the robot began by applying the sealant every 3 seconds. However, once it sensed that the mechanic had torqued the first fastener before inserting the second, the robot recompiled its schedule using group B preferences. The robot changed its pace to match the mechanic's using the newly computed flexible optimal scheduling policy. This required slowing down the rate of sealant application to every 7 seconds. Using the Adaptive Preferences Algorithm, the robot was able to make on-the-fly decisions about how to most effectively aid each worker.

VII. Conclusions and Future Work

In this paper, we introduce the Adaptive Preferences Algorithm (APA) that computes a flexible optimal policy for robot scheduling and control in assembly manufacturing. We show through empirical evaluation that the method is fast, adaptable and robust to uncertainty in execution, and supports interaction with human co-workers and supervisors whose preferences about task completion are prone to change. We also demonstrate the use of APA in human-robot teaming trials using a small industrial robot and show that the robot successfully adapts to the task preferences of different types of workers. As future work, we are investigating an approximate linear program formulation of the problem, with the aim of scaling-up APA to multi-robot orchestration problems involving thousands to tens of thousands of events.

References

[1] Gurobi optimizer version 5.0, April 2012.

[2] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1):61–91, 1991.

[3] A. Hofmann and B.Williams. Robust Execution of Temporally Flexible Plans for Bipedal Walking Devices. In *Proc. ICAPS*, pages 386–389, 2006.

[4] L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *Proc. IJCAI*, pages 322–327, 2001.

[5] P. Morris, R. Morris, L. Khatib, S. Ramakrishnan, and A. Bachmann. Strategies for global optimization of temporal preferences. In *Proc. CP*, pages 408–422. Springer, 2004.

[6] N. Muscettola, P. Morris, and I. Tsamardinos. Reformulating Temporal Plans For Efficient Execution. In *Proc. KRR*, 1998.

[7] F. Rossi, K. B. Venable, L. Khatib, P. Morris, and R. Morris. Two Solvers for Tractable Temporal Constraints With Preferences. In *Proc. AAAI workshop on preference in AI and CP*, 2002.

[8] J. Shah. *Fluid Coordination of Human-Robot Teams*. MIT PhD Thesis, Cambridge, Massachusetts, 2010.

[9] J. Shah and C. Breazeal. An Empirical Analysis of Team Coordination Behaviors and Action Planning With Application to Human-Robot Teaming. *Human Factors*, 52, 2010.

[10] J. Shah, J. Stedl, B. Williams, and P. Robertson. A Fast Incremental Algorithm for Maintaining Dispatchability of Partially Controllable Plans. In *Proc. ICAPS*, 2007.

[11] J. Shah, J. Wiken, B. Williams, and C. Breazeal. Improved Human-Robot Team Performance Using Chaski, a Human-inspired Plan Execution System. In *Proc. ACM/IEEE HRI*, pages 29–36, 2011.

[12] J. Stedl. Managing Temporal Uncertainty Under Limited Communication: A Formal Model of Tight and Loose Team Communication. Master's thesis, MIT, 2004.

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

[14] I. Tsamardinos and M. Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artif. Intell.*, 151:43–89, December 2003.