Towards Proactive Replanning for Multi-Robot Teams

Brennan Sellner and **Reid Simmons**
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Abstract

Rather than blindly following a predetermined schedule, human workers often dynamically change their course of action in order to assist a coworker who is having unexpected difficulties. We are examining how this notion of "helpful" behavior can inspire new approaches to online plan execution and repair in multi-robot systems. Specifically, we are investigating the enabling of proactive replanning by dynamically predicting task duration and adapting to predicted problems or opportunities through the modification of executing tasks by the planner. By continuously predicting a task's remaining duration, a proactive replanner is able to accomodate upcoming problems or opportunities before they manifest themselves. One way in which it may do so is by adjusting the allocation of agents to the various executing tasks through the addition or removal of agents, allowing the planner to balance a schedule in response to the realities of execution and execute schedules in less time than would otherwise be required.

We have developed a proof-of-concept system that implements duration prediction and modification of existing tasks, yielding simulated executed makespans[1] as much as 31.8% shorter than possible without these capabilities. The paper will discuss the potential benefits of proactive replanning, how it relates to previous work, and the outcome of our experiments to date.

Introduction

Current multi-robot systems can only aspire to the flexibility exhibited by teams of humans. Members of human teams are able to move smoothly between, and temporarily interrupt, tasks in order to render assistance when one of their teammates begins to struggle with his portion of the task. While much research has been performed with the aim of enabling robotic teamwork, the vast majority of implemented systems treat planning and execution as mainly independent, with plan repair or replanning occurring only once a task has failed or completed early, and with executing tasks immune from manipulation by the planning system. In contrast, humans are able to predict upcoming problems (or opportunities) and act to prevent (or take advantage of) them, often by changing their current task or approach to the task. We have created a planning/execution system that is able to anticipate problems and opportunities by *predicting task duration*, then proactively replanning, rather than waiting to replan until the event occurs. As part of this replanning, the system performs *live task modification*, that is, modifying currently executing tasks by adding or removing agents as appropriate to adjust the task's resource requirements, expected duration, and reliability. Our approach yields more efficient executions than existing approaches (such as continually replanning) by predicting problems and acting to avoid them, rather than simply reacting to them as they occur, and by providing the planner with limited control over task execution, which allows it to modify agent allocation to accommodate the realities of execution.

Our basic approach to planning is to utilize an existing symbolic planner/scheduler that makes use of iterative repair and optimization stages to build and optimize valid schedules. We have extended this planner's repair and optimization algorithms and heuristics to support an initial form of proactive replanning. We are not considering the distributed case, instead relying on a centralized planner/scheduler to formulate a scenario-wide schedule and dispatch it to the individual agents.

Our preliminary experiments have evaluated the concepts of duration prediction and live task modification while neglecting planning time. In our experimental domain, live task modification provides an average 30.3% reduction in makespan, while duration prediction yields a gain of 10.8%. When used in combination, makespan is reduced by 31.8% on average. The remainder of this paper will survey related work, discuss our approach to proactive replanning, and present our current experimental results.

Related Work

Prior work related to proactive replanning falls into two categories: planners and the architectures which enclose them. We will discuss how existing systems (fail to) address the concepts and requirements of proactive replanning and how they may be modified to realize its advantages.

Planners

The demands of proactive replanning on the planner are significant, including support for durative actions, temporal

[1]The makespan of a schedule is its overall length: the time between the start of the first task and the end of the final task.

constraints, exogenous events, multiple agents (or at least metric resources), and the ability to quickly replan or repair a plan in response to feedback from the executive.

Most relevant approaches describe the world as a set of state variables, each as a function of time. Each variable has an associated timeline, which encodes the variable's past states and expected future states, given the current plan. A task is an interval on one or more of these timelines, within which the value of the associated variables changes. This formulation is amenable to proactive replanning, as this representation of tasks lends itself to mid-task analysis. A number of planners use such formulations; we will discuss three: ASPEN (Chien *et al.* 2000b), EUROPA (Frank & Jónsson 2003), and IxTeT (Laborie & Ghallab 1995). All of these planners are capable of handling a range of resource types (e.g. reusable or consumable) through the use of underlying constraint networks and constraint satisfaction techniques.

Given a set of goals, a plan (initially empty), the current state, and the predicted variable timelines, ASPEN performs iterative plan repair to resolve conflicts and other flaws in the schedule. Repair and optimization steps may be interleaved continuously as the state of the system is updated, using a most-commitment strategy[2] (all variables are grounded as early as possible). This strategy makes the evaluation of metrics and projections of resource usage much simpler, but reduces the plan's flexibility. ASPEN appears to be a good fit to proactive replanning, although it lacks explicit multi-agent support. We have based our initial implementation on ASPEN.

EUROPA is a constraint-based interval planner which continually recasts the planning problem as a dynamic constraint satisfaction problem. It also makes use of timelines, and appears quite similar to ASPEN in functionality, although it attempts to always maintain a valid schedule.

IxTeT also plans by using timelines, each of which consists of a sequence of temporal assertions which can represent either the persistence of a value over an interval or an instantaneous change of value. IxTeT is based on a partial-order causal link planning process with constraint-satisfaction techniques and generates order-constrained plans with unbound variables. These plans are more flexible at execution time than the fully grounded plans produced by ASPEN and EUROPA, but make it significantly more difficult to evaluate metrics and predict conflicts. While IxTeT can perform plan repair through search in the partial plan space, the importance to proactive replanning of metric evaluation and conflict prediction make IxTeT less suited to proactive replanning than ASPEN.

Architectures

Architectures serve to tie the different elements of an autonomous system into a cohesive whole, and define how these components interact. Such architectures commonly have been divided into two or three primary components: a functional layer which interacts with hardware and executes low-level commands and a decisional layer which determines what commands should be executed. The decisional layer often is split into a high-level deliberative planner and a mid-level execution layer responsible for overseeing the functional layer. Proactive replanning requires a close, high-frequency connection between the planning and executive layers, so that the planner is kept up to date on execution progress, may continuously replan, and may affect currently executing tasks. Existing architectures "lock" currently executing tasks to prevent the planner from modifying them; this lack is the primary architecture-related impediment to the implementation of proactive replanning.

Three-layer architectures such as 3T (Bonasso *et al.* 1997) and ATLANTIS (Gat 1992) may be adaptable to proactive replanning, but the ties between their planning and executive layers generally are not tight enough to support duration prediction and live task modification.

LAAS (Alami *et al.* 1998)[3] and CLARAty (Nesnas *et al.* 2003) both encapsulate planning/scheduling and execution into a single layer. In the case of LAAS, the two are tightly intertwined, while in CLARAty the decisional layer still contains distinct planning and execution objects. At the moment, two instances of CLARAty's decisional layer are available: CASPER (Chien *et al.* 2000a) and CLEaR (Estlin *et al.* 2001). CASPER uses the ASPEN (Chien *et al.* 2000b) planner and a simple executive, while CLEaR adds a TDL-based (Simmons & Apfelbaum 1998) executive to CASPER. Both LAAS and CLARAty appear well-suited to proactive replanning, due to the relatively tight coordination possible between their respective planning and execution components. However, as currently implemented, neither is able to modify a currently executing task, since active tasks are locked to avoid conflicts between planner and executive. In fact, (Chien *et al.* 2000a) states that *How to ensure the planner does not change activities that are already in execution?* is an explicit problem with interleaved plan execution and repair. In order to realize proactive replanning, and specifically live task modification, this restriction on the modification of executing tasks must be carefully relaxed to allow the planner to dynamically change the allocation of agents. Due to concerns about how well IxTeT (LAAS's planner) could be adapted to proactive replanning, we selected CASPER as a foundation for our work.

The final relevant architecture is IDEA (Muscettola *et al.* 2002), which advocates the use of planning as the core of each level of abstraction, from mission planning to reactive execution. "Planner" and "executive" modules are both implemented as planners operating with different planning horizons. As currently implemented, IDEA segregates the planning horizon between the "planner" and "executive" modules. This segregation would need to be relaxed in order to support proactive replanning so that the higher-level module can modify executing tasks in response to predicted problems. However, IDEA's inter-module communications appear sufficient for lower-level modules to keep higher-level planners abreast of developments. IDEA makes use of the EUROPA planner as its internal planning module.

[2]The mix of repair and optimization steps is determined by the enclosing architecture.

[3]IxTeT is the planning component of LAAS.

Approach

Proactive replanning consists of predicting problems or opportunities, then adjusting current and future tasks as appropriate. In order to enable it, we propose two novel extensions to the conventional approach to planning and execution: task duration prediction and live task modification. We extend an existing symbolic repair-based planner/scheduler to incorporate these innovations. *Duration prediction* allows the planner to repeatedly predict how long each task will take to complete as execution progresses. In the presence of setup tasks and non-instantaneous state changes, this allows the planner to take advantage of opportunities that would pass unnoticed to conventional approaches. For instance, if a task A is predicted to finish early, the setup actions of any tasks following A may be started earlier than initially scheduled, enabling a shorter makespan. *Live task modification* tightens the connection between the executive and the planner by allowing the planner to institute changes in teams of agents *during* task execution. This provides the planner with the ability to utilize idle agents and balance resources between tasks in order to compensate for under-performance, take advantage of unexpectedly good performance, or add additional agents to deal with uncommon contingencies. Together, duration prediction and live task modification allow a planner to be proactive in its approach to plan repair and optimization, addressing likely problems rather than simply reacting to them as they occur.

Domain and Optional Roles

Task modification by changing the team performing a task is predicated upon the existence of tasks in the domain which afford the addition or removal of agents. In the absence of such tasks, the benefits of proactive replanning are significantly reduced. We characterize a task as consisting of a set of roles, some required and some optional. Required roles must be filled throughout the task's execution, while optional roles are not necessary for the successful completion of the task. When filled, optional roles may provide a variety of benefits, such as increasing robustness, allowing faster recovery from failures, and increasing the rate of progress during normal operation.

Our current scenario is a subset of a construction task similar to one that might be performed by a robotic team constructing habitats on Mars or the Moon. The overall scenario consists of adding panels to an existing framework (Figure 1), and requires four tasks per panel to bring it from storage and attach it to the framework: *Transport_Panel*, *Rotate_Panel*, *Place_Panel*, and *Bolt_Panel*. Each task has different agent requirements and different combinations of optional roles (Table 1). For instance, in the *Transport_Panel* task, two agents (the *transporters*) must be assigned throughout the task in order to carry the panel, but optionally up to two additional transporter agents can assist, increasing the team's rate of progress. If agents are available for the optional *scout* roles, they range ahead of the transporters, seeking out the fastest path to the goal that avoids terrain in which the transporters may become mired. If the transport sub-team does become mired in rough terrain, optional *tow* agents are able to assist in their extraction. Note that if the transporters become mired, scout agents may be reassigned to the tow roles.

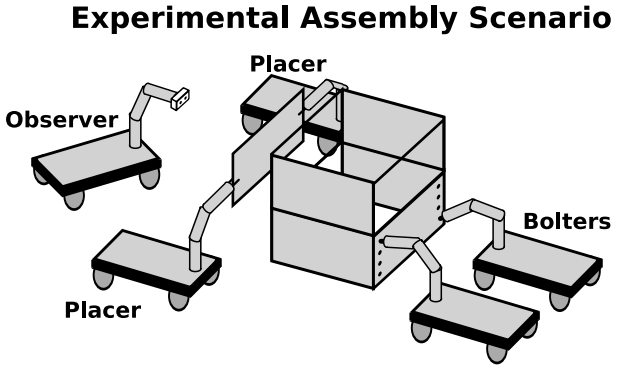

Figure 1: The notional structure being assembled in our experiments. Tasks are accomplished by fluid teams of agents filling required and optional roles.

While many of the benefits of proactive replanning are only realized in domains incorporating optional roles, such roles may be designed naturally into a wide range of tasks and domains, granting the planner significant additional flexibility through the use of proactive replanning.

Architecture

Our approach is based on an architecture related to the classic three-tiered architectures (e.g. (Bonasso *et al.* 1997)), and incorporates planning, executive, and behavioral layers (Figures 2 and 3, (Sellner *et al.* 2006)). The centralized planner forms an initial valid schedule, and is responsible for repairing and optimizing the schedule as execution proceeds. During execution, the planner dispatches tasks to the distributed executives as their start times arrive, along with their initial parameters (e.g. the composition of the initial team). The executive then activates the relevant portions of the behavioral layer. As the behavioral layer executes the task (either in simulation or the real world), it continuously provides updated state and task completion information to the executive. If state variables of interest to the planner are modified or a task has completed, an update from the executive to the planner is triggered. This update may result in a change in the predicted or final task durations, potentially creating schedule conflicts or opportunities, such as multiple tasks requiring the same agent at the same time or idle agents. The planner then acts to repair and optimize the schedule by using a variety of heuristics, including those that make use of live task modification to change the composition of teams currently performing tasks.

Planner We are making use of the ASPEN planner (Chien *et al.* 2000b) as our planning layer. ASPEN is a repair-based planner that employs user-defined heuristics to make decisions at a series of choice points throughout the repair/planning process. It also maintains task and parameter constraint networks, allowing tasks to be temporally constrained with respect to each other, and task parameters to

Task	Role	Role Type	Capacity [a]	Effect
Transport_Panel	Transporter	Required	2	Carries panel from stockpile to worksite. May become temporarily bogged down in the terrain.
	Transporter	Optional	2	Helps to carry panel; increases rate of progress.
	Scout	Optional	2	Decreases probability of becoming mired.
	TowTruck	Optional	2	Reduces time required to extract the team after becoming mired.
Rotate_Panel	Rotator	Required	2	Rotates panel from horizontal carrying position to the vertical position required for placement. Panel may slide out of Rotators' grips; recovery from this error requires a Lifter.
	Lifter	Optional	1	If the panel slides, the Lifter will reposition it to allow rotation to continue.
Prep_Hangers	Hanger	Required	1	Places hangers on the structure in preparation for panel placement. No failures; must occur prior to each *Place_Panel*.
Place_Panel	Placer	Required	2	Places panel on structure; may fail, necessitating resetting to the start position before another attempt may begin.
	Observer	Optional	2	Decreases probability of failure.
Bolt_Panel	Bolter	Required	1	Inserts bolts.
	Bolter	Optional	2	Increases bolting rate through parallelization, decreasing task duration.

[a] The capacity of an optional role indicates the maximum number of agents that may be assigned to it (e.g. 0, 1, or 2 *Observers* may be assigned to a *Place_Panel* task). The capacity of a required role indicates exactly how many agents must be assigned to the role (e.g. there must be exactly two *Placer*s assigned to a *Place_Panel* task).

Table 1: Scenario tasks and their component roles.

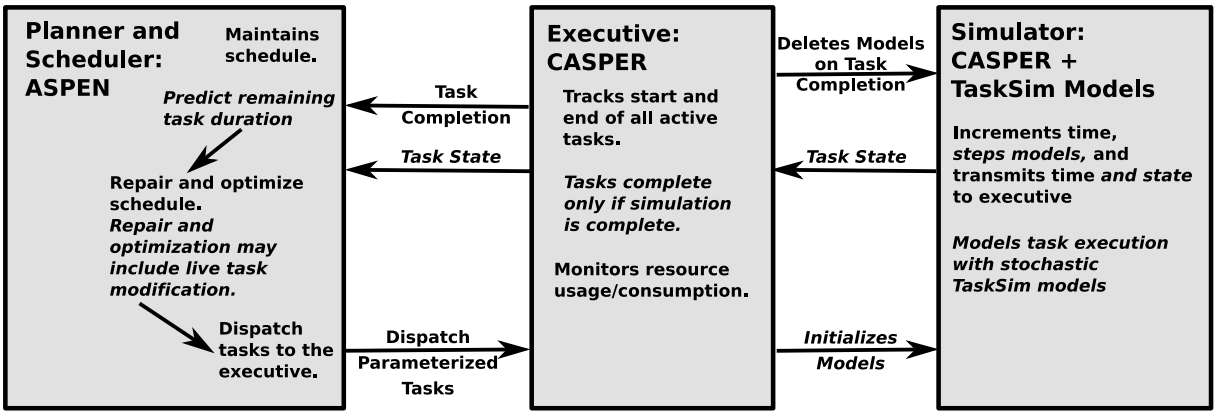

Figure 3: An overview of the planning, execution, and simulation system. Components in italics were either created or heavily modified from ASPEN/CASPER for this work. TaskSim is our task modeling library.

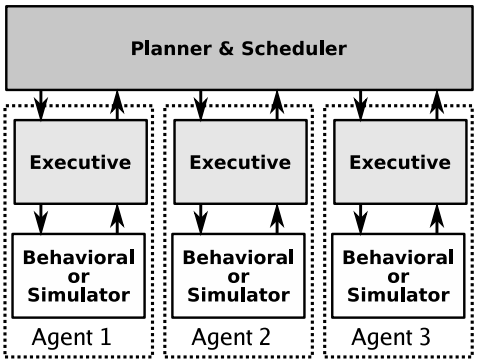

Figure 2: Our approach makes use of a centralized planner that dispatches tasks to distributed executive and behavioral layers.

be constrained with respect to each other, as well as the values of resources and parameters in other tasks. We make extensive use of the parameter constraint network when performing duration prediction. We selected ASPEN because of its repair-based paradigm, support for durative actions, and most-commitment approach to variables, all of which are either required or ease the implementation of proactive replanning.

ASPEN's approaches to repair and optimization are quite similar: (1) select a conflict to resolve (or metric to optimize), (2) stochastically select and apply a repair (or optimization) method, and (3) repeat until all conflicts are resolved or the allotted time has elapsed. The repair and optimization methods are heuristic routines applicable to one or more types of conflicts (or metrics). We have extended ASPEN in a variety of ways, including providing a number of domain-specific heuristics and adding new repair and optimization methods that utilize the structure inherent in the optional role tasks of our domain.

Executive The executive is largely composed of CASPER's (Chien *et al.* 2000a) (Estlin *et al.* 2005) single-layer executive, with a number of modifications to suit our needs. CASPER was selected due to its integration with ASPEN, our planner of choice. The original CASPER executive tracks resource consumption and task start and end points, as transmitted to it from the planner, and informs the planner when tasks complete. We have extended the CASPER executive to manage the simulator by starting parameterized task simulations as tasks are dispatched to the executive by the planner, then continuously relaying task state information back to the planner (Figure 3, center). The original CASPER simulator has no execution model, and tasks always complete exactly when scheduled to do so[4]. A wide variety of minor other extensions and modifications were made to support our simulator and the properties of our domain.

[4]This is true of the "stock" CASPER as provided by JPL. Within JPL, each project implements its own simulator that uses the CASPER framework to interact with the executive and ASPEN. We are simply doing the same.

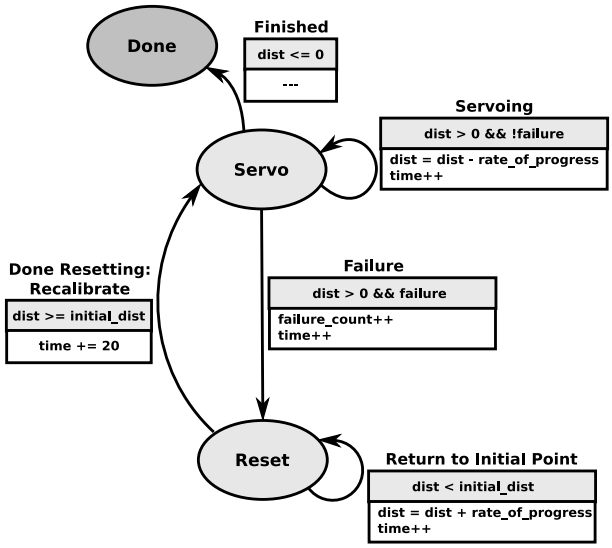

Figure 4: The stochastic simulation model for *Place_Panel*. This is used to simulate execution, and provides the state that is used for duration prediction during execution. In addition, this model is used offline to build the duration prediction tables.

Simulator The simulator maintains an independent model of each task currently being executed. This stochastic model introduces into the system a degree of uncertainty akin to that found in real-world robotic teams. In our current system, these models are precisely the models used to build our duration predictions (see Figure 4). They model only the high-level progress of a task, including nonterminal failures, and report task-level state to the executive, while allowing the team composition to be changed dynamically. For instance, while individual components (such as manipulators or sensors) of agents are not simulated, events such as a transport team becoming mired are.

Duration Prediction

The problem of duration prediction is one of approximation. Duration prediction algorithms must be capable of estimating a task's remaining duration based on the current team and task state. Since the team / state space is exponential in the number of optional roles and state variables, and factorial in the number of agents, the space can become quite large. For instance, the *Transport_Panel* task briefly discussed above may be performed by 49,876 distinct teams if 10 heterogeneous agents are available. A minimal task state for *Transport_Panel* is the remaining distance to the goal. This is a continuous value; assume the maximum distance is 20 meters, and that we discretize it into 10 centimeter portions. This yields 201 possible state space values, resulting in $201 * 49,876 = 10,025,076$ distinct inputs from which we must be able to predict the task's remaining duration. As a result, the fundamental problem associated with dura-

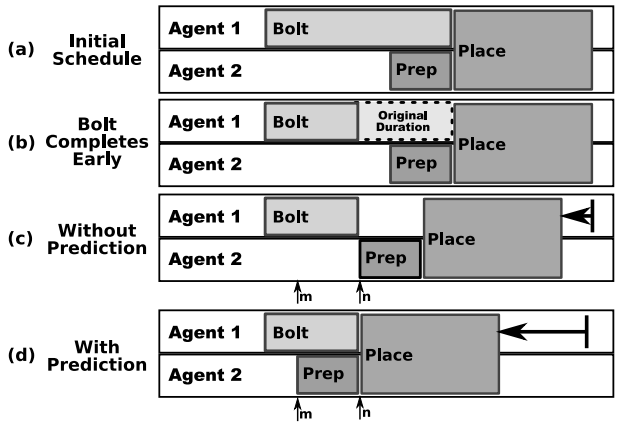

Figure 5: Continually predicting the remaining duration of executing tasks allows the planner to make the fullest use of opportunities presented by task under-runs.

tion prediction is two-fold: provide estimates of sufficient fidelity, while keeping computational and spatial requirements within limits. Our initial approach is straightforward: prior to execution, build a table of estimates of the mean expected duration that spans the team / state space, then refer to it at execution-time. As the task state evolves during execution, we update the duration of tasks in our schedule from our table of estimates.

Among other advantages, duration prediction allows the planner to take advantage of opportunities provided by task under-runs. Consider a fragment of a construction scenario consisting of the bolting of a panel ($Bolt$) and the preparation ($Prep$) and placement ($Place$) of the next panel (Figure 5). $Bolt$ requires only one agent, but must be completed before the next $Place$ begins. $Prep$ is a setup task for $Place$, and is constrained to meet $Place$ (that is, $Prep$'s end time must equal $Place$'s start time). $Place$ requires two agents, while $Prep$ only needs one. The initial schedule is depicted in Figure 5a.

We can see that if $Bolt$ completes early (Figure 5b), $Prep$ and $Place$ may in turn be started early, reducing the overall makespan. If the planner does not predict this early completion, the only optimization available is to start $Prep$ immediately upon $Bolt$'s completion (Figure 5c). However, this is inefficient, as $Prep$ may be executed in parallel with $Bolt$. If the planner were able to predict $Bolt$'s true completion time at any point prior to its completion at Point n, it would be able to start $Prep$ early, realizing a reduction in makespan. Ideally, the prediction would be made prior to Point m (that is, $Length(Prep)$ seconds before $Bolt$'s early completion), allowing $Place$ to be scheduled immediately after $Bolt$, and $Prep$ to be executed entirely in parallel with $Bolt$ (Figure 5d).

The addition of duration prediction provides the planning/execution system with additional power, allowing it to take advantage of opportunities that would go unnoticed if the system were only to react to over- or under-runs as they occur. To date, planning/execution systems replan only when over-runs or under-runs manifest themselves – they are not taking advantage of the optimizations made possible by duration prediction.

Live Task Modification

For a robotic team to exhibit the same fluidity that human teams achieve, the planning/execution system not only must predict the future state of tasks but also must be able to modify currently executing ("live") tasks. Existing planning/execution systems provide no mechanisms, or at best only crude ones[5], for the planner to manipulate the teams assigned to live tasks. This reduces the intricacy of both the planner and the executive by not requiring either to reason about or perform the addition (or removal) of agents to (or from) a task; however, it results in avoidable inefficiencies.

Live task modification expands the options available to the planner by allowing it to modify the team assigned to a task during its execution by adding or removing team members. The root problem of live task modification is complexity: a vast search space of potential teams and a multiplicity of tasks that must be managed to make the addition and removal of agents possible. (Recall, *Transport_Panel* may be performed by 49,876 distinct teams if 10 agents are available.)

To illustrate the usefulness of live task modification, consider the following example. The initial schedule consists of two $Bolt$ tasks: A and B (Figure 6). Each task must be performed by one agent, but additional optional bolting agents may be added to reduce the task's duration. Initially, let Agents 1 and 2 perform $BoltA$, while 3 and 4 work on $BoltB$ (Figure 6a). Assume that $BoltB$ finishes execution early while $BoltA$ over-runs (Figure 6b). In the absence of live task modification, the schedule's length would be the length of the over-running $BoltA$ (Figure 6c). However, if live task modification is available, Agents 3 and 4 may be transferred to $BoltA$ once $BoltB$ completes (Figure 6d), reducing its duration. If both duration prediction and live task modification are available, Agent 3 may be transferred once the over- and under-runs are predicted (Figure 6e), partially balancing the schedule and further reducing its length.

The current implementation of live task modification is used by the planner's repair methods to resolve agent oversubscription conflicts by either adding additional agents to shrink a task's duration or removing optional agents to eliminate the overlap between two tasks' agent requirements. In addition, the planner's optimization methods add idle agents to currently executing tasks on the critical path, in order to make the most use of available resources, as illustrated in Figure 6d. The transferring of agents between two tasks, as depicted in Figure 6e, has not yet been implemented. Also, we currently assume that agents may be added to a team instantaneously, an assumption we will be relaxing in our future work.

[5]For instance, IxTeT (Lemai & Ingrand) allows the planner to abort executing tasks, but this is the extent of the control the planner may exert over the executive.

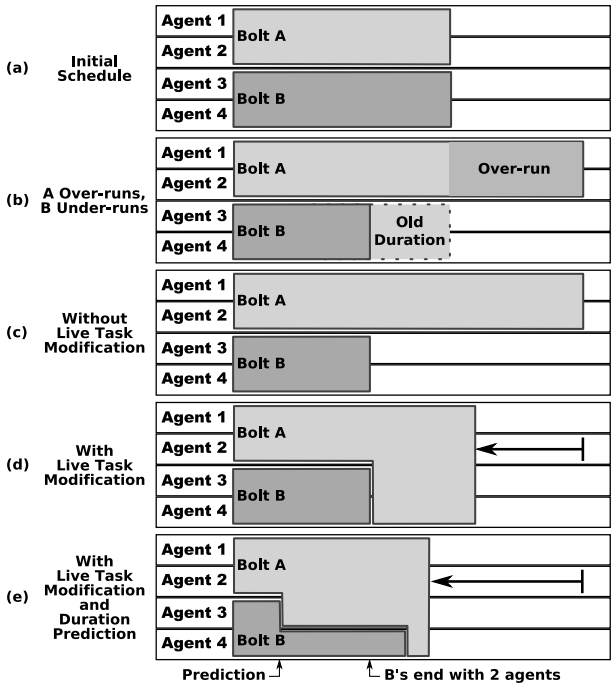

Figure 6: Reassigning agents during task execution allows the system to adjust resource allocation to reflect the realities of execution.

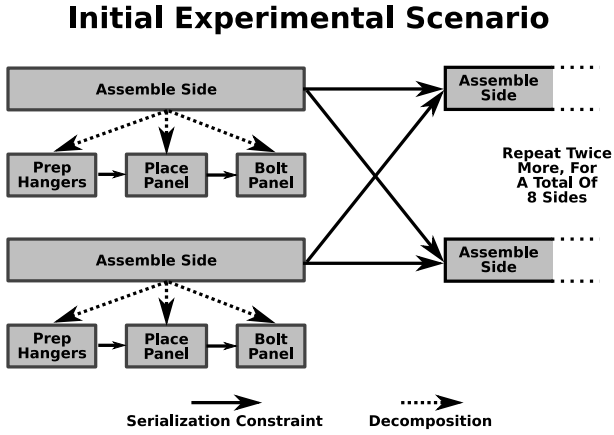

Figure 7: The scenario used in our experiments. A total of eight *Assemble_Side* tasks are included, serialized by pairs, since teams cannot work on adjacent sides of the structure without risking interference.

Results

Building on ASPEN and CASPER, we have implemented and evaluated a proof-of-concept planning and execution system to examine how duration prediction and live task modification affect the makespan of an executed schedule. When planning and optimization time is not included, both approaches result in shorter simulated makespans, although the effects of live task modification are more dramatic. When combined, they reduce makespans further than either could alone, yielding executed schedules on average 31.8% shorter than was achieved with a baseline implementation. We discuss here the current scenario, experimental design, and results.

Scenario

Our initial scenario is a subset of the domain described above and illustrated in Figure 1, and includes only the *Place_Panel* and *Bolt_Panel* tasks, which are responsible for temporarily hanging a panel from the structure and permanently fastening it, respectively. A *Bolt_Panel* task must follow each *Place_Panel*, although not necessarily immediately. Each *Place_Panel* task must be preceded by an associated setup task (*Prep_Hangers*), which places the temporary hangers used to hold the panel in place until it is bolted. These temporal constraints are enforced through the use of an *Assemble_Side* task, which decomposes into the *Prep_Hangers*, *Place_Panel*, and *Bolt_Panel* tasks (Figure 7). There are eight *Assemble_Side* tasks in the scenario, representing the construction of a two panel-high, four-sided structure (Figure 1). To avoid interference between teams, only opposite sides of the structure may be assembled simultaneously, leading to the high-level ordering depicted in Figure 7. Planning takes place over an effectively infinite planning horizon. Four homogeneous agents are available, each of which is equally capable and may perform each available role, although a given agent can hold only a single role at a time.

The *Place_Panel* task represents a team of robots attempting to place a panel onto temporary hangers on an existing structure. These hangers first must be placed by the setup task, *Prep_Hangers*. *Prep_Hangers* requires only a single agent, takes a fixed length of time, and involves no uncertainty. *Place_Panel* requires two placement agents, which perform the panel manipulation. The placers may fail and be forced to restart at any point in the task. If one or two optional observer agents are present, the likelihood of failure is reduced. With two required and two optional roles, and four available agents, there are a total of 24 possible teams.

The *Bolt_Panel* task is responsible for the permanent attachment of a previously placed panel. It has one required and two optional bolter roles. If filled, the optional roles increase the team's rate of progress, although each additional agent results in a smaller increase. However, adding additional agents does add some risk to the team: there is an independent chance that any given agent will fail on a given timestep. If any agent fails, the entire team is forced to recover, delaying the task. With four available agents, one required, and two optional roles, there are 14 possible teams (see Figure 8).

This initial scenario provides a reasonable testbed for our proactive replanning methods, exhibiting stochastic characteristics, temporal constraints, setup tasks, and temporary failures, all of which provide opportunities for both duration prediction and live task modification.

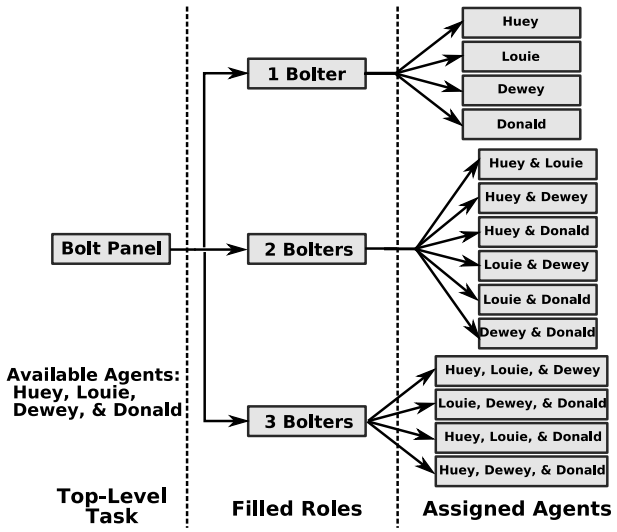

Figure 8: The *Bolt_Panel* task fastens an already-placed panel to the structure. With four agents, there are the above 14 ways to perform the task.

Experimental Conditions

In order to experimentally validate our beliefs about the effectiveness of duration prediction and live task modification, we conducted a 2x2 experiment in simulation comparing the four combinations of the two conditions, using the modified ASPEN, executive, simulator, and scenario discussed above. In our initial experiments, we neglect planning time in order to evaluate the validity of our approach. That is, the planner is given an arbitrary amount of time between timesteps to repair and optimize the schedule. Our experimental conditions are:

Baseline The baseline case utilizes neither duration prediction nor live task modification during execution. All tasks initially are set to a fixed duration equivalent to the initial prediction made for the assigned team by the duration prediction algorithm, but are not updated as the task's state evolves throughout execution. If a task over-runs, its duration is extended in increments of one time step until it completes. In addition, if a task finishes early, its duration on the planner's schedule is shrunk instantly as soon as it has completed. The planner utilizes repair and optimization heuristics that modify teams, but only with non-executing tasks. Prior to the beginning of execution, a valid schedule is created by placing all tasks on the schedule without concern for conflicts, then iterating the repair algorithm until a legal schedule is found. During execution, the following actions are performed before each time step:

1. **Right-shift:** Move any non-executing tasks with agent subscription ("resource") conflicts into the future until the conflict is resolved.
2. **Left-shift:** Move all non-executing tasks to the earliest time consistent with temporal and resource constraints.
3. **Optimize:** Run 20 iterations of the planner's iterative optimization algorithm, without considering executing tasks.
4. **Repair:** Perform iterative repair until all conflicts are resolved. Executing tasks may *not* be modified.

Prediction The prediction case adds task duration prediction to the baseline scenario. This gives the planner a more accurate estimate of when tasks will complete by updating its predictions at every timestep, allowing the planner to adapt to delays before they are manifested. The only source of advantage in this case is the timely scheduling of setup tasks. Executing tasks may *not* be modified in this condition.

Live Modification In live task modification, the planner is allowed to modify executing tasks, changing the resources used and, indirectly, the speed with which the task will be completed. However, duration prediction is not available in this condition, so the planner is limited to resolving agent conflicts and putting idle agents to use with this approach.

Prediction and Live Modification In this case, both duration prediction and live modification are enabled. The planner is able to foresee opportunities and conflicts while taking advantage of them through the modification of executing teams. For instance, the planner is able to assign idle agents to active tasks on the critical path that are taking longer than expected.

Data

We performed 50 simulated runs under each of the four conditions outlined above. Statistics are summarized in Table 2. All data is presented as the mean across the runs, with its standard deviation in parentheses. While makespan is self explanatory, several of the other terms may require some clarification. "Additional tasks scheduled" is the number of tasks on the final schedule beyond the minimum. When live task modification is available, an additional pair of tasks will be scheduled every time an executing task is modified. "Average agent usage" is the average across all agents of the fraction of the makespan during which each agent was active. This can be considered a measure of how much "space" was left in the final schedule. "Repair episodes" and "Optimization episodes" refer to the number of times the repair and optimization algorithms were run, respectively. The repair algorithm is invoked only when conflicts arise in the plan, while optimization is attempted at every time step. "Repair iterations" is the average number of individual repairs that were applied during the course of a run. "Time spent repairing" and "Time spent optimizing" are the total wall clock times spent respectively repairing or optimizing plans during a run. Finally, "Successful modifications" is a count of the number of optimization attempts that succeeded in reducing the makespan by using idle agents to begin task execution early or reinforcing teams already executing tasks.

	Baseline	Prediction	Live Task Modification	Prediction and Live Task Modification
Makespan (s):	1176.90 (343.58)	1050.14 (273.32)	820.84 (123.55)	802.76 (141.53)
Makespan (% of Baseline)	— (—)	89.2% (79.6%)	69.7% (36.0%)	68.2% (41.2%)
Reduction in makespan	— (—)	10.8% (20.4%)	30.3% (64.0%)	31.8% (58.8%)
Additional tasks scheduled:	— (—)	— (—)	32.68 (4.84)	31.16 (4.30)
Average agent usage (fraction of makespan):	0.59 (0.05)	0.56 (0.06)	0.91 (0.02)	0.86 (0.07)
Repair episodes:	30.04 (36.60)	123.28 (23.06)	15.78 (8.00)	97.38 (23.86))
Repair iterations:	4079.88 (9507.00)	3313.86 (6815.19)	3384.76 (6834.70)	7883.14 (15169.14))
Time spent repairing (s):	15.13 (24.23)	14.23 (16.99)	13.57 (15.58)	26.14 (40.82)
Optimization episodes:	1177.90 (343.58)	1129.24 (357.58)	821.84 (123.55)	807.14 (142.73)
Successful modifications:	35.26 (6.15)	35.14 (5.46)	51.64 (6.19)	48.58 (7.58)
Time spent optimizing (s):	177.43 (131.16)	181.00 (141.60)	66.32 (73.95)	67.99 (77.00)

Table 2: Results of our initial experiments. 50 execution runs were performed under each condition. All data is reported as *mean* (*standard deviation*). All runs were performed on a lightly-loaded Pentium-4 3 GHz with 1GB of RAM.

Discussion

The most notable aspect of the data in Table 2 is the average makespan, which also is depicted in Figure 9. We can see that both duration prediction and live task modification provide benefits, although the latter clearly is more effective. This is unsurprising, as task modification is useful in a variety of situations, while prediction alone can provide benefits only from task under-runs in the presence of setup tasks or significant, sudden over-runs. In the experimental scenario, setup tasks only occur in concert with *Place_Panel* tasks. The combination of prediction and live task modification performs only slightly better than live task modification alone. The lack of synergy is due in part to the implemented optimization heuristic: it assigns idle agents, but does not remove agents from active teams not on the critical path. Extending the heuristic to support such transfers should increase the effectiveness of the combined case. In addition, domains with more setup tasks, such as those which require setup tasks when adding an agent to a live team, should result in larger relative improvements in the combined case, due to duration prediction.

It is also notable that the amount of repair increases by a factor of 4 – 6 when duration prediction is enabled. This is largely due to jitter: the predicted completion time of a task will fluctuate slightly even during normal operation, since the underlying model is learned and imperfect. This often results in semi-spurious conflicts in which tasks overlap (or a temporal constraint is violated) by a few timesteps. The repair of these minor conflicts accounts for the observed increase in repair iterations. In contrast, when duration prediction is not present, a conflict will not occur until a task actually over-runs its scheduled finish time. While this results in fewer planning episodes, the cost is longer makespans. This is a classic computation/quality trade-off: if we are able to either reduce the jitter or optimize the system to perform the requisite repair operations in real time, we will be able to take full advantage of duration prediction.

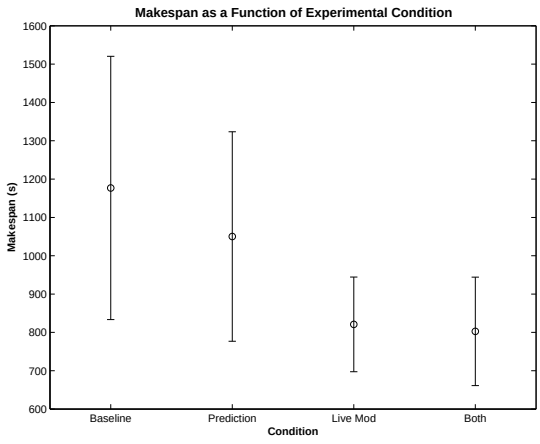

Figure 9: The average makespan and standard deviation for each of the four test conditions.

In a first attempt at alleviating this jitter, we have added a buffer of 5 seconds at the end of every task, which absorbs some of the jitter and, to an extent, reduces the number of planning episodes. Significant work on the minimization of jitter has been performed in fields such as control systems theory. A more formal solution is a component of our future work.

In the current implementation, the iterative optimization procedure is run at every timestep, which is clearly suboptimal and results in a significant waste of computation time. However, task duration estimates are updated at each timestep, making it possible that an optimization opportunity has presented itself. This indiscriminate optimization is acceptable in a proof-of-concept system such as ours, but we will be investigating a more focused approach.

Finally, the number of successful team modifications dou-

bles when live task modification is enabled. If there are free agents that cannot begin any unstarted tasks on the critical path, they will be assigned to executing tasks on the critical path. This is possible only when live task modification is present; if it is not, the optimization attempt would fail.

Future Work

Proactive replanning is an area rich in potential research; we will touch on a few possibilities here. We plan to extend our work to support non-instantaneous agent transfers. That is, the addition (or removal) of an agent to (or from) a team will take time, and potentially will slow the remainder of the team during the process. In addition, the agent being transferred will need to execute one or more setup actions (such as moving to a new location, warming up an instrument, etc.) before it can join its new team. This will complicate the planner's reasoning process and reduce the utility of live task modification somewhat, but is necessary to more accurately reflect real world tasks.

As soon as the modification of teams becomes non-instantaneous, we must address risk management. When transferring an agent, there is a possibility that its new team will complete its task prior to the arrival of the new agent, resulting in wasted time and resources. In addition, there is risk associated with starting setup tasks early if they must meet the next task. Reasoning about the uncertainty associated with task duration and how this should affect the planner's decision to modify a team is an important avenue of research.

We will also be expanding the fidelity of the simulator by moving to a much lower-level approach, in which the execution of tasks is stochastic, but is not necessarily captured accurately by the models used to build our duration prediction tables. In a similar vein, we plan to implement proactive replanning as part of our existing real world assembly team (Sellner *et al.* 2006). Our approach to duration prediction will need to be extended to adapt to mismatches between observed data and our underlying task models.

Finally, we will investigate different approaches to representing duration prediction estimates and searching the space of possible teams with the goal of bringing the benefits of proactive replanning to a real-time system.

Conclusion

Inspired by the fluidity of human teams, we have developed a proof-of-concept proactive replanning system capable of predicting task durations and modifying currently executing tasks. These two capabilities allow the system to execute more efficient final plans than are otherwise possible. We have established the efficacy of our approach in step-wise simulation, which has demonstrated a reduction in executed makespan of 10.8% - 31.8%, depending on the combination of duration prediction and live task modification that is utilized. Further research is indicated, but proactive replanning appears to be a fruitful addition to the stable of approaches to planning and execution.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming* 17(4).

Bonasso, R.; Firby, R.; Gat, E.; Kortenkamp, D.; Miller, D.; and Slack, M. 1997. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence* 9(2-3):237–256.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000a. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*.

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. 2000b. Aspen – automated planning and scheduling for space mission operations. In *Space Ops*.

Estlin, T.; Volpe, R.; Nesnas, I.; Mutz, D.; Fisher, F.; Engelhardt, B.; and Chien, S. 2001. Decision-making in a robotic architecture for autonomy. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*.

Estlin, T.; Gaines, D.; Chouinard, C.; Fisher, F.; Castano, R.; Judd, M.; Anderson, R. C.; and Nesnas, I. 2005. Enabling autonomous rover science through dynamic planning and scheduling. In *Procedings of the IEEE Aerospace Conference*.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Journal of Constraints, Special Issue on Constraints and Planning* 8(4).

Gat, E. 1992. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Lemai, S., and Ingrand, F. ??? Interleaving temporal planning and execution: Ixtet-exec. In *???*

Muscettola, N.; Dorais, G.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reactive agents. In *Proceedings of the 3rd International NASA Workshop Planning and Scheduling for Space*.

Nesnas, I.; Wright, A.; Bajracharya, M.; Simmons, R.; Estlin, T.; and Kim, W. S. 2003. Claraty: An architecture for reusable robotic software. In *Proceedings of the SPIE Aerosense Conference*.

Sellner, B.; Heger, F. W.; Hiatt, L. M.; Simmons, R.; and Singh, S. 2006. Coordinated multi-agent teams and sliding autonomy for large-scale assembly. *Special Issue of the Proceedings of the IEEE on Multi-Robot Systems*. In Press.

Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*.