Proactive Replanning for Multi-Robot Teams

Brennan P. Sellner
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
bsellner@andrew.cmu.edu

June 1, 2006

Committee members:
Reid Simmons (chair)
Tara Estlin
Sanjiv Singh
Stephen Smith

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. The goal of this research is to examine 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 continuously predicting the remaining task duration, a proactive replanner is able to adjust to upcoming opportunities or problems before they manifest themselves. One way in which it may do so is by adjusting the allocation of agents to the various executing tasks by adding or removing agents, which allows the planner to balance a schedule in response to the realities of execution. We propose to develop a planning/scheduling/execution system that, by supporting duration prediction and adaptation, will be able to execute complex multi-robot tasks in an uncertain environment more efficiently than is possible without such proactive capabilities.

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. Our initial system does not operate in real time, nor with actual hardware, instead interfacing with a simulator and allowing unlimited time for replanning between time steps. We propose to characterize the applicability of this approach to various domains, extend our algorithms to support more complex scenarios and to address shortcomings we have identified, and optimize the algorithms with respect to both computational complexity and the makespan of the final executed schedule, with the goal of bringing the advantages of duration prediction and task modification to real-time planning/execution system. We will evaluate our approach to proactive replanning both in an extensive series of simulated experiments and in a real-time assembly scenario using actual hardware. We hypothesize that proactive replanning can be performed in real time while yielding significant improvements in overall execution time, as compared with a baseline repair-based planner.

[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.

Contents

1 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 are interested in creating 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 anticipated event occurs. As part of this replanning, the system will be able to perform *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 will yield 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, which makes use of iterative repair and optimization stages to build and optimize valid schedules. The planner also supports hierarchical task decompositions, which we leverage in our approach to live task modification. We will extend this planner's repair and optimization algorithms and heuristics to support proactive replanning, specifically task duration prediction and live task modification, in an efficient enough manner to allow its use in a real time system. 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.

Both duration prediction and live task modification enable more efficient execution of plans than can be achieved by planning/execution systems not incorporating them. By dynamically predicting task durations, the planner may form a more efficient plan, as setup actions (such as prepositioning agents) may be performed prior to the actual occurrence of the conflict or opportunity. For instance, assume Agent A is placing a column into a hole, and Agent B is tasked with filling the hole with quick-setting cement (Figure 1, right). The cement must be mixed just before it is poured. The mixing initially is scheduled to take place during the final stages of the column placement. If Agent A finishes early, Agent B may immediately begin pouring cement, but only if the early finish was predicted in enough time for B to start mixing early.

Modifying an executing task by adding or removing agents allows the system to adjust a task's expected duration or reliability without expensive rescheduling, and to rebalance resources between tasks that are performing better or worse than expected. For instance, assume Agents A and B each are placing a column. A column placement may be performed by one agent, but will be completed more rapidly if two agents participate. If Agent B completes its placement first, live task modification allows the planner to reassign B to assist A in completing its task (Figure 1, left). Traditional planning/execution systems do not allow such modification, and would leave B idle.

When used together, the effects are magnified, as teams are fluidly reconfigured in response to predicted problems or opportunities. For example, assume Agents A and B are working together to place a column, while Agent C is doing the same on its own. If Agent C underperforms, such that the predicted duration of its task increases significantly, the planner can use live task modification to shift Agent B to assist C, reducing the overall schedule's makespan.

The problem of duration prediction is one of approximation. Duration prediction algorithms must be capable of mapping the many combinations of task state and teams of agents to a predicted task duration or distribution across durations. The fundamental problem associated with duration 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, which spans the team / state space, then refer to it at execution-time. As the task state evolves during execution,

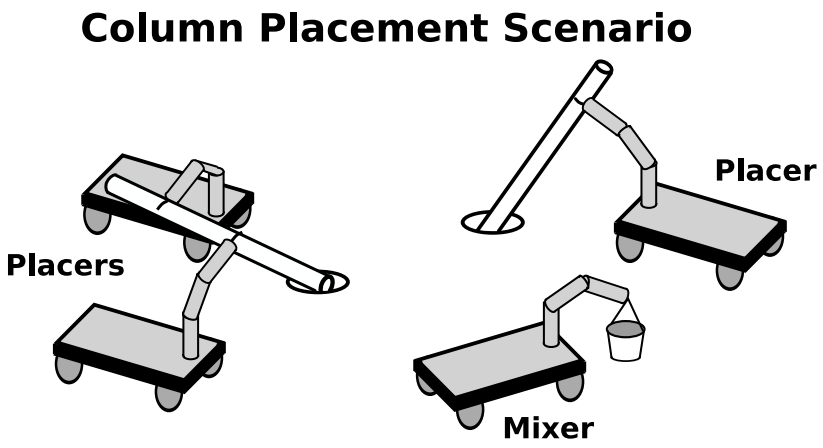

Figure 1: A simple scenario in which duration prediction and live task modification are useful.

we update our duration predictions from our table of estimates. However, this will not scale well, and is lacking in expressivity. The thesis work will develop approximation algorithms that will represent higher fidelity estimates and collapse portions of the prediction function's domain in order to reduce the approach's spatial requirements.

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 and a multiplicity of tasks which much be managed to make the addition and removal of agents possible, both of which must be addressed within the computational constraints of a real-time system. We propose to develop heuristics to search rapidly through possible teams by taking advantage of the structure unique to tasks that afford live task modification. In addition, we will develop planning algorithms to incorporate agent addition and removal, as well as setup, tasks into the planner's repair and optimization calculations in an efficient manner.

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 up to 30.3% reductions in makespan, while duration prediction yields gains of 10.8%. When used in combination, makespan is reduced by 31.8% on average. In the thesis, we will characterize, extend, and optimize our algorithms. Elements of the gains provided by duration prediction and live task modification are domain- and scenario-specific – we will characterize how the composition of a domain affects the gains possible when utilizing our approach. We will extend and optimize this work by developing planning and execution algorithms efficient enough to allow a real-time system to make use of duration prediction and live task modification, while refining them and examining in depth the opportunities and consequences that they represent.

Hypothesis Proactive replanning can be performed in real time while yielding significant improvements in overall execution time.

2 Related Work

Prior work related to proactive replanning and our approach to it falls into three categories: implemented planning, scheduling, and execution systems; task duration prediction algorithms; and work on modifying live tasks or teams. We will discuss how existing systems address the concepts of proactive replanning. We will then survey a variety of algorithms related to the prediction of task duration or similar properties, and wrap up with a discussion of work related to live task modification.

2.1 Planning, Scheduling, and Execution Systems

When considering the architecture of a proactive replanning system, there are two major components: the planner/scheduler itself and how it interacts with the lower-level elements of the system. We will discuss existing planners, then cover architectures into which they have been integrated.

2.1.1 Planners/Schedulers

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. Many existing planners are unsuited to such domains.

Classical planning approaches such as those using the STRIPS formulation [28] are of limited use due to their modeling of actions as instantaneous transitions between states. Since the core of our approach to proactive replanning is to monitor task progress and modify teams during execution, the planner must model actions as having duration and being interruptible. Some classical planning approaches have incorporated durative actions. For example, the PDDL2.1 Level3 language [30] extends the classical PDDL language to allow the specification of durative actions, preconditions and effects at the start and end of durative actions, and invariant conditions. However, PDDL2.1 maintains the representation of an action as a transition between two states, and time points within the transition cannot be specified. This makes any expression of duration prediction or live task modification impossible, and renders planners such as TLplan [4], TGP [60], O-Plan [18], and SHOP2 [52] unable to take advantage of the benefits of proactive replanning.

Conformant and conditional planners also are not applicable to domains amenable to proactive replanning. Conformant planners such as CGP, [59], CMBP [16], and C-PLAN [26] produce plans containing only actions that will lead to the goal regardless of the current state. In the complex, durative domains that make up the majority of those in which proactive replanning is useful, this is far too restrictive, as such actions almost never exist. Conditional planners such as SGP [65], MBP [8], and GPT [11] attempt to build plans with branches based on the results of sensing actions. Again, the complexity of the domains we are considering becomes an insurmountable issue, as the set of all possible state traces becomes much too large to enumerate.

Alternative 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 predicted 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 much more 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 [15], EUROPA [31] (the successor to HSTS [49]), and IxTeT [42]. 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 in the predicted schedule and other flaws. 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, and heuristics aware of the structure of tasks with optional roles had to be developed. We have built our initial implementation of proactive replanning using ASPEN as a base.

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.

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

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

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.

2.1.2 Architectures: Integrating Planning and Execution

Architectures serve to tie together 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.

Due to the need to rapidly re-evaluate team composition during task execution, the "Sense-Plan-Act" approach exemplified by the PLANEX system used on Shakey [27] is not applicable. Similarly, batch planning such as that used in the Remote Agent experiment [7] [50] is unable to take advantage of the dynamic nature of proactive replanning domains.

"Reactive planning" techniques, in which the executive is endowed with limited planning capabilities and (if present) the planner is used similarly to batch-planning, are not applicable to proactive replanning, as they do not predict state values very far into the future. Since one of proactive replanning's benefits is the ability to prevent predicted problems, the long-term projection of state values is a crucial component. Examples of reactive planners include those which simply choose among available tasks and perform some failure recovery, such as RAP [29], ESL [33], and TDL [58] and transformational planners such as XFRM [5]. While such techniques would be useful in an executive as part of a proactive replanning system, a longer-term scheduling approach also is required.

Three-layer architectures such as 3T [10] and ATLANTIS [32] may be adaptable to proactive replanning, but the ties between their planning and executive layers generally are not tight enough to support continuous replanning and live task modification.

A number of architectures, such as IPEM [3], ROGUE [36], and SIPE [66] have implemented continuous planning, in which planning and execution are seamlessly interleaved. However, IPEM and ROGUE are purely classical planning approaches, while SIPE does not explicitly represent time, rendering them not applicable to proactive replanning.

LAAS [2][3] and CLARAty [53] [64] 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 [14] and CLEaR [24]. CASPER uses the ASPEN [15] planner and a simple executive, while CLEaR adds a TDL-based [58] 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, [14] 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 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 [20] [51], which advocates the use of planning as the core of each level of abstraction,

[3] IxTeT is the planning component of LAAS.

from mission planning to reactive execution. "Planner" and "executive" modules are both implemented with planners operating on 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.

2.2 Duration Prediction

Although to our knowledge no existing planning/execution systems dynamically predict the remaining duration of a task, much research has been performed on various aspects of prediction. We are interested in predicting a continuous metric (remaining duration) given a (large) collection of continuous and discrete state inputs (the current task and team state), under the Markovian assumption.

A sizable percentage of the literature is instead focused on either recognizing an action (e.g. a discrete state) or predicting the plan being used by the human or agent (e.g. predicting a discrete goal or approach to the goal), with no assumption that the prediction function is Markovian. For instance, Das et al. [19] attempt to predict a user's next discrete action by matching the observed action sequence with a dictionary of sequences. Brown et al. [12] model users with Bayesian Networks in order to predict the user's current goal based on past actions. Such approaches are ill-suited to our task, as they are applicable only to relatively small input state spaces and predicted goals or plans.

The range of our prediction function is far too large for approaches using Markov models, such as the work of Li and Okamura [45], which uses Hidden Markov Models (HMMs) to recognize and classify user motions, or Hovland and McCarragher [37], who utilize HMMs to monitor assembly given changes in tool contact states. Dixon [21] uses Continuous-Density Hidden Markov Models (CDHMMs) to predict what task a user is attempting to program a robot to perform. Duration prediction is a problem of function approximation, not one of classification.

We have previously [57] modeled the expected duration of a task that may be attempted multiple times by either a human operator or an autonomous system. This work focused on the allocation of the task to either human or autonomous control, and resulted in an estimate of task duration for the two possible assignments of the next attempt at the task. Although the work could be used to integrate humans into a duration prediction algorithm, it is not directly applicable.

Belker et al. [6] use model trees [39] [55] to predict the expected duration of a task, given previous observations. This approach provides a piece-wise linear regression model of the duration function. This may well be applicable to our problem, although as formulated it provides a single estimate of duration; it is unclear whether it could be easily extended to predict distributions across task duration.

Due to the large potential domain and range of our duration prediction function, we have instead chosen to model it at a lower level, by building a direct mapping from current state to an estimate of the remaining duration (or an estimate of a distribution across remaining duration). As a result, work in function approximation is germane to our approach. For instance, Meuleau et al. [48] use kd-trees to condense plateaus in a value function to achieve a more compact representation. This is not directly applicable to our problem, as remaining duration often scales smoothly with the task state variables, rather than forming plateaus. A thorough survey of the function approximation literature is beyond the scope this section.

2.3 Live Task Modification

To the best of our knowledge, live task modification has not been implemented as or considered for a component of a planning/execution system. At most, existing planners are able to abort currently-executing tasks. However, a variety of research topics touch upon different aspects of live task modification.

While at first live task modification may appear similar to the multi-robot task allocation problem (MRTA), it is in fact complementary. Live task modification is concerned with adding or removing agents to or from executing multi-agent tasks in response to observations, while MRTA is generally considered to be a static optimization problem [34]. There has been some work on "dynamic" MRTA [62], but the authors define a dynamic MRTA problem as one in which new tasks arrive at random times, and must be accommodated by the scheduler. Their work still makes the implicit assertion that any task which has begun execution is immune to modification by the planner/scheduler. Mataric et al. [47] discuss MRTA incorporating the possibility of agents taking an opportunistic strategy in which they may release their current task in favor of a different one. However, all tasks are performed by a single agent, precluding live task modification, since there are no teams to modify.

Many swarm, or emergent, approaches to multi-agent systems [17] [46] can be considered to be conducting live task modification, as agents may be freely added to or removed from the team without dramatically affecting performance. However, we are interested in problems in which teams of agents perform highly coordinated tasks, rather than the loosely coupled foraging, inspection, or mapping tasks common to swarming algorithms.

The MOVER architecture [38] provides the ability to procedurally add, remove, or substitute agents on a team. However, the conditions for such team changes must be prespecified in a policy for each task. There is no planner using this capability to improve the overall performance of the scenario; instead, teams are reactively modified based on local information and policy.

Stone and Veloso's work [61] supports the periodic exchange of roles between members of a team, but only addresses domains in which there is a single team. In addition, there is no tight inter-agent coordination in their domain (robotic soccer). As a result, agents may freely move between roles with no need for explicit addition or removal tasks. A significant body of work exists that addresses the dynamic assignment of roles within a robotic soccer team (e.g. [23] [63]), but is subject to the same domain characteristics as Stone and Veloso's work. Live task modification addresses the dynamic (re)allocation of agents between many teams in order to efficiently perform a scenario, rather than reshuffling agents within a team.

2.4 Summary

There are three broad areas of work related to proactive replanning: existing planners and planning/execution architectures, duration prediction algorithms, and approaches to live task modification. The ASPEN and EUROPA planners appear to be the best suited to proactive replanning applications, due to their support for durative actions, temporal constraints, exogenous events, metric resources, and continuous replanning. The CLARAty/CASPER and IDEA architectures (which incorporate ASPEN and EUROPA, respectively) are both good fits for proactive replanning, as they provide the potential for the tight planner-executive ties necessary for successful proactive replanning. Work in the area of function approximation is best suited to duration prediction; most approaches to prediction take a classification approach and provide discrete outputs, rather than estimating a continuous function. A variety of approaches similar to live task modification exist, but no planning/execution systems incorporate the dynamic shifting of agents between teams in the process of executing tasks.

3 Approach

In order to enable true proactive replanning, 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 balance resources between tasks to compensate for underperformance, 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. In order to focus on the fundamentals of proactive replanning, we do not consider the distributed case, instead treating the planner as a centralized process that formulates a scenario-wide schedule and dispatches tasks to the individual agents for execution.

3.1 Architecture

Our approach is based on an architecture related to the classic three-tiered architectures (e.g. [10]), and incorporates planning, executive, and behavioral layers. The planner forms an initial valid schedule, and is responsible for repairing and optimizing the schedule as execution proceeds. The executive acts as an intermediary between the planning and behavioral layers, relaying data and managing the execution of tasks. The behavioral layer (either simulated or interacting with real hardware) executes tasks in a reactive manner and collects information on the state of execution. This data is filtered by the executive and passed on to the planning layer.

During execution, the planner dispatches tasks to the executive 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, 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 resulting in schedule conflicts or opportunities. The planner then acts to repair and optimize the schedule in order to adapt to the realities of execution, by using a variety of heuristics, including those that make use of live task modification to change the composition of teams currently performing tasks.

3.2 Duration Prediction

The ability to predict the remaining duration of a task given the current execution and team state allows the planner to better take advantage of opportunities (such as tasks completing early) or accommodate upcoming problems (such as tasks over-running). In the presence of setup actions such as the repositioning of agents, warming up of motors, or calibration of instruments, waiting until an opportunity manifests itself before adjusting the schedule can lead to delays in execution. In contrast, if one can predict that a task will complete early, subsequent setup actions can be performed *prior* to task completion. Prediction provides similar benefits when tasks over-run, although setup actions need not be present for the benefits to manifest themselves in this case. In our approach, this prediction takes the form of a mapping from current task and team state to expected task duration and/or resource consumption. Our work to date concentrates on task duration, making the simplifying assumption that resource usage is directly correlated with task duration.

3.2.1 Prediction Examples

To illustrate the usefulness of duration prediction in a planning/execution system, let us consider a problem consisting of five tasks (A, B, C, D, and E) and two agents (1 and 2). The initial schedule and agent-task assignments are diagrammed in Figure 2a. Suppose Task A were to over-run and take more than $Length(B) + Length(D)$ seconds to complete. If the system were to wait until Task A's scheduled finish time (Figure 2b, Point y) to determine that an over-run had occurred, time would be wasted. If, instead, the planner were able to predict the over-run by Point x, it would allow the rescheduling of D to achieve the minimum achievable makespan (Figure 2c). Smaller gains are possible if the planner's prediction occurs later than Point x, but at any point more than $Length(E)$ seconds before A's true completion time.

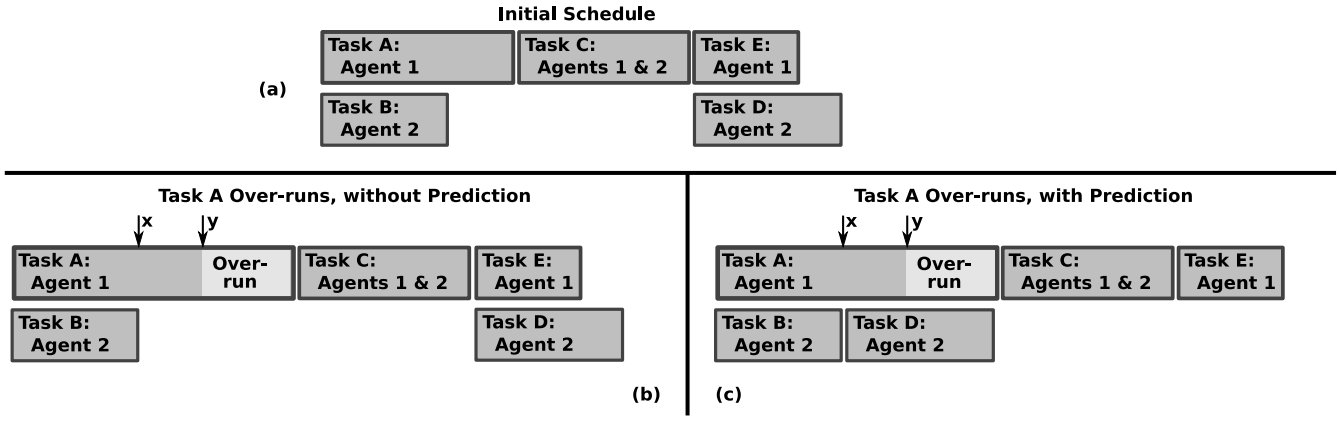

Figure 2: If the planner is able to predict that Task A will take at least $Length(B) + Length(D)$ by Point x, Task D can be rescheduled (c). If the planner were to wait until the task actually overran at Point y, inefficiencies would occur (b).

Duration prediction also allows the planner to take advantage of opportunities provided by task under-runs. Consider a scenario consisting of Tasks A, B, and B' (Figure 3). A requires only Agent 1, but must occur before B, while B' is a setup task for B, and is constrained to meet B (e.g. B''s end time must equal B's start time). B requires both Agents 1 and 2, while B' requires only Agent 2 (such a setup action could involve repositioning Agent 2, warming up a motor, etc.). The initial schedule is depicted in Figure 3a.

We can see that if Task A completes early, B and B' 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 B' immediately upon A's completion (Figure 3b). However, this is inefficient, as B' may be executed in parallel with A. If the planner were able to predict A's true completion time at any point prior to its occurrence at Point n, it would be able to start B' early, realizing a reduction in makespan. Ideally, the planner would be able to make the prediction prior to Point m (that is, $Length(B')$ seconds before A's early completion), allowing B to be scheduled immediately after A and B' to be executed entirely in parallel with A (Figure 3c).

As we can see, the addition of duration prediction provides the planning/execution system with additional flexibility, 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 actually occur – they are not taking advantage of the optimizations possible when duration prediction is available.

3.2.2 Prediction Methods

Although duration prediction is demonstrably useful, predicting the remaining duration for an executing task in a stochastic environment is far from trivial. The root problem is modeling the task with sufficient fidelity, yet in a computationally and spatially feasible fashion. The tension between these two requirements is significant, as simply enumerating the entire state space for a broad selection of tasks, much less storing a significant amount of data associated with each state, will quickly become intractable.

The fidelity of the task model will determine the accuracy of the duration predictions that may be made. The underlying distributions are often quite complex, and first-order approximations may yield disappointing results. In particular, the distribution across remaining duration for a particular state is often multimodal. When a task includes infrequent nonterminal failures, such as a joint becoming temporarily jammed or an agent becoming mired in sand, the duration distribution develops distinct modes associated with the number of failures (see Figure 4). Knowledge that a duration distribution is multimodal can be exploited in a number of ways, including only scheduling a task for a span of time coincident with one of the modes, rather than using the overall mean, which may fall into a "trough" of the distribution. In addition, the form of the distribution (or each mode of the distribution) may vary widely, from nearly normal distributions to those with hard minimums, such as the example in Figure 4.

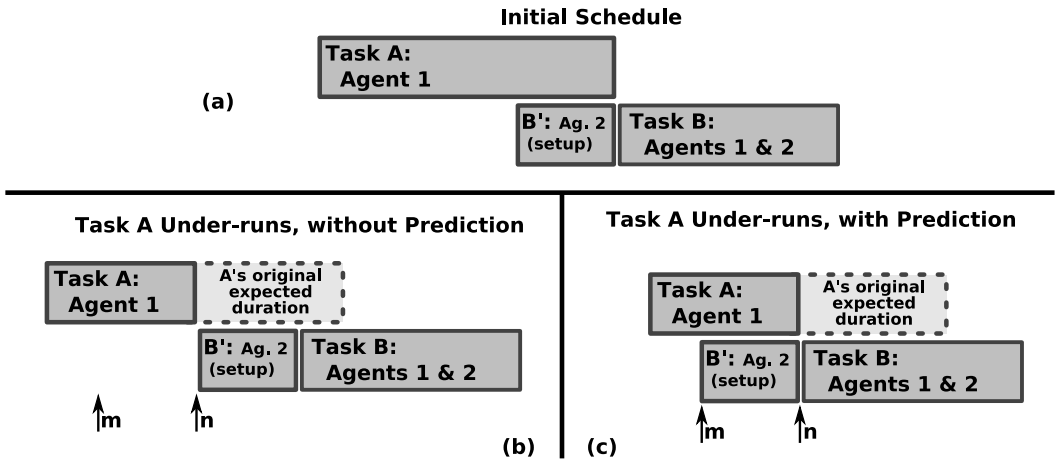

Figure 3: If the planner's first indication of an under-run were the actual completion of Task A at Point n, only a suboptimal schedule would be achievable (b). If instead the planner is able to predict that Task A will finish early, Task B' could be scheduled to partially overlap A. If the planner's prediction were made at least $Length(B')$ units before Point n (Point m in (b) and (c)), B' could be reschedule to completely overlap A, yielding the minimum achievable makespan (c).

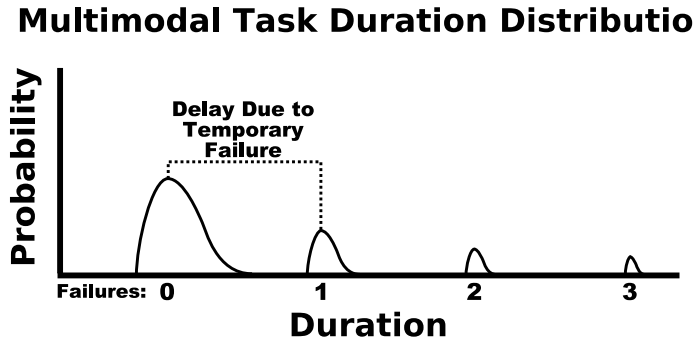

Figure 4: A simple multi-modal task duration distribution. Multiple modes arise primarily when infrequent nonterminal failures occur, leading to a series of modes such as those depicted here.

Given that a distribution such as that in Figure 4 represents only a single point in the task state space, the overall function to be modeled can become arbitrarily complex, precluding the development of a closed-form solution. Various approximations can be made in the task model, such as storing only the mean of the distribution, but only at the cost of a significant loss of fidelity. The fundamental problem is to arrive at an appropriate balance between fidelity, computational cost, and spatial requirements.

A naive approach would be to build an empirical distribution across duration for each possible state of each task by collecting the results of a series of simulations. This would provide the planner with a high-fidelity estimate of the true, arbitrarily formed, duration distribution. Unfortunately, this would also generate enormous amounts of data, which quickly would become unmanageable.

There are a number of potential approaches to providing high-fidelity, yet efficient, duration estimates, including generating duration distributions dynamically, reducing the size of each duration estimate, and reducing the overall number of estimates. While aesthetically appealing, online simulation as part of a real-time system is computationally infeasible. In the presence of highly stochastic tasks, a significant number of simulation runs would need to be performed in order to yield any sort of reasonable estimate. Since many such predictions must be performed in the course of a planning session, performing the simulations online will not be a feasible solution for most domains in which we are interested.

The size of the duration estimate for a specific state may be reduced in a number of ways, including storing only the mean and variance, fitting parameterized continuous distributions to the underlying discrete distribution, and attempting to find arbitrary discrete distributions that can be used as basis functions for many states. This is a complexity/capability tradeoff: in the presence of underlying multimodal duration distributions, the most efficient approach is to store only the mean and variance of the overall distribution; however, not representing the distribution's modes may lead to suboptimal schedules. Similarly, using discrete distributions as basis functions may lead to a more accurate estimate, but will consume more spatial and computational resources than utilizing continuous distributions such as the gamma or normal functions.

Finally, the number of estimates can be reduced by coarsening our discretization of the state space or attempting to parameterize regions of the space. It may be possible to form higher-level basis functions that are used to estimate a range of states. The state to duration mapping spaces in our domains often exhibit a significant amount of structure, which may be used by more domain-specific approximation functions.

Our initial implementation takes a naive approach to the problem, and builds a table mapping each point in the state space to an estimate of the mean of the underlying duration distribution. This is spatially tractable for the small- to medium-sized tasks in our initial scenario, but will not scale well, nor can it take multimodal distributions properly into account. The thesis work will investigate different approaches to duration prediction and how to represent multimodal duration distributions in a spatially and computationally tractable manner.

3.3 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[4], for the planner to manipulate the teams assigned to live tasks. This reduces the complexity of both the planning task 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.

We propose to allow the planner to modify executing tasks by adding or removing resources (agents) to or from the team during execution. This type of live task modification increases the planner's flexibility in its primary area of concern: the assignment and scheduling of resources (agents). Traditionally, planners are responsible for the assignment of resources and high-level sequencing of tasks, while the executive attempts to carry out the provided plan. Following this division of responsibility, any change in the team assigned to a task falls within the purview of the planner.

In contrast, other types of live task modification fall more properly to the executive. For instance, modifying the parameters of a task, such as the target trajectory, allowable tolerances and time outs requires intimate knowledge of the task and the specific approach being taken that is more readily available within the executive than the planner. Similarly, determining when to change execution states, such as the triggering of a low-level recovery action, is a decision that can be made reactively in response to the detailed data easily available to the executive.

3.3.1 Optional Roles

Task modification by changing the team performing the task is predicated upon the existence of tasks which 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. 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.

[4]For instance, IxTeT [35] [44] allows the planner to abort existing tasks, but this is the extent of the control the planner may exert over the executive.

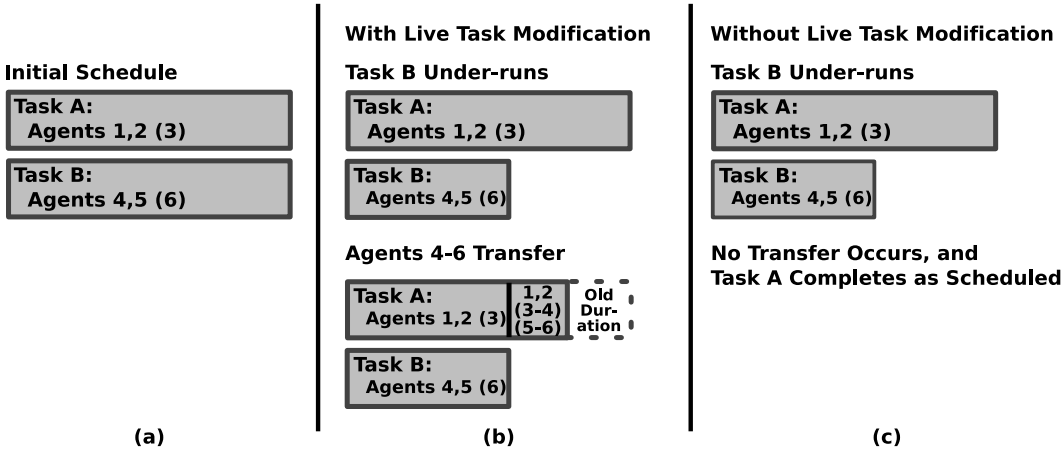

Figure 5: Reassigning agents during task execution allows the system to take advantage of opportunities as they arise. Agents performing optional roles are denoted in parentheses.

For instance, an assembly scenario on Mars might incorporate a 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 subteam 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.

With proper design of the execution system, optional roles can be designed naturally into a wide range of tasks, giving the planner significant flexibility when scheduling.

3.3.2 Examples

To illustrate the usefulness of live task modification, consider the following example. The initial schedule consists of two transport tasks: A and B (Figure 5). For the purposes of illustration, we will disregard tow roles, leaving two required and two optional transporter roles, as well as two optional scout roles. Initially, let Agents 1, 2, and 3 fill transporter roles in Task A, while Agents 4, 5, and 6 do the same in Task B (Figure 5a). If, during execution, Task B finishes early, its agents may be moved to Task A to fill the remaining transporter and two scout roles, which will reduce its run-time, as well as the final schedule's makespan (Figure 5b). If live task modification were not available, Task A would be left to run its allotted course while Agents 4-6 stood idle (Figure 5c).

If both duration prediction and live task modification are available, the system becomes even more flexible. In the example above, instead let Task A over-run and B under-run (Figure 6). This could occur if the two teams were to take routes that prove to be of varying difficulty. If the planner is able to predict this during execution, Agent 6 may be reassigned from B to A to fill Task A's final transporter role, balancing the actual execution times of the tasks and reducing the schedule's makespan.

Live task modification clearly allows the planning/execution system to execute a schedule more efficiently in a dynamic, uncertain environment. However, there is a reason live task modification has not been extensively studied: it results in significant additional complexity in both the planner and executive.

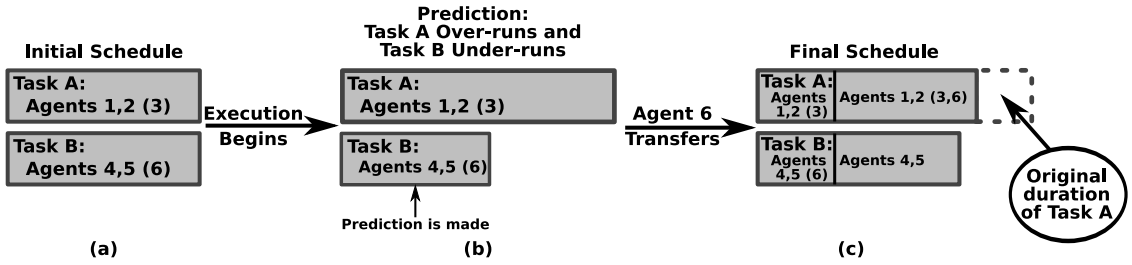

Figure 6: Reassigning agents during task execution in response to task duration predictions allows the system to adapt to the realities of execution. Agents performing optional roles are denoted in parentheses.

3.3.3 Task Modification and the Planner

There are two primary sources of added complexity in the planner when using live task modification: a vastly widened search space, and the need to support the myriad additional tasks and constraints associated with live task modification.

Live task modification is primarily useful when employed with tasks containing optional roles, since the flexibility added by its use is directly related to the execution-time flexibility of the tasks in question. Such a task structure results in a large number of ways to accomplish a given task, greatly increasing the planner's search space. While this complex search space is not caused by live task modification, properly addressing it is a prerequisite to the use of live task modification.

The Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP) [13] [22] is the closest match from the scheduling literature to domains involving optional roles: in the MRCPSP, multiple "modes"[5] are available for each task. Each mode has different resource requirements and a different duration. However, the benchmarks studied (e.g. PSPLib [1] [41]) incorporate at most 5 modes per task, while a multi-agent task incorporating optional roles easily can have hundreds, if not thousands, of possible resource assignments. For instance, the *Transport_Panel* task discussed in Section 3.3.1 has two required roles and three pairs of optional roles. With 10 distinct agents available, there are ${}_{10}C_4 *_6 C_2 *_4 C_2 = 18,900$ possible ways to assign agents to roles if we require all roles to be filled. If optional roles may be left empty, a total of 49,876 possible teams exist. To the best of our knowledge, no results have been reported to date with more than 5 modes per task, much less thousands.

While the search space is daunting, there is a structure to domains incorporating optional roles that may work to a planner's advantage. In particular, adding or removing agents from a task will have potentially well-understood effects, even if exact models of each role set and filling combination are not available. For example, adding a scout agent to the *Transport_Panel* task should reduce the number of failures and the expected duration. Similarly, removing an optional transporter agent will decrease the rate of progress and increase the expected duration. While we may not have precise estimates available for the degree to which adding or removing agents will affect the task, the first-order effect on the task's duration and reliability is known. This structure may lead to powerful search heuristics, although their design remains a significant challenge. Issues involved in the design of such heuristics include setup costs (repositioning, warming up, etc.); costs involved in adding or removing agents from a team; whether and how to work with uncertain task durations; reasoning about resource availability; balancing duration, reliability, and resource usage; and whether or how to incorporate knowledge of past decisions.

In addition to the requisite expanded search space, the complexity of the planning problem is directly affected by live task modification due to the inclusion of agent addition and removal tasks, as well as the extensive use of setup tasks. While the depiction in Figure 6 is a good first-order approximation of what occurs when shifting agents between live teams, Figure 7 is more accurate. When moving an agent between teams, it must first be removed from the source task (timespan a), then any setup actions (such as relocating the agent, warming up a tool, etc.) must occur (timespan b), and finally the the agent must be integrated into the team

[5] "Modes" are different approaches to a task requiring different resources and resulting in different durations. A mode corresponds to a specific filling of optional roles by specific agents.

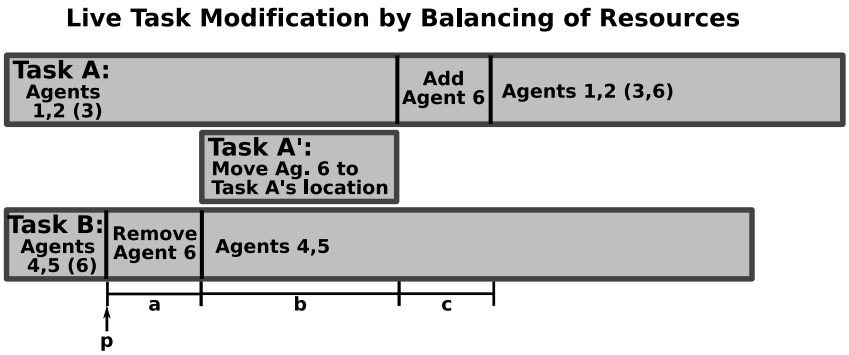

Figure 7: Live task modification will result in a more complex planning problem, as addition, setup, and removal tasks must be considered. In the above extension of the example in Figure 6, the decision to shift Agent 6 from Task A to Task B in order to balance their predicted durations is made at Point p. Timespan a is spent removing Agent 6 from Task B, timespan b is used to relocate Agent 6 to Task A's location, and the final integration of Agent 6 into Task A occupies timespan c. Only after the end of c does Task A realize any benefit.

performing the destination task (timespan c).

When determining whether to modify a team or teams, the planner must take into account these additional tasks, as they may reduce the benefit of the modification, especially since progress often will be impeded during the addition and removal phases. Once the modification has been selected, the planner must manage these additional tasks. This can become much more complex than depicted in Figure 7, as the planner may wish to add or remove multiple agents at a given time, each requiring one or more setup actions. The capabilities of the executive must be modeled both to provide predicted durations and to inform the planner as to the interactions between addition (or removal) tasks. For instance, with some tasks it may be possible to remove many agents at once, while with others agents may be constrained to be added or removed serially.

If constraints exist regarding when the modification can be performed, the problem becomes even more difficult, as the planner must now determine when the team will be in a state in which agents may be added or removed. For instance, in the transport panel task, additional transporters may be added only when the transport subteam is able to shift the existing agents around the perimeter of the panel. At times, this may not be possible due to loose soil, obstacles, or other constraints. Determining whether the team currently is in a state from which it is modifiable is far from simple, but is primarily the task of the executive. However, predicting when a modification window will occur (or is likely to occur) sufficiently far in advance to allow the planner to make use of it, lies within the planner's purview. It may be possible to combine such prediction with the duration prediction models by expanding the models' state space to cover the relevant aspects.

Finally, the difficulty of scheduling a modification increases even further if the setup actions for the addition are dependent on an evolving component of the existing team's state. For instance, an agent joining a transport panel task already in progress must rendezvous with the moving team. The planner must determine whether the current team should continue moving while the new agent attempts a "flying" rendezvous, or whether the team should stop and wait for the agent. While performing the full path planning necessary to completely evaluate the available options likely will be too computationally expensive, the planner must at least approximate the relative costs in order to determine the value of such a modification, as well as the proper approach.

3.3.4 Task Modification and the Executive

Implementing the dynamic addition and removal of agents to and from a team within the lower layers of the system also is far from simple. Although the details will be highly task-specific, the difficulty of modifying a task will vary depending on the characteristics of the role and four characteristics of the modification:

1. Does the modification require action on the part of other team members?
2. Does the modification require close coordination between one or more team members?
3. Are there constraints on when the modification may occur?
4. Are any setup actions dependent on an evolving portion of the existing team's state?

The difficulty of the modification is correlated with the number of affirmative responses to these questions. If no action is required by the remainder of the team during the filling or vacating of a role, the implementation likely will be trivial. For instance, the scout roles of the transport panel task described in Section 3.3.1 may be filled or vacated with no involvement of the agents filling the transporter or tow roles.

However, if close coordination is required, the difficulty of the modification increases dramatically. Consider the transporter roles of the transport panel task. If an additional transporter is added, the current team must rendezvous with the new agent, then stop to incorporate the new agent. This not only requires the existing team members to modify their execution, it delays the entire task, negating some of the benefit garnered by expanding the team.

If constraints exist regarding when the modification can be executed, the executive must determine when the team is in a state in which a modification may occur, in order to provide the planner with the data it requires. This additional monitoring may be somewhat complex. For instance, determining whether the transporters of a transport panel task are able to shift around the perimeter of the panel depends on soil conditions, local obstacles, and the capabilities of the transporter agents. Determining when this is the case is not a trivial problem.

Finally, the capabilities required of the executive expand further if the setup tasks for a modification are dependent on an evolving state. For instance, if the planner determines that rendezvousing with a moving team is necessary, the team and the new agent will need to continuously replan and negotiate in order to determine and reach a rendezvous point, potentially while obeying any constraints on where or when the modification may be executed. Capabilities such as this will make the efficient implementation of live task modification for the task at the executive layer significantly more complex.

3.4 Summary

Proactive replanning seeks to address problems and inefficiencies before they occur, in order to achieve shorter executed makespans than otherwise possible. We propose to achieve proactive replanning through two novel additions to the planning / execution field: duration prediction and live task modification. By updating task duration estimates throughout execution and modifying active teams, the planning / execution system is able to foresee and act to avoid, or at least ameliorate, scheduling problems before they arise. In the next section, we will discuss our experimental scenario, implementation, and initial results.

4 Results To Date

Building on ASPEN and CASPER, we have implemented and evaluated a proof of concept planning and execution system to evaluate 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 makespans, although the effects of live task modification are much more dramatic. When combined, they reduce makespans further than either could alone, yielding executed schedules up to 31.8% shorter than was achieved with a baseline implementation. We discuss here the current scenario, architecture, implementation, and experimental results.

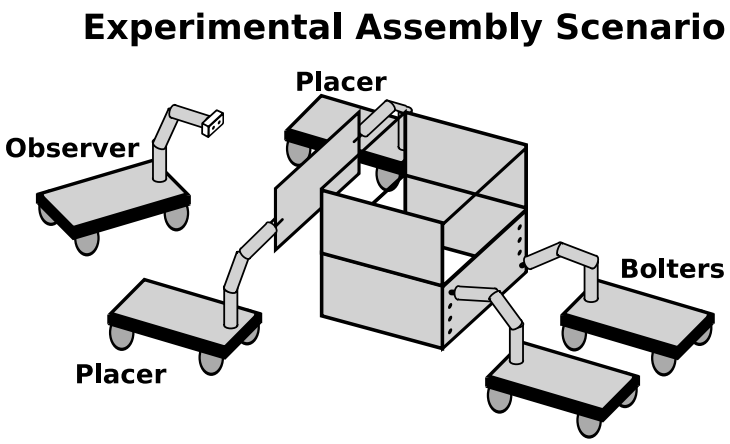

Figure 8: The notional structure being assembled in our experiments. Agents working on adjacent sides may interfere with each other, leading to the temporal constraints detailed in Figure 9.

4.1 Scenario

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, which 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 (see Table 1). Our initial implementation 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 (*Add_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 *Add_Hangers*, *Place_Panel*, and *Bolt_Panel* tasks (see Figure 9). There are eight *Assemble_Side* tasks in the scenario, representing the construction of a two panel-high, four-sided structure (see Figure 8). To avoid interference between teams, only opposite sides of the structure may be assembled simultaneously, leading to the high-level ordering depicted in Figure 9. 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.

Before discussing the details of *Place_Panel* and *Bolt_Panel*, we introduce the way that tasks incorporating optional roles are structured. Within the planner, each such task decomposes at two levels: the “role set” and “role filling” (see Figures 10 and 11). The role set decomposition determines which roles will be filled; for example, how many bolters will be assigned to a *Bolt_Panel* task. Since all agents are equally capable, the choice of role set determines the duration of the task (subject to the stochasticity of the execution model). After choosing a role set, the specific agents that will perform the task must be chosen - this is the “role filling”. The planner may compare the different available role fillings when resolving a resource conflict.

In a domain in which agent capability is homogeneous (that is, if an agent can perform a task, it performs the task just as well as any other agent capable of doing so), this two-level decomposition provides a useful abstraction. The choice of role set determines the task’s characteristics with respect to remaining duration (which is also a function of task state), reliability, and the abstract number of agents which will be required. This allows the planner to reason about temporal constraints without considering the full breadth of possible decompositions. For instance, if the planner is attempting to resolve a conflict by reducing a task’s duration, it need only examine the role sets and the current task state to determine if this is a feasible approach to the repair. After selecting a role set or sets which fit the duration constraints at hand, the planner can then consider which specific resources are available. This staged

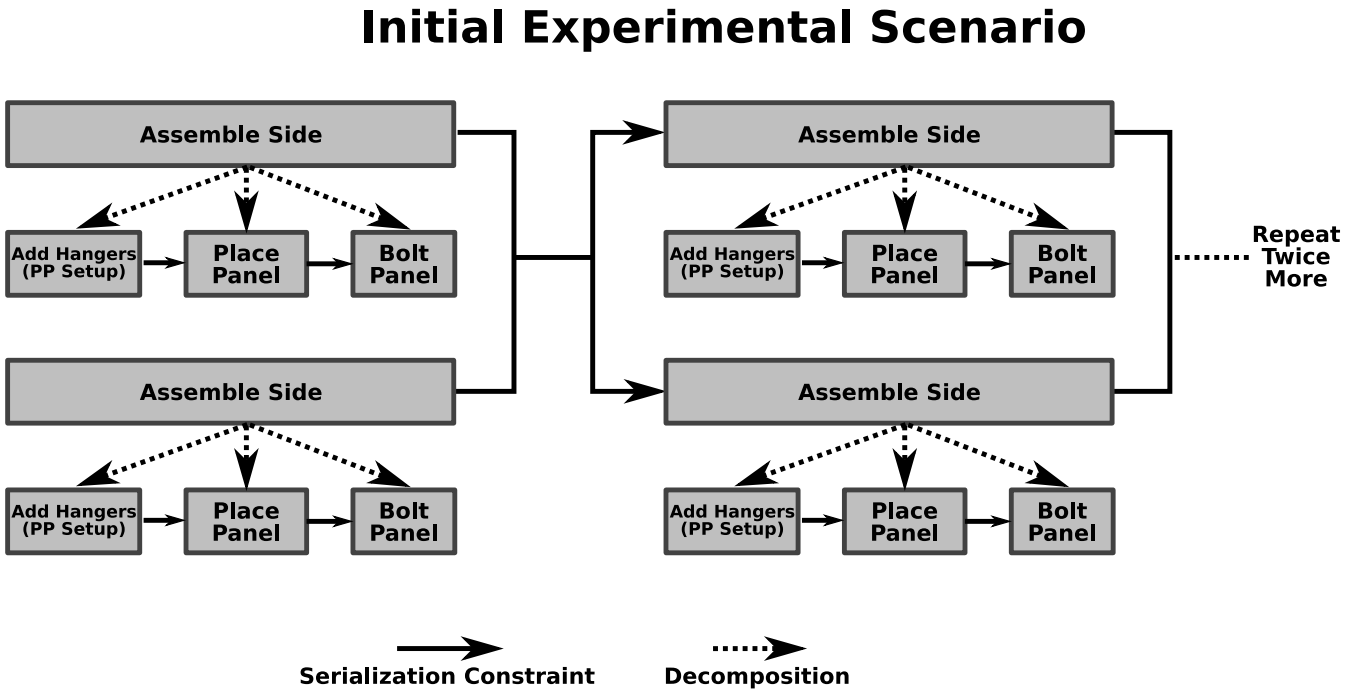

Figure 9: The scenario used in our experiments. A total of eight *Assemble_Side* tasks are included, serialized by pair, since teams cannot work on adjacent sides of the structure without risking interference. The *Add_Hangers* task is a single-agent, fixed-length setup task for *Place_Panel*.

decomposition divides the problem of determining how to accomplish a task into two more manageable pieces: selecting the task characteristics, and selecting the specific resources to assign to the task.

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, *Add_Hangers*. *Add_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 panel must travel a distance of 2 meters during the placing process, which will nominally take 40 seconds. However, failures may occur at any time during the course of a placement operation. These can be quite costly, as the placers must return to their initial position before attempting the placement once again, incurring a penalty of $2t$ seconds if the failure occurs t seconds into the attempt. Such a failure occurs on a given timestep with a probability $P \in [0.01, 0.05]$. P initially is 0.01, but increases by 0.01 with each failure, to a maximum of 0.05. This reflects our experience in the Trestle project [56]: errors during a given task instance are rarely independent. However, *Place_Panel* includes two optional observer roles, which serve to lessen the likelihood of failure. The observer roles represent agents not directly involved in the manipulation, but tasked with observing the placement and providing feedback to the manipulation agents. The likelihood of an error drops to $0.01 - 0.0049 * num\_observers$ when one or more observers are present, regardless of previous failures. With two required and two optional role slots, and four available agents, there are a total of 24 possible role fillings (see Figure 10).

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. There are initially 40 bolts to attach, and a single bolter may do so at a rate of 0.5 bolts per second. Because multiple agents working in a constrained space will interfere with one another, adding a second bolter increases this rate to only 0.95 bolts/second, while filling the final bolter role allows the team to reach a speed of 1.3 bolts/second. However, adding additional agents does add some risk to the team: there is an independent $P = 0.05$ chance

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.
Add_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.

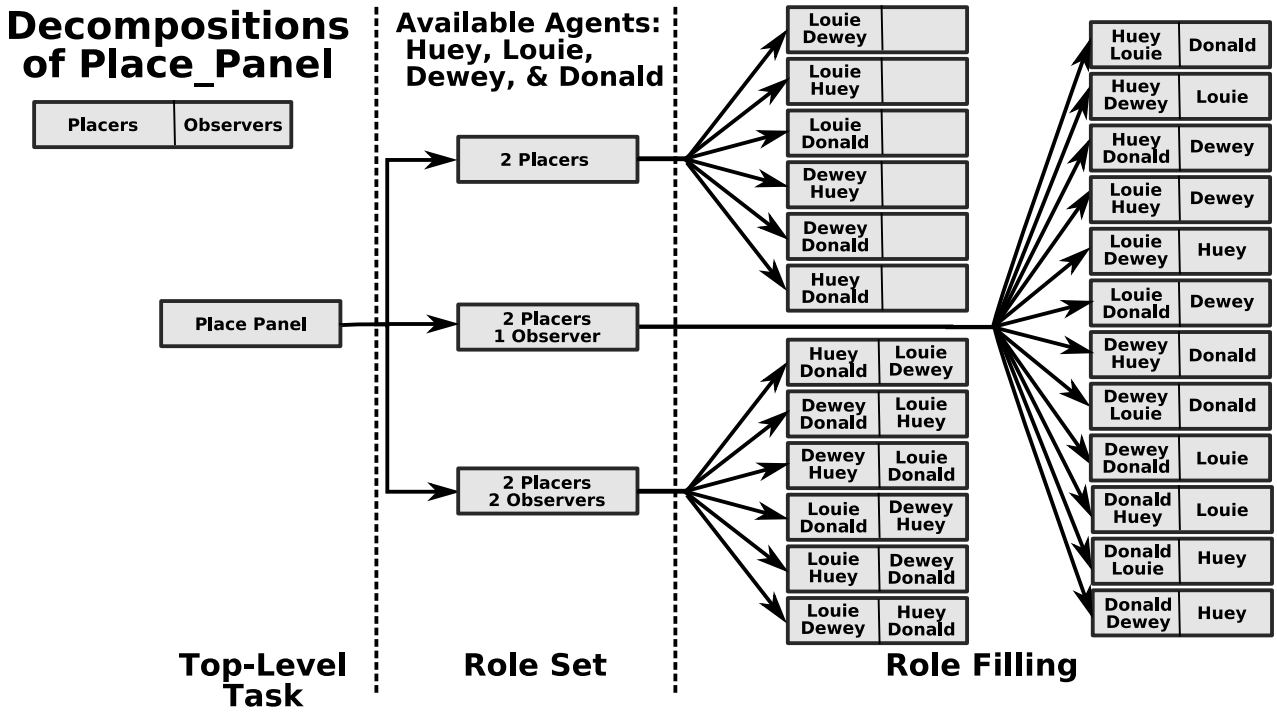

Figure 10: The *Place_Panel* task is responsible for precisely hanging a panel from an existing structure. As constituted in our experimental four-agent scenario, there are 24 possible ways to perform the task.

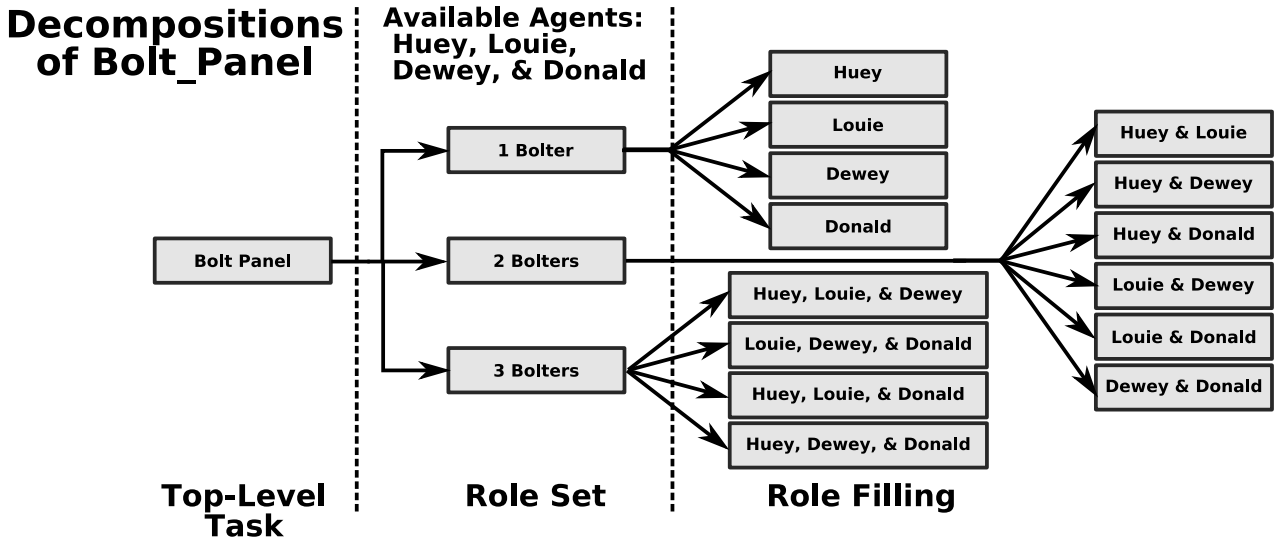

Figure 11: The *Bolt_Panel* task fastens an already-placed panel to the structure. With four agents, there are the above 14 ways to perform the task.

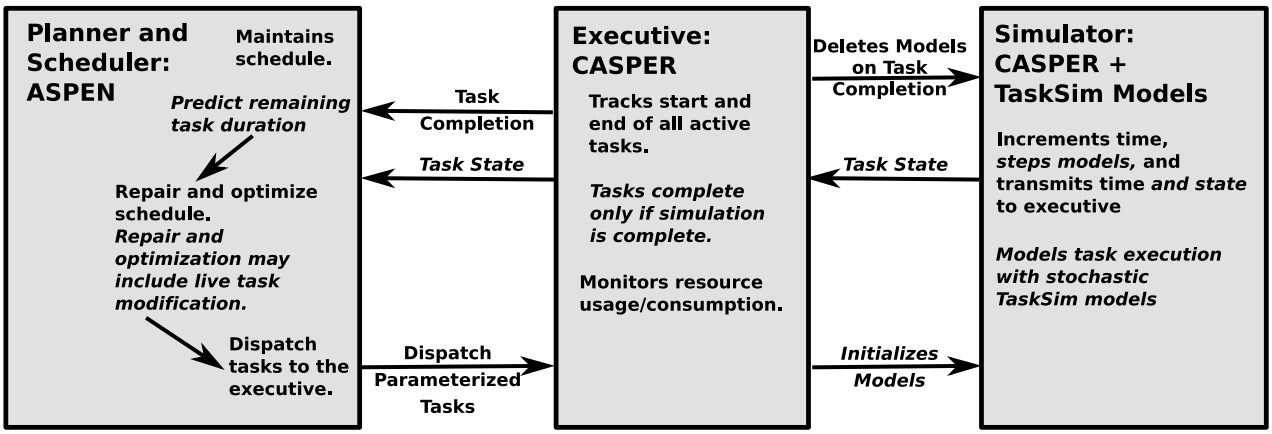

Figure 12: An overview of the planning, execution, and simulation system. Components in italics were either created or heavily modified from ASPEN/CASPER for this work.

that any given agent will fail on a given timestep. If any agent fails, the entire team is forced to recover, consuming a number of seconds drawn from a normal distribution with mean 10 and standard deviation of 1. With four available agents, one required, and two optional roles, there are 14 possible agent assignments (see Figure 11).

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.

4.2 Architecture

The overall architecture of the system is a classic three-tiered approach [10] [56], consisting of planning, executive, and behavioral layers (Figure 12). The planner is responsible for building, maintaining, and optimizing a viable schedule, subject to the temporal and resource constraints imposed by the domain and scenario. As execution proceeds, the planner dispatches tasks to the executive as their start times arrive and repairs the schedule in response to state information provided by the executive. The executive is relatively simple, effectively acting as an interface between the planner and the simulator. As tasks arrive in the executive, it informs the simulator of each task's start time. The simulator then stochastically simulates the execution of tasks, with the initial values of the simulation provided by the planner via the executive (e.g. the number of agents assigned to each role). In addition, the simulator provides task state to the executive, some of which is forwarded to the planner for use in task duration prediction.

4.2.1 Planner

We are making use of the ASPEN planner [15], which has been generously provided to us by the Jet Propulsion Laboratory, 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. 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. We make extensive use of the parameter constraint network when performing duration prediction (Section 4.3.2). We selected ASPEN because of its repair-based paradigm, support for durative actions, and most-commitment approach to variables (they are grounded as soon as possible), all of which are either required or ease the implementation of proactive replanning. We also considered and rejected the EUROPA [31] and IxTeT [42] planners. EUROPA appears to share much of ASPEN's functionality; in the absence of a clear advantage, we chose ASPEN due to our previous contacts with members

1. *Step the executive and simulator one timestep*
   (a) *Process any newly-dispatched tasks*
   (b) *Increment all executing models in the simulator*
   (c) *Update the planner with task state and completion status from the models*
2. *Shift left: shift all tasks as early as possible without violating temporal or resource constraints.*
3. *Shift right: resolve resource conflicts by moving non-executing tasks into the future the minimum distance that eliminates the conflict.*
4. *Optimize: Perform the following 20 times:*
   (a) *Choose a metric to optimize (in our case, this is always makespan)*
   (b) *Choose and apply an optimization method. These methods are hard-coded, and take different approaches, such as adding, moving, or removing activities. In our case, we always use our task modification optimization method, which is detailed in Figure 17.*
      - *Within each optimization method, a variety of "choice points" may be reached. At each choice point, user-specified heuristics are applied to choose an option or narrow the set of choices.*
5. *Repair: Repeat the following until the schedule is conflict-free:*
   (a) *Choose a conflict to resolve*
   (b) *Choose and apply a repair method. Again, these are hard-coded repair methods, each of which is applicable to a different set of conflicts.*
      - *We primarily make use of existing ASPEN repair methods, although we have added our own task modification repair method, detailed in Figure 16.*
      - *As with optimization methods, each repair method contains a series of choice points at which heuristics are applied. Our decomposition heuristic is discussed in Figure 16.*

Figure 13: The flow of execution through the executive, simulator, and planner (ASPEN) during an experimental run.

of the ASPEN team. IxTeT generates plans with unbound variables. While these provide execution-time flexibility, they would make duration prediction significantly more difficult, leading is to select ASPEN over IxTeT.

During execution, the planner dispatches tasks to the executive as their start times arrive, while receiving updates to the state of executing tasks. It then replans to resolve any conflicts that arise during execution. Conflicts may be either violations of resource constraints (e.g. multiple tasks requiring the same agent at the same time) or temporal constraints (e.g. a *Place_Panel* must complete prior to the start of its associated *Bolt_Panel*.). For our initial experiments, we do not execute in real time, allowing the planning layer as much time as needed to resolve conflicts and optimize the schedule. The details of the execution process are laid out in Figure 13. Note that the simulator is not operating in real-time mode; instead, simulated time progresses only once both optimization and repair have been performed at each timestep. One of the goals of the thesis work is to develop algorithms that take advantage of the opportunities provided by duration prediction and live task modification while remaining computationally and spatially efficient enough to be useful in a real time system.

We have extended ASPEN in a variety of ways, including providing a number of domain-specific heuristics (see Sections 4.3.2 and 4.3.3) and adding new repair and optimization methods, that utilize the decomposition structure inherent in the optional role tasks of our domain. These extensions are discussed in detail in Section 4.3.4.

4.2.2 Executive

The executive is largely composed of the CASPER [14] [25] 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. However, originally the duration of tasks had to be fixed when they began, and state variables could change only at the beginning and end of tasks. 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. The original CASPER simulator has no execution model, and tasks always completed exactly when scheduled to do so[6]. A wide variety of minor other extensions and modifications were made to support our simulator and the properties of our domain.

4.2.3 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, and which are discussed in Section 4.3.1 and depicted in Figure 14. 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 in sand are. One aspect of the thesis work will be to dynamically adapt our duration predictions to compensate for discrepancies between the abstract models and the realities of execution.

4.3 Implementation

4.3.1 Task Modeling with TaskSim

We have implemented the "TaskSim" library, an extended version of Augmented Transition Networks (ATNs) [67] [68], to model our tasks within the simulator and to build and query the planner's task duration model. The library supports both the execution of a single run through the task model and the construction of the aggregate task duration prediction models used by the planner. This structure allows the centralization of all code related to task modeling. Both the simulator and the planner link against the same library.

We represent tasks with ATN-like constructs. ATNs are finite state machines in which arcs have associated tests and effects. An arc is followed only if the test evaluates to true, at which point the effects are applied. By convention, in our models exactly one edge's test condition(s) must evaluate to true in any given state and time. Effects may modify the value of variables or invoke a submachine. This enables recursion, as well as making models significantly more modular. Our implementation supports the standard ATN semantics and extends ATNs by supporting random variables from a number of distributions (to date, uniform and normal), which are reevaluated at every timestep. This extension to the ATN formalism allows us to introduce stochastic elements into our model and represent, to an extent, the uncertainty inherent in execution.

The models for *Place_Panel* and *Bolt_Panel* make extensive use of our randomization extension, but do not contain any recursive effects. We model durative actions by incrementing a global time variable as a side effect of each transition.

We utilize these models in two ways: simulation and prediction. They are used directly by our simulator to drive each task instance independently. That is, we instantiate a separate model each time a task is started – there is no correlation between task instances. At every time step, the simulator evaluates the test conditions associated with each outgoing arc from the current state. It then examines the effects of the applicable arc, and determines if following it will advance the model's time variable into the future.

[6]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.

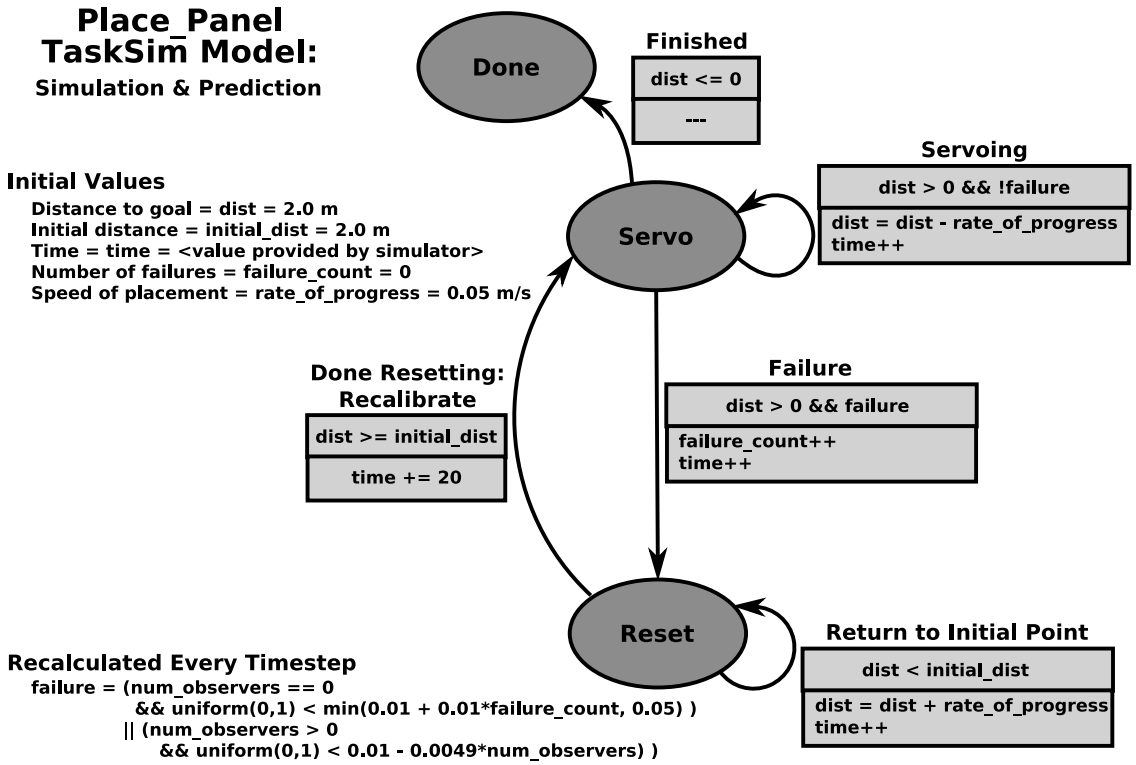

Figure 14: The TaskSim model for *Place_Panel*. This is used by the simulator to dynamically update the variables in the planner's top-level task definitions, such as *dist* (Line 5). Similarly, the planner parameterizes the TaskSim model by setting variables, such as *num_observers*.

If so, no arcs are traversed, and the simulator moves onto either the next model or the next timestep. If the arc does not advance the model past the current execution time, the arc is followed and the side effects are applied. If the model has reached a sink state, its final state and notification of its completion are passed to the executive. If state variables of interest to the executive or planner are modified (regardless of task completion), an update to the executive is triggered. The executive passes the modified state along to the planner, which causes a repropagation through the planner's parameter constraint network, finally resulting in a re-evaluation of our task duration predictions, and potentially plan repair or optimization.

4.3.2 Duration Prediction

The planner-level duration-prediction algorithms make use of the models in a number of ways, albeit in an indirect fashion. State information about each executing task is provided to the planner by the executive, and is passed into a predictive function to estimate the remaining duration of the task, given the current agent assignments and state. The stochastic nature of some models and the effects optional roles may have on execution make it extremely difficult to develop closed-form equations for task duration, even for relatively simple tasks. As a result, for our initial implementation we have chosen to repeatedly simulate each model from every possible state to estimate actual task duration; this is the naive option discussed in Section 3.2.2.

When performing such simulations, one must determine how many iterations to perform for each state before terminating the simulation. Without available ground truth, we are unable to determine a model's accuracy. Instead, we focus on the preciseness (variability) of the estimate. Since each simulation run is independent, we have a series of independent identically distributed (I.I.D.) samples, and estimate the task duration as the mean of the samples. Storing only the mean also serves to keep the problem spatially tractable. After each run, we calculate the 95% confidence interval of the corpus of observed data, 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 and Kelton ([43]), 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 iterations, simulation is terminated. Law and Kelton ([43], 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 [9] [54].

We have built prediction tables mapping the role set / task state space to an estimate of the mean of the underlying duration distribution for all five tasks discussed in Table 1, although only the tables for *Add_Hangers*, *Place_Panel*, and *Bolt_Panel* are used in our experiments. The results of these simulations are presented in Table 2. Of particular note is the *Avg Time for One Estimate* column: this is how long it would take to form an estimate for a single point in the role set / task state space, given no prior information. If duration prediction were to be performed online, such an estimate would have to be formed for every prediction. Note that the time varies significantly, depending on the task, but never decreases below 0.5 seconds (*Place_Hangers* is unique in that it incorporates no stochastic elements). During the experiments that we will report on in Section 4.4, we determined that between 4 and 14 task duration predictions were made per executing task per timestep. Thus, in order to perform duration prediction in real time, a single estimate would need to be made in less than $\frac{0.07}{number\_of\_executing\_tasks}$ seconds, even if the remainder of the planning process consumed no time. This reinforces our belief that we must form our duration distribution estimates offline and store them in an easily accessible fashion.

One potentially serious drawback of this approach is jitter: as the task state changes, our estimate of the task's remaining duration will change slightly at every time step, even when a failure has not occurred. This jitter is due to the structure of our underlying model: we determine our predicted value for each point in the task's state space *a priori* by a series of simulations. There is no guarantee that, if we move from state M to state N in one timestep, the predicted remaining durations from M and N will differ by precisely one timestep. As a result, the task duration will fluctuate slightly at every timestep. This fluctuation may cause conflicts in the schedule, especially if tasks are scheduled very close together, resulting in excessive replanning. For the moment, we are maintaining a 5-second deadband buffer around the end of each task in order to reduce the effects of jitter. Jitter is a well-understood phenomenon

Task	Number of Role Sets	Role Sets x Task States[a]	Runs to Characterize Entire Task[b]	Time to Characterize Entire Task (s)	Avg Runs for One Estimate[c]	Avg Time For a Single Run (s)	Avg Time for One Estimate (s)[d]
Transport_Panel	27	8997	157,964	225.21	478.38	0.0014	0.67
Rotate_Panel	2	1274	2292	5.017	338.32	0.0022	0.74
Place_Hangers	1	21	10	0.0062	10.00	0.00054	0.0054
Place_Panel	3	123	20,552	23.98	1696.20	0.0012	2.04
Bolt_Panel [e]	3	123	642,895	10921.2	164,238.87	0.017	2792.06

[a] This is a count of the number of states in the space of role sets by task states. Note that this does not take into account the assignment of individual agents, only which roles will be filled.

[b] The 'Entire Task' consists of all possible combinations of fillings/vacancies of optional roles and all possible task states. Since a single run provides data for more than just the initial state, this is smaller than the size of the state space multiplied by the runs needed to characterize a single state.

[c] 'One Estimate' is a single point in the role set / task state space. This column is the average number of simulation runs needed to build an estimate of the mean of the duration distribution for one point in the space, with no prior data.

[d] This is simply the product of the average number of runs needed to converge one estimate and the average time for a single simulator run.

[e] *Bolt_Panel* incorporates a greater degree of stochasticity than the other tasks, resulting in a broader underlying distribution. This makes it much more difficult to achieve a desired relative precision. Although the target relative precision could be relaxed, there will always be a model that tests its bounds.

Table 2: TaskSim model simulation results. The experiment was performed on a lightly-loaded Pentium-4 3 GHz with 1GB of RAM.

in fields such as control systems; as part of the thesis work we will implement a more principled approach to the problem.

4.3.3 Live Task Modification

Our current implementation of live task modification at the planning level supports instantaneous addition and removal of agents, but does not yet address the agent addition, removal, and setup tasks that will be present in the thesis work. Once the new role set and role filling tasks have been determined, the previous set and filling tasks are shrunk to span only the time they have already spent executing (Figure 15b) and the parent-child link between the top-level task and the old role set task is removed. Maintaining the old tasks on the schedule is necessary in order to keep depletable resource timelines in the proper state, since resource reservations are made by the role filling task. For instance, if the *Place_Panel* task consumed one unit of power per second from each agent and we simply removed the old role set and filling tasks, the planner would overestimate the amount of power remaining to the Huey and Louie agents. After shrinking the old set and filling tasks, the new ones are added to the schedule, the top-level task's duration is constrained to be the sum of its former children's and new children's durations, and the new tasks are dispatched to the executive (Figure 15c).

While the executive and simulator do not model any sort of durative actions associated with adding or removing agents to a team, the executive has been extended to detect when a task modification has occurred. It does so by detecting when a new role set task is dispatched whose parent task is already executing. In response, the executive terminates the existing role set and filling tasks (the planner does not send termination commands when shrinking them in its schedule (Figure 15b)). The executive then updates the parameters of the simulator's task model, which is tied to the top level task, to reflect the new role set. This allows the simulator to maintain a single model throughout the course of a task, regardless of the number of modifications that take place. This extension was necessary since originally ASPEN was allowed only to dispatch tasks to the executive, and could not modify executing tasks in any way, much like most other existing planning / execution systems.

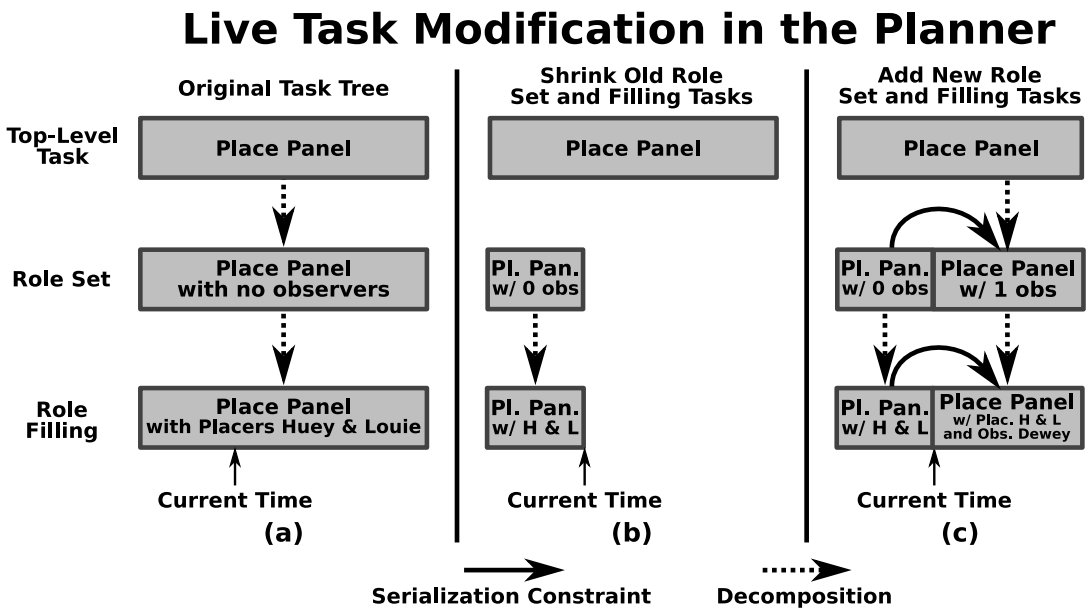

Figure 15: When modifying an executing task (a) in the absence of addition, removal, or setup actions, the planner shrinks the existing role set and filling tasks to span the time already spent executing (b), then adds the new subtasks and constrains the top-level task to contain all of its former and new children (c).

4.3.4 Planning and Live Task Modification

Determining when and how to modify a live task is the core of live task modification, and is the responsibility of the planner. Live task modification has been incorporated into both the repair and optimization stages (Steps 4 and 5 in Figure 13) of the planner's execution flow. Repair is performed whenever agents are over-subscribed or temporal constraints are violated anywhere in the schedule, while optimization is performed at every timestep to ensure that the planner takes advantage of any opportunities that have arisen to reduce the makespan. Live task modification is useful in both situations, but is applied slightly differently.

Repair We implement live task modification within ASPEN's repair algorithm using a combination of a new repair method for resource conflicts and a heuristic that guides the selection of the role set and role filling decompositions. In brief, ASPEN's repair algorithm is (1) choose a conflict, (2) choose a repair method, (3) apply the repair method, (4) repeat until all conflicts are resolved or a time limit has been reached.

A repair method is a predetermined "recipe" for resolving conflicts of a particular type. The repair method may include (many) branches and stochastic choices, but the "recipes" themselves do not vary. For instance, if a *Bolt_Panel* is scheduled to begin before its associated *Place_Panel* finishes, either the "move" or "task modification" repair method could be applied. The move repair method would select one of the tasks and attempt to move it within the schedule to resolve the conflict. If the task modification method were selected, the planner would attempt to add resources to one of the tasks until its expected duration was short enough to resolve the conflict. A stable of generic repair methods is included in ASPEN; we have extended this set by adding our task modification method. See Section 4.2 and Figure 13 for further details on the general repair process.

Each repair method contains one or more "choice points". These points correspond to decisions that may be guided by user-defined heuristics, such as which task to move, what duration to use, or which task decomposition to use. ASPEN allows a set of weighted heuristics to be specified for each choice point, and randomly selects one according to the specified weights whenever a choice point is reached. We have developed a decomposition heuristic that is used to select role set and filling tasks during a task

1. *Find all tasks which are contributing to the current conflict and have not yet completed execution.*
2. *Pick a random contributing task to modify which hasn't been modified in the last N seconds.*
3. *If no such task exists, reduce N until a task is found or $N = 0$. If $N = 0$, fail.*
4. *For each currently-executing child (e.g. the role set or role filling task):*
   (a) *Remove all constraints between the top-level task and the child.*
   (b) *Set the child's end time to the current time.*
5. *Choice Point: Choose a new role set task, using our decomposition heuristic:*
   (a) *Weight each possible decomposition according to the inverse of its predicted remaining duration. Double the weight of decompositions which are short enough to resolve the current conflict.*
   (b) *If no decomposition was short enough to resolve the conflict, reweight all options according to their predicted remaining duration. The implicit assumption is that instantaneous resource usage is inversely correlated with duration: if we can't resolve the conflict temporally, perhaps we can select nonconflicting resources.*
   (c) *Stochastically select a decomposition according to the assigned weights.*
6. *Choice Point: Choose a new role filling task, using our decomposition heuristic, which simply chooses a random role filling.*

Figure 16: Our task modification repair method and decomposition heuristic. The heuristic consists of Steps 5a - 5c.

modification (Figure 16, Steps 5a - 5c).

Our repair method (Figure 16) randomly selects a task that is contributing to the conflict and has not completed execution. If the task has not begun execution, its role set and filling tasks are replaced with new ones chosen using our decomposition heuristic (Figure 16, Steps 5-6). If the task is live, the repair method modifies the team performing the task as discussed above in Section 4.3.3 and Figure 15, again using new role set and filling tasks chosen by our decomposition heuristic.

If role sets are available that result in a task duration short enough to resolve the conflict, our decomposition heuristic increases the weight on those sets. If it is not possible to shrink the task's duration sufficiently, we instead prefer role sets requiring fewer agents. While allocating fewer agents to each task will increase the tasks' duration, it may resolve the conflict by allocating disjoint sets of agents to the conflicting tasks. After weighting the role set options, our decomposition heuristic randomly selects an option according to the weights. The role filling task (and hence the agents assigned to the new task) are then selected entirely at random. Further details of the heuristic may be found in Figure 16, Steps 5-6.

In resource-scarce environments, this random assignment of agents can lead to a "domino" effect, in which we modify Task A to solve a resource conflict between it and Task B, only to cause a conflict between Tasks A and C by assigning an agent to A that is already committed to C. When repairing the new conflict between Tasks A and C, we may recreate the original conflict, opening the door to an endless cycle of repairs. However, due to the randomness of the heuristic, the planner eventually will build a valid schedule as we settle on a legal allocation of resources. For completeness, we eventually need to attempt all possible assignments, as the current implementation does, but we should prefer those assignments which do not introduce additional conflicts. This extension will be a minor component of the thesis work.

Optimization While repair is necessary to ensure an executable schedule, a live task modification-enabled repair algorithm alone will not result in a reduced makespan. In order to take advantage of the efficiencies offered by live task modification, we perform a series of iterative optimizations at every timestep. ASPEN's iterative optimization algorithm is quite similar to its approach to repair: (1) Choose a metric to optimize, (2) Choose an optimization method, (3) Apply the method, (4) Repeat N times, or until the metric

Task	Agents	Predicted Duration (s)[a]	Progress Per Agent Per Second[b]
Rotate_Panel	2	538	0.00093
	3	461	0.00072
Place_Hangers	1	20	0.05
Place_Panel	2	342	0.0015
	3	52	0.0064
	4	40	0.0063
Bolt_Panel	1	104	0.0096
	2	85	0.0059
	3	77	0.0043

[a] 'Predicted duration' is the duration predicted for this role set from the initial state.
[b] The fraction of the task which is performed by each agent at each timestep: $1/(number\_agents * predicted\_time)$.

Table 3: The durations predicted for each task in the experimental scenario from the task's initial state, given the assigned number of agents. Note that the "usefulness" of additional agents, as measured by the fraction of the task performed by each agent per timestep, decreases when additional agents are added, except when adding the first observer to *Place_Panel*. The 27 *Transport_Panel* entries have been omitted for brevity.

is maximized.

The only optimization metric in our experimental domain is makespan: all iterative optimizations are performed with the intent of reducing the schedule's makespan.

Optimization methods are nearly identical in structure to repair methods, consisting of a predetermined sequence of steps and "choice points" that attempt to optimize some aspect of the schedule. While we use a mix of generic and custom repair methods during the repair process, during iterative optimization we use solely the makespan optimization method detailed in Figure 17, which uses any agents that are free at the current time to reduce the duration of a task on the current critical path. Thus, every iterative optimization (Figure 13, Step 4) consists of an application of our makespan optimization method and associated heuristics.

The goal of our makespan optimization method is to ensure that agents do not needlessly stand idle, and does so at every timestep by assigning any free agents to tasks on the current critical path. The method first selects a set of R (initially $R = 1$) free agents and a task on the critical path, using our Agent Selection (Figure 17, Steps 1a-1c) and Task Selection (Figure 17, Steps 2a-2b) heuristics, respectively.

Our Agent Selection heuristic randomly selects a set S of R agents from those agents that are not committed to executing a task at the current time. If no set of R free agents exists, the optimization method fails.

Our Task Selection heuristic first searches for tasks on the critical path that are not being executed and can be performed by the set of agents S. If no such tasks exist, it then attempts to find tasks on the critical path that are currently being executed, and to which the agents S may be added. We prefer non-executing tasks, as in this domain adding additional agents to a task generally results in ever-decreasing incremental gains. Table 3 details the utility of adding additional agents to each task in the experimental domain. In all but one case, adding an agent results in a lower degree of progress per agent per second – that is, the agents interfere with each other to some extent. In the absence of temporal constraints or other bottlenecks, this makes parallelization more worthwhile than concentrating large amounts of agents on a single task. Extending the Task Selection heuristic to reason about the relative utility of assigning an agent to different tasks will be a minor portion of the thesis work.

If no tasks have been found, we increment R and select a new set of agents. If we have found a task, we modify the task to use the new set of agents (Figure 17, Steps 5a-5b). If the task is not yet executing, its role set and filling tasks are simply replaced, while if we are modifying a live task we take the approach previously discussed in Section 4.3.3 and Figure 15.

For $R = 1$ to $NumberOfAgents$

1. *Choice Point: Agent Selection Heuristic:* Select a set S of R agents:
   (a) Randomly select R agents from the set of agents free at the current time.
   (b) If no such set S is available, fail.
2. *Choice Point: Task Selection Heuristic:* Select a task to add the available agents to:
   (a) Find all tasks on the critical path[a] that are not being executed, and can be executed by S.
   (b) If no such tasks exist, find all tasks on the critical path which are being executed, and have enough unfilled optional roles to accommodate the addition of all agents in S.
3. If the Task Selection Heuristic did not find any tasks, continue the loop with the next value of R.
4. Otherwise, randomly select one of the tasks it found and exit the loop.

End Loop

5. Modify the task:
   (a) If the task is not yet executing, replace its role set and filling tasks with a role set containing R roles and a role filling corresponding to the members of S, then move it to the current time and begin execution.
   (b) If the task is already executing, add the S agents to the task, as discussed in Section 4.3.3 and Figure 15.
6. Shift left: move all tasks as close to the current time as possible, while respecting temporal and agent/resource constraints. This will fill any hole created by the pulling forward or shrinking of a task on the critical path.

[a] See Figure 18 for the details of how we determine the critical path of the current schedule.

Figure 17: Our makespan optimization method and associated heuristics.

This approach is susceptible to a problem similar to the "domino" effect described above: the "flip-flop" effect. Suppose there are only two identical tasks, A and B, currently executing, with one required and one optional role each, and three available agents. Two agents are permanently assigned, one to A and one to B, while one agent is free to fill the optional role of either A or B. If it assists Task A, A's duration will shrink, putting B on the critical path. In an attempt to optimize, the optimization algorithm will shift the additional agent from A to B. Unfortunately, this puts A on the critical path. We currently avoid this issue by shifting resources away from an executing task only if doing so results in a shorter predicted makespan.

4.4 Experimental Results

In order to experimentally validate our claims about the effectiveness of duration prediction and live task modification, we conducted a series of 2x2 experiments 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 the duration prediction and live task modification concepts. The thesis work will focus on refining these concepts and improving the algorithms until they are efficient enough to be used in a real-time planning/execution system.

4.4.1 Conditions

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 given role set by the duration prediction algorithm. Although no duration prediction is performed by the planner during execution, if a task overruns its duration is extended in increments of

1. Find the task (or tasks) with the latest ending times, fill a list A with them, and associate a slack value of 0 with each. Let F be the list of final traces through the schedule, initially empty.
2. Iteratively perform the following steps until A is empty:
   - Clear the temporary list of lists A'.
   - For the first task T of each list L in A:
     (a) For each agent used by T, find the preceding task T' which makes use of the same agent. If one exists, push $[T'L]$ onto A', incrementing L's associated slack by $start(T) - end(T')$.
     (b) For each temporal constraint between T or its ancestors and an earlier task T', push $[T'L]$ onto A', incrementing L's associated slack by $start(T) - end(T')$.
     (c) If no preceding tasks via either agents or constraints were found, the trace is complete. Increment L's slack by $start(L) - current_time$ and add to F.
3. The trace in F with the lowest total slack is the critical path.

- This is roughly equivalent to Dijkstra's shortest-path problem, where the set of nodes is not initially known and edge weight corresponds to slack. The above will be reimplemented using Dijkstra's approach.
- Regardless of the search algorithm, this approach is quite inefficient, as we search the entire tree of tasks related to the latest-ending task(s) every time we need to determine the critical path. Some type of caching or repair-based search will be in order, especially in larger domains.

Figure 18: Our approach to determining the critical path of the current schedule.

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 task modification repair and optimization methods discussed in Section 4.3.4 are utilized, but only on 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 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 optimization method detailed in Figure 17, without considering executing tasks.
4. **Repair:** Perform iterative repair until all conflicts are resolved.

Prediction The prediction case adds task duration prediction to the baseline scenario, as detailed in Section 4.3.2. This gives the planner a more accurate estimate of when tasks will complete, and updates at every timestep, allowing the planner to adapt to delays as they occur. 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 resource conflicts and putting idle resources to use with this approach.

Prediction and Live Modification In this case, both duration prediction and live modification are enabled. We hypothesized that this would be the most efficient case with respect to makespan, as the planner is able to foresee opportunities and conflicts while

	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%)
Tasks scheduled:	81.00 (0.00)	81.00 (0.00)	113.68 (4.84)	112.16 (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 optimizations:	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 4: 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.

taking advantage of them through the modification of executing teams. For instance, in this case the planner is able to shift resources from executing tasks not on the critical path to those that are, based on the evolving predictions of remaining task duration.

4.4.2 Results

We performed 50 runs under each of the four conditions outlined above. Statistics are summarized in Table 4. 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. "Tasks scheduled" is the total number of tasks on the final schedule. The minimum is 81: 1 *Build_Structure* task (which simply contains the rest), 8 *Assemble_Side* tasks, 8 *Add_Hangers* tasks, 8 *Add_Hangers* role set tasks, 8 *Add_Hangers* role filling tasks, 8 *Place_Panel* tasks, 8 *Place_Panel* role set tasks, 8 *Place_Panel* role filling tasks, 8 *Bolt_Panel* tasks, 8 *Bolt_Panel* role set tasks, and 8 *Bolt_Panel* role filling tasks. When live task modification is available, an additional pair of tasks (the new role set and filling tasks) will be scheduled every time an executing task is modified, as discussed in Section 4.3.3 and Figure 15. "Average agent usage" is the average across all agents of the fraction of the makespan during which each agent was performing a task. This can be considered a measure of how much "space" was left in the final schedule. "Repair episodes" and "Optimize 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 repair methods 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 optimizations" is a count of the number of optimization attempts that succeeded in optimizing some aspect of the schedule.

4.4.3 Discussion

The most notable aspect of the data in Table 4 is the average makespan, which also is depicted in Figure 19. 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

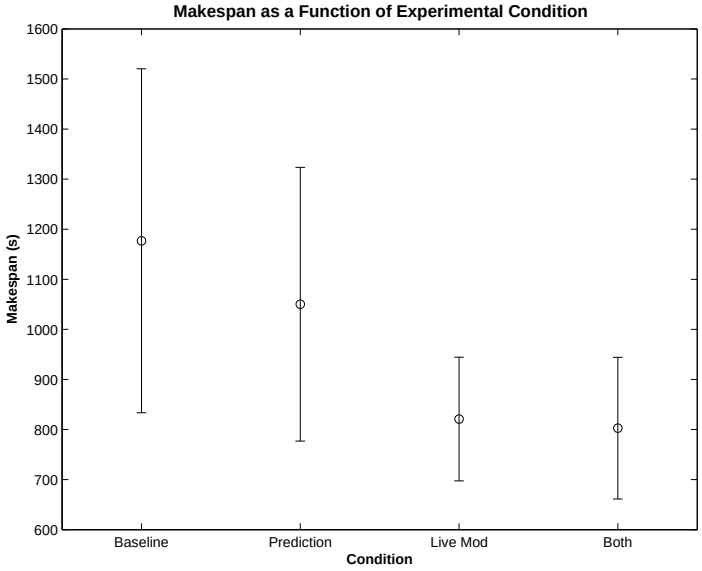

Figure 19: The average makespan and standard deviation for each of the four test conditions.

the presence of setup tasks or significant, sudden over-runs. In the experimental scenario, setup tasks only occur in concert with *Place_Panel* tasks. In addition, the combination of prediction and live task modification performs slightly better than live task modification alone. In domains with more setup tasks, such as those which require setup tasks when adding an agent to a live team, duration prediction will result in larger improvements in the combined case. These gains can be partially attributed to the better utilization of available resources, as detailed in the "Average resource usage" line of Table 4. Note that these are reported as a fraction of the relevant makespan, so they will not necessarily decrease with makespan. Instead, these data can be interpreted as a measure of how "full" the final schedule was.

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: as we discussed previously, 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 tradeoff: 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.

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 reduces the number of planning episodes, to an extent. However, significant work on the minimization of jitter has been performed in fields such as control systems theory. We will implement a more formal solution as a component of the thesis work.

In the current implementation, the iterative optimization procedure is run at every timestep, which is clearly suboptimal, resulting in significant waste of computation time. However, task duration estimates are updated at every 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 a more focused approach will be an aspect of the thesis work.

Finally, the number of successful optimization attempts doubles 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 currently 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.

4.5 Summary

Using ASPEN and CASPER as a basis, we have implemented and evaluated a proof of concept planning and execution system to determine the effect of duration prediction and live task modification on the makespan of an executed schedule. Disregarding planning and optimization time, both approaches reduce the overall makespan, although live task modification has by far the greater effect. When used together they synergize, reducing the makespan further than either could alone, yielding final schedules up to 31.8% shorter than was achieved with neither approach. We believe further incremental gains to be possible, but our focus must now turn to achieving the same gains when working in a real-time environment.

5 Proposed Research

While our experiments to date have demonstrated the theoretical utility of duration prediction and live task modification, significant research remains. Our current approach is not real-time, nor does it support important domain characteristics such as explicit, non-instantaneous addition and removal of agents from live teams. The proposed thesis work falls into four broad categories: duration prediction, live task modification, architecture, and characterization. We will develop duration prediction algorithms capable of producing estimates of sufficient fidelity without unreasonable computational or spatial requirements, and incrementally adapt these estimates in response to observations. The thesis work will extend our current live task modification search heuristics to make use of the structure inherent in tasks with optional roles and to properly handle a wider range of scenarios. We also will optimize our live task modification approach to enable real time execution, and incorporate support for associated agent addition, removal, and setup tasks. Third, we will investigate different ways to structure the tight interaction between executive and proactive planner. Finally, we will formally characterize how different aspects of tasks and domains relate to the efficacy of our approach.

The overarching goal of the thesis is to understand the power and applicability of proactive replanning, specifically duration prediction and live task modification, then demonstrate its effectiveness in real-time settings. We will evaluate the work in both simulated and real world environments.

5.1 Duration Prediction

The problem of duration prediction is one of approximation. Duration prediction algorithms must be capable of mapping each point in the combination of task and state spaces to a predicted duration, or distribution across durations. Since the remaining duration is affected by both the team performing the task and the current state, the domain of the prediction function is extremely large. For instance, the *Transport_Panel* task briefly discussed in Section 3.3.1 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. Inevitably, additional state variables will be needed, further expanding the prediction function's domain.

Given such vast domains, the fundamental problem facing the design of a duration prediction algorithm is twofold: represent the expected duration with sufficient fidelity, while remaining computationally and spatially feasible. In addition, duration prediction must surmount the difficulties normally associated with prediction or modeling functions: mismatches between the abstract model and actual execution. We discuss here how the thesis work will address each of these three points, as well as validate the accuracy of the predictive function.

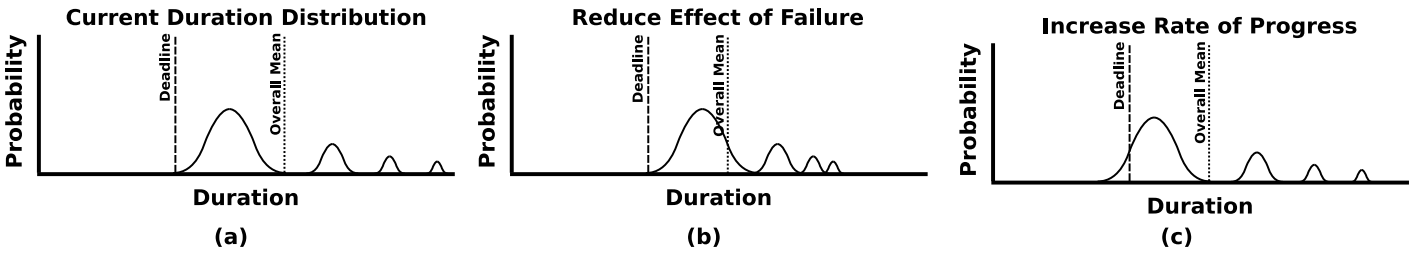

Figure 20: An illustration of the advantages of modeling multimodal durations. (a) is the current team's empirical duration distribution. The team has two unfilled optional roles: one reduces the effect of failures (b), and the other increases the rate of progress when operating normally (c). Note that the means of (b) and (c) are identical, but only (c) provides a non-zero probability of meeting the deadline. This distinction cannot be made using only the overall mean or a unimodal model of duration.

5.1.1 Fidelity

In our work to date, we take a straightforward approach to predicting task duration: offline, we build a table mapping the task/state space to a single estimate of remaining duration. We then predict task duration throughout execution by referencing this table. While this approach is computationally feasible for the tasks in our current limited experimental domains, it provides only a first-order approximation of the task's true behavior. We propose to extend our approach to provide more detailed duration predictions capable of reflecting the true multimodal task duration distributions at any given point in task/state space. In a real-world scenario, these prediction tables could be built in two ways: in the same fashion as we have already done, using simulated models of the real-world tasks and many iterations of simulation, or through the collection of empirical performance data from the repeated execution of the scenario with real-world hardware. We will take a hybrid approach, by building initial estimates from simulation, and updating them as empirical data becomes available. This will be further discussed in Section 5.1.3.

To date, we have modeled the duration of a task as being drawn from a unimodal distribution, and have further simplified this model by considering only the mean of this distribution. However, the underlying distributions are generally multimodal, with each mode reflecting a different number of failures that occur during execution. The differentiation between modes will increase as the delay induced by each failure increases, leading to significant separation between peaks in the duration distribution. If we were to extend the system to provide a duration distribution, rather than a simple mean, the planner would be able to form more realistic plans and make better use of live task modification. For instance, consider a task with a hard deadline, such as that diagrammed in Figure 20a. Assume that the planner has one additional agent available, and can use it to fill one of two optional roles. One reduces the delay associated with a failure (Figure 20b) while the other increases the team's rate of progress (Figure 20c). As we can see, it is possible for the two options to result in identical means, while only one option provides any chance of meeting the deadline.

While increasing the fidelity of the duration predictions clearly is useful, it increases the complexity of the modeling problem. In particular, it is unclear how best to represent multimodal distributions. Options include a discrete distribution, a mixture of parameterized continuous distributions, or a mixture of parameterized discrete distributions. Section 5.1.2 evaluates the relative complexities, as well as other computational issues with duration prediction.

5.1.2 Complexity

The complexity of duration prediction is a classical time/space tradeoff. However, as discussed in Section 4.3.2, the time needed to estimate even the mean of a duration distribution for a single point in the task/state space may be prohibitively large. Since this makes online prediction estimates infeasible in general, we must build them offline in such a way that they may be quickly referenced, yet do not consume an unreasonable amount of space. The root problem is representing a complex multidimensional function in a space-efficient manner that allows rapid calculation. For clarity, let F_{mean} be the function mapping the task/state space to a single-valued

estimate of the remaining duration, while F_{dist} represents a mapping to a distribution across remaining duration.

There are two approaches to reducing the size of our precalcuated prediction tables: reduce the domain of the prediction function, or reduce the size of each estimate. Depending on the task, we have observed a large degree of structure to F_{mean}. This likely will also be the case with F_{dist}; we may be able to take advantage of this structure to collapse portions of the prediction function's domain. A significant amount of work has been done on function approximation; we hope to adapt work from this area to help resolve this problem. For instance, one potentially promising approach is that taken by [48], in which they use kd-trees to condense plateaus in their value function. This may not be directly applicable to our domain, as the time needed to complete the remainder of a task is likely to scale smoothly, not discretely, as a function of our state variables. However, we may be able either to approximate our functions as a series of plateaus or to discover related work more applicable to our problem.

In addition to examining approximation methods that condense the domain of the prediction function, we will investigate approaches to achieving the desired prediction fidelity with compact individual estimates. Given the potentially arbitrary nature of task models, it is unlikely that we will be able to achieve reasonable multimodal fits using a mixture of continuous functions such as the normal, beta, or gamma distributions. We likely will build a library of unimodal discrete distributions to use as parameterized basis functions for each task, then store each estimate as a mixture of such basis functions.

5.1.3 Adaptation

Once the fidelity and complexity components of duration prediction have been addressed, we will turn our attention to addressing the perennial problem of modeling: adapting to mismatches between the model and reality. Since we will be forming our initial estimates based on simulated models, the predictions will at least initially be flawed, and the planner will build plans that either cannot be executed in the scheduled time or that overestimate the time required for a task, resulting in inefficiencies. We will apply machine learning techniques to the incremental adaptation of our duration prediction models, in order to allow the system to efficiently resolve mismatches between the *a priori* estimates and the real world. The simplest approach conceptually would be to incorporate each observed execution trace directly into our duration estimates, but this will require a vast number of training examples to update the entire model. Our precise approach will depend heavily on the final representation of F_{dist}. If we condense or otherwise approximate regions of the function's domain, we may be able to adapt large portions of the domain at once.

5.1.4 Validation

One critical aspect of adaptation is the evaluation of our duration estimates against empirical data – without a way to compare the estimates to ground truth, adaptation cannot occur. The empirical data from the execution of a task will consist of a series of points in the task/state space, each associated with an observed duration. In order to validate our estimates, we will need to determine how closely the observed data match our estimated duration distributions. Our first approach will be to score an estimated distribution by the likelihood that the observed data was drawn from it. To do so, for each point in task/state space we will calculate the average probability that the duration distribution assigned to the corresponding empirical data. The result will be large if the observed data fit the estimated distribution well, and will be small if it does not.

In our work to date, we have used the same underlying task models to form our estimates and drive the simulator, in order to better evaluate the potential effectiveness of duration prediction and live task modification. In the thesis work, we will investigate domains in which the simulator's model differs from the model used to form duration predictions, as well as real world domains, in which the models are mere approximations of reality.

5.1.5 Duration Prediction Summary

The thesis work will develop duration prediction algorithms capable of mapping the task/state space to a (multimodal) distribution estimate of the remaining task duration in real time. In addition, we will apply machine learning techniques to enable the adaptation

of our estimators to observed data as execution proceeds, then validate the results.

5.2 Live Task Modification

The fundamental problem of live task modification is complexity: a vast search space and a multiplicity of tasks that must be managed to make live task modification possible, both of which must be addressed within the computational constraints of a real-time system.

Live task modification will come into its own as the number of possible teams for a given task increases, broadening the planner's options and ability to precisely react to events during execution. The tasks in our current experimental scenario provide only 14 or 24 possible decompositions, but more complex tasks will cause this number to explode. (Recall, the *Transport_Panel* task briefly discussed in Section 3.3.1 may be performed by 49,876 distinct teams if 10 agents are available.) Such drastic increases in the branching factor of this portion of the plan search space will strain our existing search heuristics. We propose to extend these heuristics to take better advantage of the structure inherent in domains involving optional roles and to handle correctly a wider range of scenarios, while optimizing them to provide real-time performance.

In addition, our work to date assumes that modifications may be made to an executing team instantaneously, with no intermediate effects upon tasks receiving or losing agents. The thesis work will remove this approximation by incorporating agent addition, removal, and setup tasks into the live task modification algorithm.

5.2.1 Search

Our current exhaustive approach to searching the space of teams that can accomplish a task will not scale to more complex instances, such as *Transport_Panel*. We propose to take advantage of the structure inherent in tasks involving optional roles to develop more efficient search heuristics. Our initial approach will be to develop a graph-based representation of the possible ways in which a task can be accomplished. Nodes in the graph represent a specific solution to the task (e.g. a specific role set and/or role filling task), and will be interconnected by arcs. Each arc represents a task modification action, and is weighted by the estimated time needed to accomplish the modification. For instance, if it will take 60 seconds for a new agent to rendezvous with the team, and an additional 40 for it to integrate into the team, the edge would have a weight of 100. These weights will be only approximate, as the precise calculation of the durations of these addition, setup, and removal tasks is computationally expensive. Using such a structure, the planner will be able to search outwards from the current team for an appropriate modification action (or a series of actions) that can be accomplished quickly enough to address the problem or opportunity at hand.

In addition to allowing the planner to search large task spaces, such an extension will increase the expressiveness of live task modification. While we can describe many variations on a task through the use of optional roles, the variations must have a common core of required roles. A natural extension is meta-tasks, which provide a number of alternative approaches to a task (e.g. driving vs. flying vs. walking), each of which is made up of a different set of required and optional roles. This can be naturally incorporated into a graph-based representation such as that discussed above.

5.2.2 Optimization

Our current system is far from real-time: in order to evaluate the potential of proactive replanning, we allow an arbitrary amount of time for the planner to repair plan conflicts and optimize the schedule between simulation steps. During our initial experiments, we spend as much as 17% of the final makespan repairing or optimizing the schedule. While improving the efficiency of our task space search heuristics will help, we also propose to reduce both the number of repair/optimization cycles and the time taken by each through improvements to the accuracy and efficiency of our task modification heuristics.

Reduce the Number of Repair Iterations The greatest weakness of our current task modification repair heuristics is their failure to recognize the possible conflicts created by each modification option. During a repair via task modification, our current heuristics consider the expected duration of each option, but address neither whether it will resolve any existing agent/resource conflicts nor whether it will introduce new ones. While the overall repair algorithm will resolve any introduced conflicts, this results in many avoidable additional iterations of repair. We propose to incorporate a limited degree of reasoning about resources into our repair heuristics. The tradeoff between the complexity of a single repair iteration and the total number of iterations will need to be carefully balanced: while this will reduce the total number of repair iterations, taking resources into consideration expands the set of tasks which the repair heuristics need to reason over from the set of role set tasks to the (much larger) set of role filling tasks. We may be able to ameliorate this to an extent by structuring the data to enable the filtering of role filling tasks by the agents required.

ASPEN's default online repair approach incorporates a "commitment window". This is a window of time extending from the current time to a set distance into the future, within which ASPEN will not modify any task. This is intended to prevent the planner from modifying a task during planning that will begin to run before the planning session completes. An alternative to this approach, in the light of live task modification, would be to weight the planner towards repairing conflicts that occur earlier in the plan. The goal of this approach is to ensure that the short-term schedule is conflict-free while keeping individual planning sessions to lengths compatible with real-time execution, at the potential cost of conflicts further in the future. While this would not directly affect the overall amount of (re)planning that must be performed, it would allow the planner to make do with fewer repair iterations, speeding its response time.

Optimize a Single Repair Cycle While reducing the number of useless planning cycles should garner significant gains, sizable gains also will be realized by optimizing each repair cycle. We will do so by developing a search heuristic that takes advantage of the structure of tasks involving optional roles, as discussed in Section 5.2.1.

Reduce the Number of Optimization Iterations At the moment, we perform a set number of iterations of the optimization heuristic at every time step. This is useful since predicted task durations are updated at every step, potentially creating opportunities. Unfortunately, only about 6% of these iterations result in any optimization actions, yet they consume a considerable amount of computational power. Instead, we may want to consider approaches such as layered heuristics, in which we perform fast, inaccurate calculations to provide a loose upper bound on the potential benefits of optimization before invoking the computationally expensive optimization routines.

Optimize a Single Optimization Cycle The performance of each optimization cycle also will benefit from a search heuristic that takes advantage of relevant task structure (see Section 5.2.1).

We believe a critical path-based approach to optimization, such as ours, is likely to dominate most other approaches with respect to effectiveness, but a thorough survey of the plan optimization literature may provide additional options and insights. Less powerful but more efficient approaches may prove to be a better tradeoff. In addition, there has been recent work on building schedules by minimizing the resource-constrained critical path [40]. We may be able to adapt this, or related work, to increase the speed and improve the effectiveness of our optimization algorithm.

5.2.3 Addition, Setup, and Removal Tasks

Our work to date assumes that agents may be added or removed from tasks instantaneously, with no intermediate effects on the task, nor any required setup actions. This assumption will be removed in the thesis work, allowing the system more accurately to represent reality. Doing so will entail the addition of tasks representing the addition and removal of agents from a team, as well as a variety of setup actions, such as repositioning or warming up of an instrument.

The optimization problem will become significantly more difficult as we incorporate addition, setup, and removal tasks, since the task duration calculation will become much more complex. Instead of simply comparing two ways of accomplishing a task, we must compare the *status quo* with the time taken for the new agent to rendezvous with the team, the potential slowing of the team during agent addition (or removal), and finally the improved speed or robustness of the team once the new agent is incorporated. We will need to make tradeoffs between the accuracy of this calculation and its computational complexity, with direct effects on the effectiveness of our optimization heuristics. For instance, obtaining the best possible estimate of the length of a repositioning setup task would involve invoking a detailed path planner, which likely will be prohibitively expensive. Instead, we will develop duration prediction models for the addition, setup, and removal tasks, just as we do with "normal" tasks. These models likely will make use of rough approximations of elements such as path planning, in order to remain tractable.

5.2.4 Live Task Modification Summary

We propose to develop a live task modification algorithm and associated heuristics that can operate in real time by taking advantage of the structure inherent in tasks incorporating optional roles, as well as handle a wide range of scenarios, including explicit agent addition, setup, and removal tasks.

5.3 Architecture

Proactive replanning implies a much tighter connection between planner and executive than traditionally exists in planning/execution systems. In most such systems, the planner dispatches tasks to the executive, and is informed when they complete. In proactive replanning, rather than ceding all control over a task after dispatching it to the executive, the planner may modify the allocation of resources to a task during execution, and receives high-frequency updates on the task's state. In this way, the planner maintains its control over resource allocation throughout the execution of each task. This fundamental difference in planner-executive interaction raises a number of architectural issues, including the degree to which data is shared and conflict resolution. We propose to investigate the implications of this unique bond between planner and executive.

5.3.1 Data Sharing

One aspect of the architecture of proactive replanning is the degree to which the planner and executive share data. We propose to investigate how proactive replanning may be performed with varying amounts of data flow between the executive and planner, and the tradeoffs involved. In a traditional approach, this flow is quite minimal, generally consisting of notifications as tasks complete or fail and updates of the availability of agents and resources. In our current approach to proactive replanning, the planner performs task duration prediction throughout execution. In order to do so, it needs access to the current state of each task, greatly increasing the amount of data shared by the executive and the planner. We will investigate alternative approaches to proactive replanning that support a looser link between planner and executive.

There are two possible approaches to relaxing proactive replanning's current tight connection between planner and executive: reduce the amount of data being shared, or have the two operate on data at distinct degrees of abstraction.

One potential approach to reducing the amount of data sharing is to move the task duration prediction calculations into the executive and transmit the results to the planner. While architecturally satisfying, this may prove computationally infeasible, as the planner must evaluate the expected durations of many different tasks during the repair and optimization stages. For instance, our current approach makes on average 10-14 duration predictions per task per second. In addition, in a real-world domain, communication bandwidth between the agents and the centralized planner may be severely limited.

A more likely approach to relaxing the planner-executive connection is to provide the planner with data that is distinctly abstracted from that available to the executive. This will reduce the complexity and fidelity of the planner's calculations, reduce communications

flow between the two components, and clarify the distinction between planner and executive. The type of abstraction will depend heavily upon the task, but could incorporate elements such as a first-order approximation of path planning, a coarser discretization of continuous state variables such as location, or a simple enumerated task state variable (rather than a collection of continuous variables). We propose to investigate the effects of a variety of abstractions in order to understand how proactive replanning is affected by the closeness of the link between planner and executive.

5.3.2 Conflict Resolution

Regardless of how we define the split in responsibilities and data between proactive planner and executive, conflicts may arise between the planner's and executive's intentions. Specifically, the planner may request a change in a live team that the executive is unable to implement, either entirely or at the moment. Such a situation could arise when agents may be added or removed from a team only when it is in certain states. For instance, a transporter agent can be added to the *Transport_Panel* task only when the existing transporters are able to shift around the edge of the panel to make room for the new agent. At times, this may not be possible due to local terrain conditions or other factors known only to the executive.

We propose to develop techniques to effectively resolve such conflicts. Research questions include: whether the executive should refuse an invalid modification outright, or queue it for execution when the conditions are amenable; whether the executive should inform the planner of the conflict, and if so, what (if any) additional information should be provided; and what latitude the executive has to refuse modifications (e.g. can it only refuse when the modification will never be possible, when it is not possible at this instant, etc.).

5.3.3 Architecture Summary

The proactive planning architecture incorporates a much tighter link between planner and executive than is seen in more traditional planning / execution systems. We propose to investigate different approaches to dividing responsibilities and data between the two components, as well as evaluating how conflicts between the planner and executive may be resolved.

5.4 Characterization

Our experiments have validated our approach within a single, relatively simple, scenario. In order to truly understand the power provided by duration prediction and live task modification, we must broaden the scope of our domain to investigate the effects of a variety of scenario characteristics on the efficacy of our approach. This will provide us with a better understanding of the applicability of the approach, but more importantly will expose further opportunities to improve or generalize our algorithms as strengths and weaknesses of the current approach are highlighted. Once our exploration is complete, we will be able to describe how different scenario characteristics determine the usefulness of proactive replanning.

5.4.1 Task Structure

We will vary the structure of individual tasks to determine how it affects the gains realized through the use of duration prediction and live task modification. Specifically, the ratio of the delay caused by nonterminal failures to average task time, the number of possible teams that can accomplish the task, and the unimodal or multimodal structure of task duration distributions deserve our attention.

As the delays caused by failures increase, we hypothesize that our approach will be of increasing utility. The effectiveness of filling optional roles to decrease the likelihood and/or effect of a delay increases as the delay lengthens, making live task modification of more use. In addition, the ability to predict task duration becomes ever more advantageous as it exposes longer and longer "holes" into which other tasks may be scheduled. Thus, we predict that our approach will provide larger gains as the ratio of failure delay

to average task duration increases for tasks which make up a reasonable portion of the makespan. We will evaluate this hypothesis through a series of simulated experiments, in which we will systematically vary the degree to which failures delay task completion.

We will also investigate how the effectiveness of proactive replanning is affected by the number of distinct teams that can perform a given task. We posit that our approach will become increasingly effective as the number of teams increases, within limits. As more options become available, the planner is given more flexibility through live task modification to precisely adapt each team to the realities of execution. While our proposed approaches to searching the team space will allow proactive replanning to operate in large team spaces, they are not a panacea – if the team space becomes complex enough, we may be unable to search it while constrained to real time execution. We plan to determine where these bounds lie, and in what ways proactive replanning degrades as the team space expands.

Finally, we will evaluate a variety of underlying task duration distributions, and determine how they affect proactive replanning. We posit that if the underlying task duration distribution is multimodal, high fidelity duration prediction will yield greater gains than in the unimodal case, as it gives the planner a much more accurate representation of duration than is possible with traditional single-point or mean and variance estimates. This allows the planner to better manage deadlines, as discussed in Section 5.1.1.

5.4.2 Task Interactions

While our experimental scenario incorporates a variety of task interactions, such as temporal constraints at a number of levels, more complex scenarios may provide additional insight into the usefulness of our approach. Live task modification in particular shines in the presence of bottlenecks, such as those created by one-to-many or many-to-many temporal constraints, as it allows the planner to concentrate additional resources on the bottleneck as they become available. As we diversify the types and numbers of temporal constraints, the theoretical gains possible through proactive replanning will increase. We will systematically investigate the effects on proactive replanning of the number and type of constraints in the scenario.

We will also evaluate the effects of a diversification of resource requirements and the introduction of heterogeneous agents. We hypothesize that, in addition to increasing the fidelity of the planner's model, they will magnify the usefulness of our approach. In the real world, most agents can perform only a subset of the available tasks. This increases the likelihood of bottlenecks and opportunities, which we again hypothesize will in turn further increase the effectiveness of proactive replanning.

5.4.3 Characterization Summary

As part of the thesis work we plan to investigate the applicability of our approach to a broad range of scenarios, characterizing the effect of task and scenario characteristics on the effectiveness of proactive replanning. This exploration will both deepen our understanding of the strengths and weaknesses of our approach and expose further opportunities for improvement or generalization of our algorithms.

5.5 Evaluation

5.5.1 Simulation

We will evaluate our final system in both simulation and the real world Trestle assembly task. In addition to its current capabilities, the simulator will incorporate tasks that deviate from the models used for duration predictions, utilize metric locations, and include any other warranted extensions. We will primarily use the simulator to perform a wide-ranging characterization of our approach, as we can easily evaluate a variety of domains and task characteristics.

5.5.2 Real-World Scenario

To verify that our final approach can operate in real time and handle the many quirks associated with real hardware, we plan to incorporate proactive replanning into the existing Trestle project's [56] assembly scenario. The Trestle scenario consists of assembling a lattice of compliant nodes and beams, with a number of agents coordinating to perform each assembly. The project currently is moving to a new scenario and adding additional agents, which will provide opportunities for parallelization of assembly, agents with varying expertise, and tasks with a number of optional roles. Initially, these optional roles will consist primarily of additional sensing agents, but opportunities exist for tightly coordinated manipulation.

We currently envision the final task as the assembly of a 3 by 3 lattice, consisting of 9 nodes and 12 beams. There will be two primary tasks: docking one end of a beam to a node (*Dock_Beam*), and shifting the structure (*Shift_Structure*) (see Table 5). The scenario will consist of 24 *Dock_Beam*s, an indeterminate number of *Shift_Structure*s, and a variety of miscellaneous tasks not necessarily involving optional roles. Sufficient agents will be available to perform at least two *Dock_Beam* actions in parallel.

The *Dock_Beam* task is at the core of the scenario: it is the only assembly operation which must be performed in order to build the lattice. At a minimum, it requires two agents: a Docker (Beam) and an Observer. The Docker (Beam) holds and manipulates the beam in order to attach one end to a node, while the Observer provides it with data on the relative positions of the beam and node. The speed and reliability of the team may be enhanced by adding a Docker (Node) agent. A Docker (Node) manipulates the node in order to rotationally align it with the beam. This frees the Docker (Beam) to concentrate on the remaining degrees of freedom. Since the node is statically stable, the Docker (Node) agent may be added or removed from the team at any point during execution. Two additional Observers may also be included in order to increase the team's robustness. Neither of the Docker agents has the means to determine if its grasp on the beam or node is slipping; the additional Observers monitor one or both of the grasps to provide the Dockers with enough warning to maintain their grips. As with the Docker (Node) agent, these additional Observers may be added or removed at any point in the execution of the task. In addition, *Dock_Beam* has a number of preconditions that will be fulfilled by setup tasks: the agents must be positioned properly, the Docker (Beam) must be grasping the beam, and the Docker (Node) (if present) must be grasping the node. The relevant setup tasks will need to be performed both initially and prior to the addition of any agents.

The Trestle team includes at least one nonmobile agent (a large, fixed crane) and will be assembling the lattice in a workspace only slightly larger than the final structure. As a result, the partially assembled structure will need to be repeatedly repositioned in order to allow mobile agents to move around it and to ensure that the relevant portion is within the crane's workspace. These moves will be accomplished by the *Shift_Structure* task. The task may be performed by a single Shifter agent operating via dead reckoning. However, the partially assembled structure will contain significant compliance at the beam-node joints. To accommodate this, an arbitrary number of additional Shifter agents may be added to the team. They will grasp other points of the structure and move in synchrony to reposition the structure with less overall deformation. In addition, for long or complicated shifts, dead reckoning may be too inaccurate. In this case, an arbitrary number of Observers may be added to the team, with each monitoring a Shifter or a portion of the assembly to determine when the move has been successfully completed. The Observers also monitor for potential breakage of the structure: if the compliance of a beam-node joint is exceeded, the beam may break free from the node. In this way, Observers improve both the accuracy and robustness of *Shift_Structure*. The task incorporates similar setup tasks to *Dock_Beam*, in that all agents must position themselves, and the Shifters must grasp the structure prior to moving it. While Observers may be added or removed at will, the team's movement must be temporarily interrupted to add or remove a Shifter.

The *Dock_Beam* and *Shift_Structure* tasks lend themselves well to proactive planning. *Dock_Beam* incorporates three optional roles that may be filled or vacated at will, but which require not insignificant setup actions. While seemingly simple, *Shift_Structure* requires more closely coordinated actions in order to add or remove agents from some of its optional roles. These characteristics will allow proactive replanning to shift agents between tasks as execution proceeds in order to minimize the executed makespan.

We will demonstrate the effectiveness of proactive replanning by performing a series of assemblies using either proactive replan-

Task	Role	Role Type	Capacity	Effect
Dock_Beam	Docker (Beam)	Required	1	Holds and manipulates the beam in order to dock one end into a node.
	Docker (Node)	Optional	1	Holds and manipulates the node, controlling alignment to allow the Docker (Beam) agent to focus on the remaining degrees of freedom.
	Observer	Required	1	Observes the beam and node, reporting relative positioning information.
	Observer	Optional	2	Observes one or both of the Docker agents to monitor their grasps (they have no way to detect a slipping grip).
Shift_Structure	Shifter	Required	1	Grasps and moves the partially-completed structure to clear travel lanes or bring it within the workspace of nonmobile team members.
	Shifter	Optional	N	Assists in the structure movement by grasping other points and moving in synchrony. Additional shifters help to keep a compliant structure from deforming.
	Observer	Optional	M	Visually monitors the structure for placement and potential breakage. The task can be performed via dead-reckoning with no Observers, but the results will be less precise and robust.

Table 5: Two tasks in the envisioned real world Trestle assembly scenario which will benefit from proactive replanning.

ning or a more traditional repair-based method akin to the current "Baseline" experimental condition.

5.5.3 Evaluation Summary

We will evaluate proactive replanning in both simulation and a real world assembly task through a series of experiments. In simulation, we will concentrate on characterizing and evaluating our approach across a wide range of scenarios. The real world assembly task will test proactive replanning's effectiveness in a noisy, difficult to model real-time scenario. We hypothesize that it will achieve a more efficient execution of both simulated and real world scenarios than possible with a baseline planner.

5.6 Proposed Research Summary

The thesis work will strive to fully understand the power of proactive replanning by characterizing its effectiveness against a number of different scenario variables, and demonstrate its usefulness in real-time settings. Evaluation will take place in both a broad range of simulated domains and the real world Trestle assembly scenario.

6 Conclusion

Inspired by the fluidity of human teams, we propose to develop a real-time planning/execution system capable of predicting task durations and modifying currently executing tasks. These two capabilities, not before seen in a planning/execution system, will allow the system to execute more efficient final plans than are otherwise possible. We have established the efficacy of our approach through a proof-of-concept system operating 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.

The thesis work will strive to understand the power and applicability of proactive replanning. In doing so, we will further investigate duration prediction, live task modification, and the architecture of a proactive replanning system. We will develop algorithms for duration prediction that produce and adapt high fidelity estimates without consuming an undue amount of computational or spatial resources. The thesis work will expand our live task modification heuristics to better utilize task structure, represent agent addition and removal tasks, and handle a wider range of scenarios. We also will optimize and extend our algorithms to enable real time proactive replanning on a real world assembly system. The architecture of proactive replanning will be investigated, especially the degree to which the planner and executive must be integrated. Finally, we will explore the applicability and usefulness of our approach by characterizing how task structure and interactions affect the gains realized through its use.

7 Schedule

A proposed timeline for the completion of my thesis research is as follows:

Summer 2006

- Focus on duration prediction, increasing the fidelity of predictions and investigating approximations to reduce the prediction function's domain.
- Begin integration of ASPEN with Trestle system.

Fall 2006

- Develop algorithms for the adaptation and verification of duration prediction models.
- Begin development of a graph-based search of the task space.
- Continue ASPEN-Trestle integration.

Spring 2007

- Complete graph-based task space search.
- Optimize repair and optimization algorithms.
- Begin implementation of support for addition, setup, and removal tasks within live task modification.
- Complete ASPEN-Trestle integration.

Summer 2007

- Complete support for addition, setup, and removal tasks.
- Perform simulator-based characterization of system.
- Develop formal characterization of approach.

Fall 2007

- Perform experiments on Trestle hardware.
- Write and defend thesis.

References

[1] PSPlib: Project scheduling problem library. URL: http://129.187.106.231/psplib/.

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

[3] J. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1988.

[4] F. Bacchus and M. Ady. Planning with resources and concurrency: A forward chaining approach. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

[5] M. Beetz and D. V. McDermott. Improving robot plans during their execution. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 1994.

[6] T. Belker, M. Hammel, and J. Hertzberg. Learning to optimize mobile robot navigation based on htn plans. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '03)*, volume 3, pages 4136–4141, Sept 2003. DOI: 10.1109/ROBOT.2003.1242233.

[7] Doug Bernard, Gregory A. Dorais, Ed Gamble, Bob Kanefsky, James Kurien, William Millar, Nicola Muscettola, Pandu Nayak, Nicolas Rouquette, Kanna Rajan, Ben Smith, Will Taylor, and Yu-Wen Tung. Final report on the remote agent experiment. In *Proceedings of NMP DS-1 Technology Validation Symposium*, Pasadena, CA, February 2000.

[8] P. Bertoli, A. Cimatti, M. Roverie, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

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

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

[11] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 2000.

[12] Scott M. Brown, Eugene Santos Jr., and Sheila B. Banks. Utility theory-based user models for intelligent interface agents. In *Proceedings of the Twelfth Canadian Conference on Artificial Intelligence (AI '98)*, Vancouver, British Columbia, Canada, June 1998.

[13] Peter Brucker, Andreas Drexl, Rolf Mohring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112:3–41, 1999.

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

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

[16] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[17] N. Correll and A. Martinoli. Collective inspection of regular structures using a swarm of miniature robots. In *Proceedings of the Ninth International Symposium on Experimenatl Robotis (ISER-04)*, number 6 (2005) in Springer Tracts in Advanced Robotics, Singapore, June 2004.

[18] K. Currie and A. Tate. O-plan: the open planning architecture. *Artificial Intelligence*, 52, 1991.

[19] Sajal K. Das, Diane J. Cook, Amiya Bhattacharya, Edwin O. Heierman III, and Tze-Yun Lin. The role of prediction algorithms in the mavhome smart home architecture. *IEEE Wireless Communications*, pages 77–84, December 2002.

[20] M. B. Dias, S. Lemai, and N. Muscettola. A real-time rover executive based on model-based reactive planning. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2003.

[21] Kevin R. Dixon and Pradeep K. Khosla. Programming complex robot tasks by prediction: Experimental results. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, October 2003.

[22] S. E. Elmaghraby. *Activity networks: Project planning and control by network models.* Wiley, New York, 1977.

[23] R. Emery, K. Sikorski, and T. Balch. Protocols for collaboration, coordination and dynamic role assignment in a robot team. In *Proceedings of the IEEE International Conference on Robotics and Automation, 2002 (ICRA'02)*, volume 3, pages 3008–3015, Washington, DC, May 2002. DOI: 10.1109/ROBOT.2002.1013689.

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

[25] Tara Estlin, Daniel Gaines, Caroline Chouinard, Forest Fisher, Rebecca Castano, Michele Judd, Robert C. Anderson, and Issa Nesnas. Enabling autonomous rover science through dynamic planning and scheduling. In *Procedings of the IEEE Aerospace Conference*, 2005.

[26] P. Ferraris and E. Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2000.

[27] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3, 1972.

[28] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 1971.

[29] R. J. Firby. Task networks for controlling continuous processes. *Artificial Intelligence Planning Systems*, 1994.

[30] Maria Fox and Derek Long. Pddl 2.1: An extension to pddl for expressing temporal planning domains. Technical report, University of Durham, UK, 2002.

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

[32] E. Gat. 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)*, 1992.

[33] E. Gat. Esl: A language for supporting robust plan execution in embedded autonomous agents. In *Proceedings of the 1997 IEEE Aerospace Conference*, 1997.

[34] B. P. Gerkey and M. J. Mataric. A formal framework for study of task allocation in multi-robot systems. Technical Report CRES-03-13, University of Southern California, 2003.

[35] M. Ghallab and H. Laruelle. Representation and control in ixtet, a temporal planner. In *AIPS*, pages 61–67, 1994.

[36] K. Z. Haigh and M. M. Veloso. Planning, execution and learning in a robotic agent. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 1998.

[37] Geir E. Hovland and Brenan J. McCarragher. Hidden markov models as a process monitor in robotic assembly. *The International Journal of Robotics Research*, 1996.

[38] James Jennings and Chris Kirkwood-Watts. Distributed mobile robotics by the method of dynamic teams. In *???*, 1998.

[39] A. Karalic. Linear regression in regression tree leaves. In *Proceedings of ISSEK '92 (International School for Synthesis of Expert Knowledge)*, 1992.

[40] Kyunghwan Kim and Jesus M. de la Garza. Evaluation of the resource-constrained critical path method algorithms. *Journal of Construction Engineering and Management*, pages 522–532, May 2005. DOI: 10.1061/(ASCE)0733-9364(2005)131:5(522).

[41] Rainer Kolisch and Sonke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 2005. In press, available online May 23, 2005: http://dx.doi.org/10.1016/j.ejor.2005.01.065.

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

[43] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, 1982.

[44] Solange Lemai and Felix Ingrand. Interleaving temporal planning and execution: Ixtet-exec. In *???*, ???

[45] Ming Li and Allison M. Okamura. Recognition of operator motions for real-time assistance using virtual fixtures. In *Proceedings of 11th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (2003 IEEE Virtual Reality Conference)*, Los Angeles, CA, March 2003.

[46] Maja J. Mataric. Designing emergent behaviors: from local interactions to collective intelligence. In J. A. Meyer, H. Roitblat, and S. Wilson, editors, *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior (SAB-92)*, pages 432–441, Cambridge, MA, 1992. MIT Press.

[47] Maja J. Mataric, Gaurav S. Sukhatme, and Esben H. Ostergaard. Multi-robot task allocation in uncertain environments. *Autonomous Robots*, 14:255–263, 2003.

[48] Nicolas Meuleau, Richard Dearden, and Rich Washington. Scaling up decision theoretic planning to planetary rover problems. In *AAAI-04: Proceedings of the Workshop on Learning and Planning in Markov Processes Advances and Challenges*, volume Technical Report WS-04-08, pages 66–71, Menlo Park, CA, 2004. AAAI Press.

[49] N. Muscettola. *Intelligent Scheduling*, chapter HSTS: Integrating planning and scheduling. Morgan Kaufmann, 1994.

[50] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1-2), August 1998.

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

[52] D. Nau, H. Mun oz Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning with partially ordered subtasks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

[53] I.A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and Won Soo Kim. Claraty: An architecture for reusable robotic software. In *Proceedings of the SPIE Aerosense Conference*, Orlando, Florida, April 2003.

[54] Krzysztof Pawlikowski, Donald C. McNickle, and Gregory Ewing. Coverage of confidence intervals in sequential steady-state simulation. In *Proceedings of the 1995 EUROSIM Congress*, 1995.

[55] J. Quinlan. Learning with continuous classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, 1992.

[56] Brennan Sellner, Frederik W. Heger, Laura M. Hiatt, Reid Simmons, and Sanjiv Singh. Coordinated multi-agent teams and sliding autonomy for large-scale assembly. *Special Issue of the Proceedings of the IEEE on Multi-Robot Systems*, 2006. In Press.

[57] Brennan Sellner, Reid Simmons, and Sanjiv Singh. User Modelling for Principled Sliding Autonomy in Human-Robot Teams. In Lynne Parker, Frank Schneider, and Alan Schultz, editors, *Multi-Robot Systems: From Swarms to Intelligent Automata*. Springer, 2005.

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

[59] D. Smith and D. Weld. Conformant graphplan. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1998.

[60] D. Smith and D. Weld. Temporal planning with mutual exclusion reasoning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[61] Peter Stone and Manuela Veloso. Task decomposition and dynamic role assignment for real time strategic teamwork. In *Proceedings of 5th International Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages (ATAL '98)*, Paris, France, July 1998.

[62] Malcolm Strens and Neil Windelinckx. Combining planning with reinforcement learning for multi-robot task allocation. *Adaptive Agents and MAS II, LNAI 3394*, pages 269–274, 2005.

[63] Milind Tambe, Jafar Adibi, Yaser Al-Onaizan, Ali Erdem, Gal A. Kaminka, Stacy C. Marsella, and Ion Muslea. Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence*, 110:215–239, 1999.

[64] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. Claraty: Coupled layer architecture for robotic autonomy. Technical report, Jet Propulsion Laboratory, December 2000.

[65] D. Weld, C. Anderson, and D. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1998.

[66] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, 1988.

[67] W. A. Woods. Transition network grammars for natual language analysis. *CACM 13*, 10:591–606, October 1970.

[68] W. A. Woods. *Natural Language Processing*, chapter An experimental parsing system for transition network grammars, pages 111–154. Algorithmics Press, New York, 1973.