# 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 change tasks in order to assist a coworker experiencing 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 proactive replanning, which attempts to predict problems or opportunities and adapt to them by shifting agents between executing tasks. By continuously predicting a task's remaining duration, a proactive replanner is able to accommodate upcoming problems or opportunities before they manifest themselves. One way to do so is by adding or removing agents to or from the various executing tasks, allowing the planner to balance a schedule in response to the realities of execution.

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 32% shorter than possible without these capabilities.

## 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 a 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, 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 problem or opportunity arises. 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. In our experimental domain of multi-robot assembly, 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 surveys related work, discusses our approach to proactive replanning, and presents our current experimental results.

## Related Work

Prior work related to proactive replanning falls into two categories: planners and the architectures that include them. We 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 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



[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.

states and expected future states, given the current plan. A task is an interval on one or more of these timelines, within which the values of the associated variables change. Several planners use such formulations; we 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 (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 that recasts the planning problem as a dynamic constraint satisfaction problem. It also makes use of timelines, although it attempts to always maintain a valid schedule, while ASPEN tolerates conflicts. EUROPA uses a least-commitment strategy to maintain flexibility in the plan, delaying the grounding of variables, such as task duration, as long as possible. Proactive replanning could be implemented using EUROPA, but the reasoning would likely be more complex due to this least-commitment approach.

IxTeT also plans by using timelines, each of which consists of a sequence of temporal assertions that can represent either the persistence of a value over an interval or an instantaneous change of value. IxTeT is based on partial-order causal-link planning with constraint-satisfaction techniques and generates partially ordered plans with unbound variables. These plans are more flexible at execution time than the fully grounded plans produced by ASPEN, 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 the components interact. Such architectures commonly have been divided into two or three primary components: a functional layer that interacts with hardware and executes low-level commands and a decisional layer that 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) and CLARAty (Nesnas *et al.* 2003) both encapsulate planning 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 order to realize proactive replanning, this restriction on the modification of executing tasks must be 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.

Another 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. 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.

## Approach

Proactive replanning consists of predicting problems (or opportunities), then adjusting current and future tasks as appropriate. To enable proactive replanning, we extend CASPER with two novel extensions to the conventional approach to planning and execution: *task duration prediction* and *live task modification*. *Duration prediction* allows the planner to 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 by conventional approaches. For instance, if a task $A$ is predicted to finish early, the setup actions of 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 in order to compensate for under-performance, take advantage of unexpectedly good performance, or add additional agents to deal with 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 that afford the addition or removal of agents. 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, they 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.

## Architecture

Our conceptual approach integrates CASPER/ASPEN into an architecture related to the classic three-tiered architectures (e.g. (Bonasso *et al.* 1997)), and incorporates planning, executive, and behavioral layers (Figure 2, (Sellner *et al.* 2006)). The planner forms an initial valid schedule, and is responsible for repairing and optimizing the schedule as execution proceeds. During execution, as task start times arrive, the planner dispatches them to the distributed executives, 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, it continuously provides updated state and task completion information to the executive.

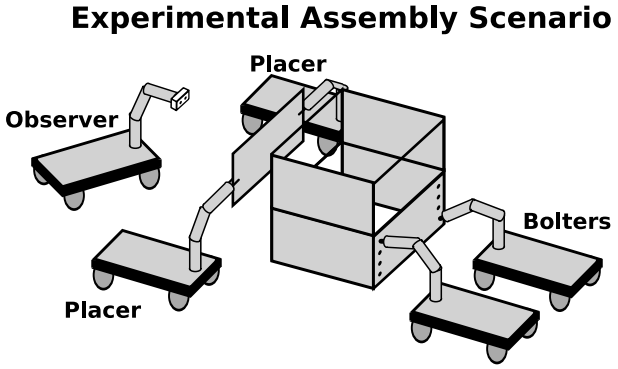


Figure 1: The notional structure being assembled in our experiments. Tasks are accomplished by fluid teams of agents filling required and optional roles.

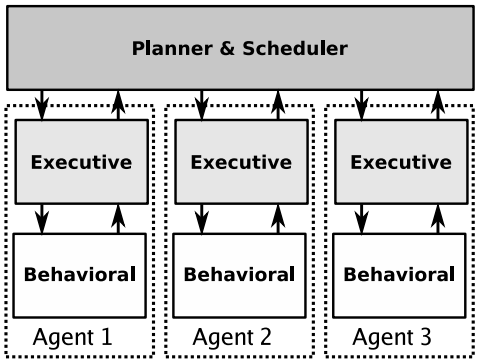


Figure 2: Our approach makes use of a centralized planner that dispatches tasks to distributed executive and behavioral layers.

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. 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.

Our initial implementation uses CASPER/ASPEN as the planning layer, a modified version of the CASPER executive as the executive, and a simulator that simultaneously models the behavioral layer and the stochastic nature of execution.

**Planner** We are currently using ASPEN (Chien *et al.* 2000b) as our planner, with the CASPER architecture interfacing between ASPEN and the executive. 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. ASPEN also maintains task and parameter constraint networks, allowing tasks to be temporally constrained with respect to each other, and task parameters to be constrained with respect to each other, as well as the values of resources and parameters in other tasks.

| 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 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. |

[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.

We make extensive use of the parameter constraint network when performing duration prediction. The duration of each task is constrained to be equal to the output of a function, within which we perform our prediction calculations. ASPEN's constraint network automatically calls the prediction function whenever one of its inputs has updated, and any resulting change in the task's predicted duration is propagated through the constraint network. Any conflicts or opportunities that arise as a result are handled during ASPEN's next repair/optimization pass. 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 several domain-specific heuristics and adding new repair and optimization methods that utilize the structure inherent in the optional role tasks of our domain.

We have added one repair and one optimization heuristic to ASPEN's default set. When repairing temporal or resource conflicts between tasks, the heuristic searches the possible combinations of roles for one which will complete the task quickly enough to avoid the conflict. If the conflict cannot be resolved by adding agents (and thus shrinking the task's duration), the heuristic instead attempts to use the minimum number of agents, in order to resolve conflicts in which agents are oversubscribed. Once a new set of roles has been selected, it searches for a set of agents to fill them that minimally disturbs other active teams.

If there are agents standing idle, our optimization heuristic attempts to make the best use of them. It first constructs an estimate of the critical path by examining the resource and ordering constraints of the schedule, beginning with the last task and working backwards. The heuristic then searches for an unstarted task on the critical path that the available agents can perform. If no such task exists, the idle agents are added to currently executing tasks with unfilled roles, with preference given to tasks on the critical path.

**Executive** The executive is based on CASPER's single-layer executive (Chien *et al.* 2000a) (Estlin *et al.* 2005), with several 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, and informs the planner when tasks complete. For instance, we extended the CASPER executive to start parameterized simulations as tasks are dispatched to the executive by the planner, then relay task state information back to the planner as relevant state changes. The original CASPER simulator has no ex-

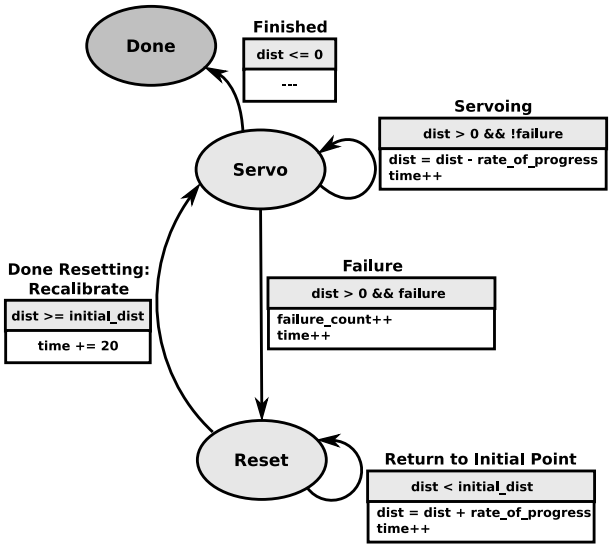


Figure 3: The stochastic simulation model for *Place_Panel*. This is used to simulate execution, and provides the state that is used for duration prediction during execution. In addition, this model is used offline to build the duration prediction tables.

ecution model, tasks always complete exactly when scheduled to do so, and updated state information is not relayed to the planner until a task completes.

**Behavioral** The behavioral layer is responsible for reactive control, and is dynamically configured by the executive. It can either interface with hardware or with a simulator. In our initial implementation, we use a simulator which models both the reactive nature of the behavioral layer and the stochastic nature of execution in the real world. In our future work we will be moving to a lower-level simulator and adding the behavioral layer we currently use in our assembly work (Sellner *et al.* 2006).

The current simulator maintains an independent model of each task 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 (see Figure 3) are precisely the models used to build our duration predictions. 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 of agents (such as manipulators or sensors) are not simulated, events such as a transport team becoming mired are.

## Duration Prediction

Duration prediction algorithms must be capable of estimating a task's remaining duration based on the current team and task state. Since the size of 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 (Table 1) may be performed by 49,876 distinct teams if 10 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.

Thus, the fundamental problem associated with duration prediction is to provide estimates of sufficient fidelity, while keeping computational and spatial requirements within limits. Our initial approach is straightforward: prior to execution, we 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 executes, we recompute its end time based on this table and the observed state.

The table is filled by repeatedly simulating the task and averaging the results. Each simulation run contributes data to every point in the table that is traversed during the run. When performing repeated simulations such as this, one must determine how many data points need to be accumulated for each state in order for the estimate to be sufficiently accurate. Without available ground truth, we are unable to determine a model's accuracy. Instead, we focus on the preciseness (variability) of the estimate. After each run, we calculate the 95% confidence interval of the corpus of observed data for the state (table entry) in question, and compare its width with the mean value. The ratio of confidence interval width to mean value is referred to as relative precision by (Law & Kelton 1982), and is a measure of how narrow our confidence interval is in the context of the task. Once the relative precision has decreased below a given threshold, or has not decreased in the past 10 simulations, we move on to the next state. Law and Kelton (pg. 293) recommend a threshold no greater than 0.15; we have used 0.05 in our work to date. The choice of this threshold is relatively arbitrary, but 0.05 has been used by a number of researchers (Bienstock 1996) (Pawlikowski, McNickle, & Ewing 1995). Note that since every simulation contributes data to many states, the number of additional simulation runs performed per state falls off rapidly as we iterate through the table.

While this table-based approach is sufficient for our simple test domain, it has two fundamental weaknesses: its spatial complexity scales very poorly, and it requires a massive amount of data to initialize, which is rarely available in real world systems. We are currently investigating various approaches to approximating this table in a spatially tractable way that supports extrapolation to provide estimates at unobserved states. Both model trees (Quinlan 1992) and local estimators such as LWPR (Vijayakumar & Schaal 2000) show promise.

One advantage of duration prediction is that it allows the planner to take advantage of opportunities provided by task under-runs. Consider a fragment of a construction scenario consisting of bolting a panel ($Bolt$) and preparing ($Prep$) and placing ($Place$) the next panel (Figure 4). $Bolt$ requires only one agent, and must be completed before the next $Place$ begins. $Prep$ is a setup task for $Place$, and is

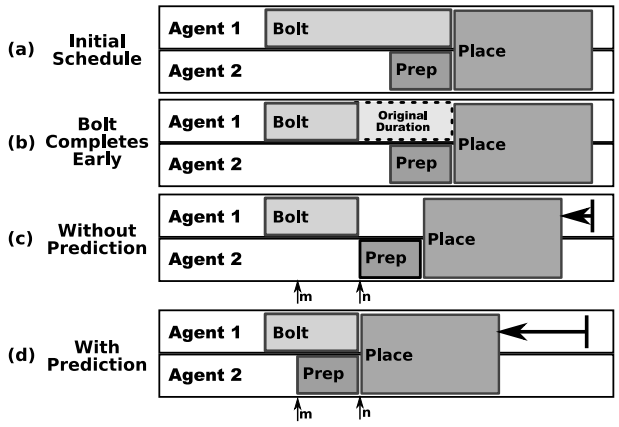


Figure 4: Continually predicting the remaining duration of executing tasks allows the planner to make the fullest use of opportunities presented by task under-runs.

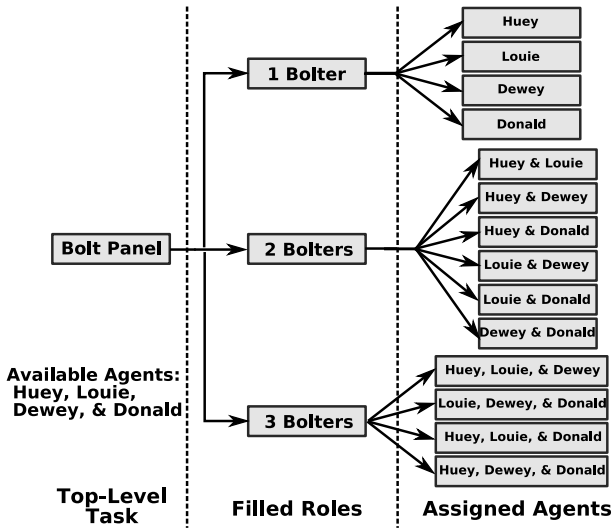


Figure 5: The *Bolt_Panel* task fastens an already-placed panel to the structure. With four agents, there are the above 14 ways to perform the task. This tree branches rapidly in the presence of more roles and/or agents. Our live task modification heuristics search the "Filled Roles" level when adjusting the duration of a task and portions of the "Assigned Agents" level when resolving agent oversubscription conflicts.

constrained to end at $Place$'s start time. $Place$ requires two agents, while $Prep$ needs only one. The initial schedule is depicted in Figure 4a.

If $Bolt$ completes early (Figure 4b), $Prep$ 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 4c). 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 further 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 4d).

## Live Task Modification

For a robotic team to exhibit the same fluidity that human teams achieve, the planning/execution system must not only predict the future state of tasks but must also be able to modify currently executing tasks. Existing planning/execution systems do not provide sufficient mechanisms for the planner to manipulate the teams assigned to live tasks. This reduces the complexity of both the planner and the executive, but 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 that *Transport_Panel* may be performed by 49,876 distinct teams if 10 agents are available).

The repair and optimization heuristics discussed in the "Planner" section use the structure inherent in tasks with optional roles to intelligently search the space of teams. A task with optional roles can be represented as in Figure 5: in the second level a selection of which roles will be filled is made, while specific agents are assigned in the third level of decomposition. We currently define agents to be homogeneous: all agents are able to perform all available roles. Given this, the selection of the roles themselves determines the task's expected duration. Thus, when attempting to resolve a conflict by reducing a task's duration, the heuristics search only the second ("Filled Roles") level of the task description, selecting specific agents only after a role set has been chosen. If instead the heuristic is attempting to resolve an agent oversubscription conflict by minimizing the number of agents used, it selects the role set requiring the fewest agents, then searches the third level for a non-conflicting set of agents. In this way, we avoid an exhaustive search of the team space.

However, if agents have differing levels of skill, task duration and agent assignment become interrelated. Efficiently searching such team spaces is an area we are investigating, although this paper addresses only homogeneous agents. It may be possible to build a network, rather than a tree, with connections between nodes differing by only one agent. Searches could then be concentrated on those teams that could be formed with the least amount of disturbance to the existing schedule. It may be possible to enable an initial coarse search by layering a meta structure representing the different optional roles on top of such a network.

To illustrate the usefulness of live task modification, consider a scenario where the initial schedule consists of two

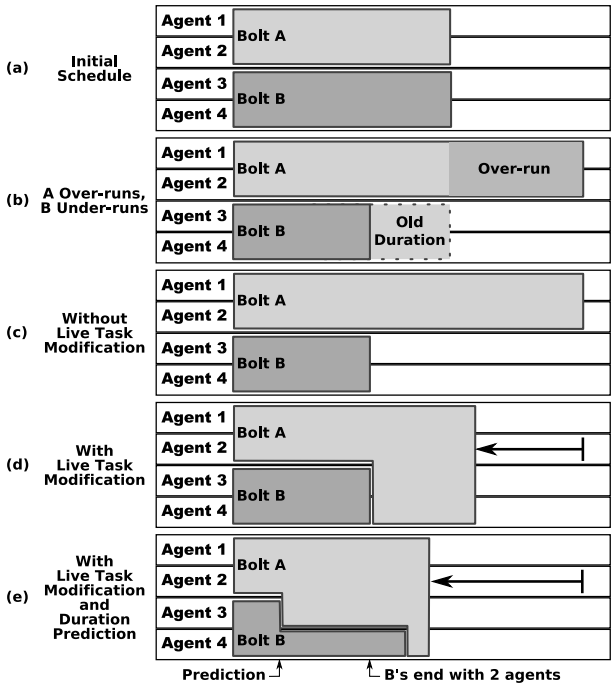


Figure 6: Reassigning agents during task execution allows the system to adjust resource allocation to reflect the realities of execution.

$Bolt$ tasks: $BoltA$ and $BoltB$ (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.

As discussed in the "Architecture" section, the current implementation of live task modification is used by the planner's repair methods to resolve agent over-subscription 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.

# 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 does alone, yielding executed schedules on average 31.8% shorter than was achieved without these capabilities. We discuss here the current scenario, experimental design, and results.

## Scenario

Our initial scenario is a subset of the domain described in the "Approach" section (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 concurrently, 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.

*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 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 5).

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

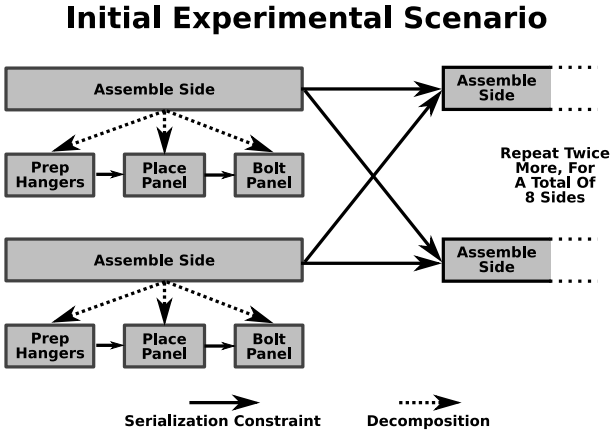


Figure 7: The scenario used in our experiments. A total of eight *Assemble_Side* tasks are included, serialized by pairs, since teams cannot work on adjacent sides of the structure without risking interference.

### Experimental Conditions

To experimentally validate our beliefs about the effectiveness of duration prediction and live task modification, we conducted an experiment comparing the four combinations of the two conditions. 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:

1. **Baseline** The baseline condition utilizes neither duration prediction nor live task modification. All tasks are set initially 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. If a task finishes early, its duration on the planner's schedule is shrunk when it has completed. The planner utilizes repair and optimization heuristics that modify teams, but only with non-executing tasks. During execution, the following actions are performed before each time step:
   (a) **Right-shift:** Move any non-executing tasks with agent subscription ("resource") conflicts into the future, until the conflict is resolved.
   (b) **Left-shift:** Move all non-executing tasks to the earliest time consistent with temporal and resource constraints.
   (c) **Optimize:** Run 20 iterations of the planner's iterative optimization algorithm, without considering executing tasks.
   (d) **Repair:** Perform iterative repair until all conflicts are resolved. Executing tasks may *not* be modified.
2. **Prediction** This condition adds task duration prediction to the baseline scenario. Executing tasks still may *not* be modified in this condition. As a result, the only source of advantage with respect to the baseline is the timely scheduling of setup tasks, as described in the "Approach" section (illustrated in Figure 4). This occurs within the left-shift and optimize steps of the replanning algorithm. No additional heuristics are employed, since duration prediction simply updates the contents of the planner's current schedule, allowing the baseline system to react appropriately.
3. **Live Modification** In this condition, the planner is allowed to modify executing tasks by changing the assigned agents. 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. As discussed in the "Approach" section, additional heuristics are used in the optimize and repair steps to reason about the benefits of modifying executing tasks.
4. **Prediction and Live Modification** In this condition, both duration prediction and live modification are enabled. Agents are shifted between tasks during both the optimize and repair steps in response to changes in the predicted duration of tasks.

### 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 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 effectively the agents were utilized. "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.

## Discussion

The most notable aspect of the data in Table 2 is the average makespan, which also is depicted in Figure 8. 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

| | 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) |
| 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.

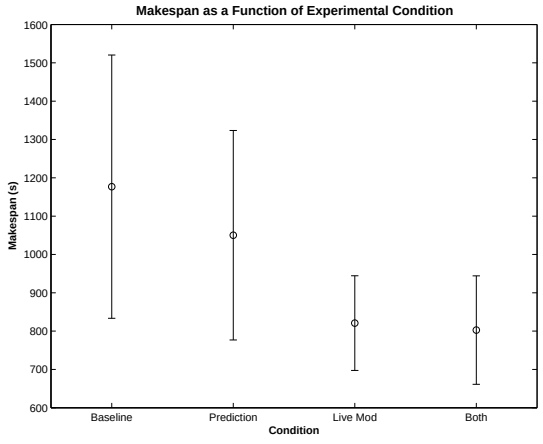


Figure 8: The average makespan and standard deviation for each of the four test conditions.

scenario, setup tasks occur only 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 condition. In addition, domains with more setup tasks, such as those that require setup tasks when adding an agent to a live team, should result in larger relative improvements in the combined condition.

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.

As an initial 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. The above results include this buffer. Significant work on the minimization of jitter has been performed in fields such as control systems theory (e.g. (Mansuri 2003) (Skormin, Tascillo, & Nicholson 1993)). 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 doubles when live task modification is enabled. This indicates that about half of the changes in agent assignments being made in the live task modification conditions are affecting executing tasks. This primarily occurs when idle agents are assigned to executing tasks on the critical path.

## Future Work

Proactive replanning is an area rich in potential research. Initially, 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.

We also plan to extend our work to support non-instantaneous agent transfers. That is, the addition of an agent to a team takes time, and potentially will slow the remainder of the team during the process. In addition, the agent being transferred may 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 their completion must coincide with the start of 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.

Finally, 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.

## Conclusion

Inspired by the fluidity of human teams, we have developed an initial proactive replanning system capable of predicting task durations and modifying currently executing tasks. While many of the benefits of proactive replanning are realized only 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.

Duration prediction and live task modification allow the system to execute plans more efficiently than otherwise possible. We have conducted experiments in simulation demonstrating reductions in the executed makespan ranging from 11% - 32%, depending on which aspects of proactive replanning are implemented. 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).

Bienstock, C. C. 1996. Sample size determination in logistics simulations. *International Journal of Physical Distribution and Logistics Management* 26(2):43–50.

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*.

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)*.

Law, A. M., and Kelton, W. D. 1982. *Simulation Modeling and Analysis*. New York: McGraw-Hill.

Mansuri, M. 2003. *Low-Power Low-Jitter On-Chip Clock Generation*. Ph.D. Dissertation, University of California, Los Angeles.

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*.

Pawlikowski, K.; McNickle, D. C.; and Ewing, G. 1995. Coverage of confidence intervals in sequential steady-state simulation. In *Proceedings of the 1995 EUROSIM Congress*.

Quinlan, J. R. 1992. Learning with continuous classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, 343–348. Singapore: World Scientific.

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)*.

Skormin, V. A.; Tascillo, M. A.; and Nicholson, D. J. 1993. A jitter rejection technique in a satellite-based lasercommunication system. In *Proceedings of NAECON 1993*, volume 2, 1107–1115.

Vijayakumar, S., and Schaal, S. 2000. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, volume 1, 288–293.