Duration Prediction for Proactive Replanning

Brennan Sellner and **Reid Simmons**
bsellner@andrew.cmu.edu, reids@cs.cmu.edu
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Abstract

Proactive replanning attempts to predict scheduling problems or opportunities and adapt to them throughout a schedule's execution. By continuously predicting a task's remaining duration, a proactive replanner is able to accommodate upcoming problems or opportunities before they manifest themselves. We have developed a kernel density estimation-based method for predicting a task's duration distribution as it executes, and have integrated our prediction algorithm with an existing planner based on heuristic repair. Our predictor allows the planner to anticipate problems, or opportunities, early enough to avoid, or take advantage of, them, resulting in executed schedules that score significantly higher on a number of metrics. We have evaluated a limited form of our approach in simulation, and present the results of our experiments. *Duration prediction* achieves an average reward 28.5% higher than the baseline, with 35.9% more reward-laden tasks executed within a fixed horizon.

Introduction

When working with others, humans often exchange information about their progress on the tasks at hand and whether they will likely complete their work on time. This allows each individual to adapt his schedule to make the best use of his time. For instance, the foreknowledge that a group meeting will be delayed by an hour because the team leader is caught in traffic allows everyone to take on an appropriate task during their now-free window. Many planning and execution systems, however, do not predict how long executing tasks will take to complete. Instead, they assume each task will take as long as it was scheduled for and react only when tasks complete early or over-run their scheduled times, resulting in suboptimal execution.

Proactive replanning encompasses the prediction of problems, or opportunities, such as these, and the adaptation of the schedule to avoid, or take advantage of, them before they occur. This allows the proactive replanner to modify its schedule early enough to accommodate the realities of execution: by predicting the team leader's late arrival from his current location and the state of the roads, a proactive replanner would schedule additional tasks into the now-empty hour for the remainder of the team, and move tasks aside to accommodate the delayed meeting.

A vital element of proactive replanning is the prediction of a task's duration during its execution, a component we refer to as *duration prediction*. Duration prediction is the prediction of a task's remaining duration, given a measurement of its current state. Rather than predicting a single estimate of the duration, we estimate a distribution across the possible task durations.

By predicting a duration distribution, it becomes possible for the planner to engage in several new strategies that are unattainable if we simply predict a scalar duration. One such strategy is multi-metric optimization. Because the form of the duration distribution varies greatly between tasks, reducing the time allocated to two tasks by the same amount will have different effects on the likelihood that each task will overrun its new scheduled time. The planner will be able to leverage these differences to reason in a principled fashion about trading off the likelihood of a task over-running its allocated time against other metrics, such as makespan or total reward.

Another planning capability we are researching is *live task modification*, which consists of transferring agents between teams while tasks are executing. In order to do so, we must be able to predict the effect of such a transfer. Since a physical agent cannot be moved instantaneously, there will be some uncertainty as to when it will join the receiving team, affecting the utility of the transfer. In order to evaluate the effect of a proposed live task modification in a principled manner, we must begin with duration distributions of the arrival time and receiving task. Scalar estimates of duration provide insufficient information to accurately reason about risk management and the utility of a live task modification. For instance, with a scalar duration estimate, it is impossible to predict the likelihood that the transferred agent will arrive in time to be useful, and small perturbations in the available data could result in large changes in the predicted usefulness of a transfer.

We have developed a kernel density estimation-based approach to duration prediction that enables the estimation of the duration distribution given relatively sparse training data, in addition to a measure of the task's current state. Training data is necessarily sparse, since the state space of tasks usually involves several continuous dimensions, making it extremely difficult to collect a dense set of data outside of simulation. This necessitates the use of function approximation techniques to estimate the duration distribution.

Kernel density estimation (KDE) is a nonparametric es-

timation method related to histograms that allows arbitrary distributions to be easily approximated from training data, without making *a priori* assumptions about the form of the underlying distribution (Silverman 1986). This flexibility makes KDE well-suited to problems such as ours, where the duration distribution of a task can take on any form due to potential failures, environmental variations, or other factors.

We have evaluated our approach to duration prediction using the ASPEN planner (Chien *et al.* 2000b) and a high-level stochastic execution simulator. In this initial evaluation, we predict the duration of the task as the mean of its distribution, and evaluate the effect of predicting miscoordinations on the final executed schedule. We have not yet evaluated live task modification, nor multi-metric optimization. Our experimental results indicate that the use of duration prediction increases the total reward and number of executed tasks over the baseline by a statistically significant degree: 28.5% greater reward is achieved, with 35.9% more reward-laden tasks executed within a fixed horizon.

Related Work

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 function approximation. We are interested in predicting a distribution across a continuous metric (remaining duration) given a (potentially large) collection of continuous and discrete state inputs (the current task state), under the Markovian assumption. There are two elements to this problem: (1) predicting the duration distribution at a specific point in the state space, and (2) generalizing this to allow predictions across the entire space with relatively sparse training data.

The first portion of the problem has been well-studied by the function approximation community. Parametric distributions, such as the gamma and normal, can be fit to arbitrary data using approaches such as maximum likelihood estimation (Kay 1993). However, parametric distributions make assumptions about the underlying distribution that may not hold, especially when predicting the duration of tasks executed in dynamic, uncertain environments.

Nonparametric approaches such as thin-plate splines (Bookstein 1989) and piecewise linear regression are able to fit arbitrary functions, and in general are sufficient for the first portion of the problem. However, they break down when generalizing across larger numbers of dimensions.

Approaches such as multivariate adaptive regression splines (Friedman 1991), locally weighted projection regression (Vijayakumar & Schaal 2000), model trees (Belker, Hammel, & Hertzberg 2003) (Quinlan 1992), ensemble regression modeling (Merkwirth *et al.* 2004), and neural networks are capable of approximating functions from high-dimensional input spaces. While all could in theory be utilized for duration prediction, fitting times tended to be long and over-fitting often occurs in our domain.

We have selected a modified form of kernel density estimation (KDE) (Silverman 1986) as our prediction method. KDE is a relatively simple nonparametric approach that allows the estimation of arbitrary distributions. Distributions can be generated relatively quickly from multi-dimensional data, and are not subject to any assumptions about the structure of the distribution. KDE will be discussed in detail in the Approach section.

Planning and Execution

Integrating duration prediction with a planning and execution system imposes a number of constraints on the planner and the architecture it fits into. In order to make the most use of duration prediction, the planner must support durative actions, temporal constraints, metric resources, and be able to quickly replan or repair a plan in response to feedback from the executive. Three planners that meet these requirements are ASPEN (Chien *et al.* 2000b), EUROPA (Frank & Jónsson 2003), and IxTeT (Laborie & Ghallab 1995). We have used ASPEN as the basis of our work to date, as ASPEN's most-commitment strategy makes the implementation of proactive replanning somewhat less complex. However, we are not aware of any fundamental problems that would preclude the use of proactive replanning with least-commitment planners.

In addition to the planner requirements, duration prediction needs a tight connection between planner and executive: the executive must be able to provide the planner with relatively high-frequency state updates. Classic three-layer architectures such as 3T (Bonasso *et al.* 1997) and ATLANTIS (Gat 1992) generally have planning/executive ties that are too loose to support duration prediction. LAAS's unnamed architecture (Alami *et al.* 1998) and CLARAty (Nesnas *et al.* 2003) both encapsulate planning and execution into a single layer, and provide sufficient pathways between planner and executive. LAAS uses the IxTeT planner, while CLARAty has two executives available: CASPER (Chien *et al.* 2000a) and CLEaR (Estlin *et al.* 2001). CASPER uses the ASPEN planner and a simple executive, while CLEaR extends CASPER with a TDL-based (Simmons & Apfelbaum 1998) executive. While both LAAS's architecture and CLARAty are amenable to duration prediction, we chose CLARAty and CASPER due to our use of ASPEN. For our initial investigations, we opted for the simpler CASPER executive, as we do not yet need the flexibility of CLEaR, although we may move from CASPER to CLEaR in the future. IDEA (Muscettola *et al.* 2002) makes use of EUROPA and also is flexible enough to support our work.

Approach

Duration prediction allows the planner to recognize future scheduling problems and opportunities in time to address or take advantage of them. For instance, if a task is predicted to over-run, it will delay other tasks, potentially creating opportunities to insert tasks into the predicted window of now-idle time. Without duration prediction, the planner would miss such opportunities. In addition, this provides the planner with a longer time window in which to repair or optimize the plan before execution reaches the problem point. This reduces the likelihood that execution must be paused to allow the planner to resolve scheduling difficulties, and increases overall efficiency.

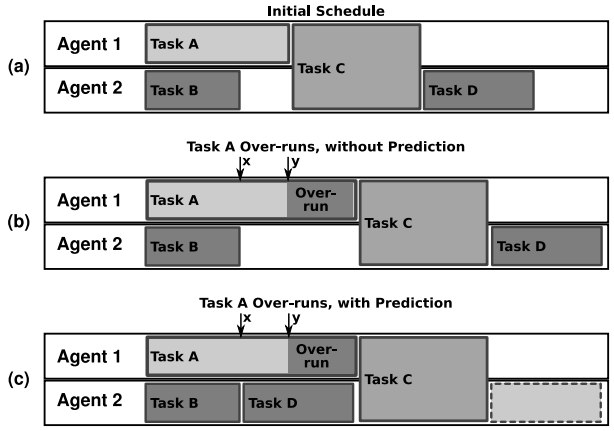

Figure 1: Predicting the remaining duration of executing tasks allows the planner to make use of opportunities presented by task over-runs.

Duration prediction allows the prediction of two classes of execution anomalies: under-runs and over-runs. When a task is predicted to under-run, setup actions for any subsequent tasks may be started early, decreasing or eliminating dead time between tasks. When an over-run is predicted, agents participating in now-delayed multi-agent tasks are able to fill the window with useful work, rather than incrementally waiting until the slow task completes. If prediction were unavailable, there would be no way to know whether the over-running task would complete in the next second or in half an hour, and agents committed to the delayed multi-agent tasks would lie idle until the slow task completed, unable to perform any useful work in the meantime.

For instance, Figure 1 depicts a canonical example of an over-running task. In the initial schedule (Figure 1(a)), agents 1 and 2 perform individual tasks (A and B) prior to a group task (C). Suppose that task A over-runs, as presented in Figure 1(b), with the over-run occurring at point Y. If duration prediction is not being performed, the planner will not realize in time that there may be space for an additional task in agent 2's schedule. However, if the planner is able to predict the over-run by point X, agent 2 will be able to execute task D earlier (Figure 1(c)). This both reduces the makespan of this segment of the schedule and provides additional time for other tasks to be scheduled later on.

Duration prediction also enables the exploitation of under-runs. Consider the scenario presented in Figure 2: agent 1 performs the single-agent task A, after which agents 1 and 2 are scheduled to execute the multi-agent task B. The BPrep task is a setup task for task B, and must be performed immediately prior to B. The initial schedule is depicted in Figure 2(a).

If task A completes early (Figure 2(b)), BPrep 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 BPrep immediately upon A's (early) completion (Figure 2(c)). However, this is inefficient, as BPrep may be executed in parallel with A. If the

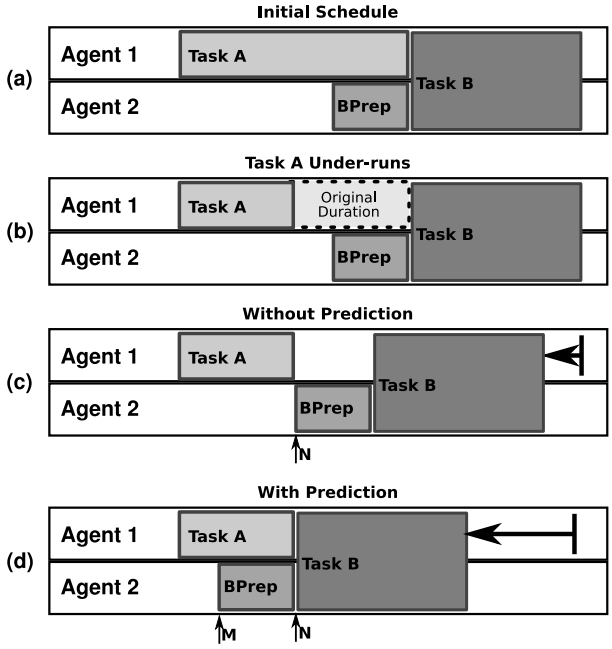

Figure 2: Duration prediction allows the planner to start setup tasks early when a preceding task is predicted to under-run.

planner were able to predict A's true completion time prior to point N, it would be able to start BPrep even earlier, realizing a further reduction in makespan. Ideally, the prediction would be made prior to point M (that is, $Length(BPrep)$ seconds before A's early completion), allowing B to be scheduled immediately after A, and BPrep to be executed entirely in parallel with task A (Figure 2(d)).

The goal of duration prediction is to predict a distribution across the possible durations of the remainder of an executing nondeterministic task, in order to allow the planner both to take advantage of execution anomalies and to support additional functionality, such as live task modification. When performing duration prediction, we assume an estimate of the task's current state is available, as well as a relatively sparse corpus of data from previous executions. The majority of state variables are continuous, with a sprinkling of discrete dimensions. The training data consists of a series of observations from previous executions of the task, and is not guaranteed to cover the state space, much less provide a large number of observations at each state. While training data is easy to obtain in simulated environments, it is difficult and expensive to collect when working with actual robots.

Prediction Method

We use a form of kernel density estimation (KDE) (Silverman 1986) to predict duration distributions. KDE is a non-parametric method related to histograms that is used to estimate an arbitrary distribution from training data. A histogram can be thought of as a set of unit-height blocks, where each observation generates a block. The blocks are

aligned with the histogram bins into which the corresponding observations fall, and are stacked (summed) when multiple blocks fall into a single bin. A simple kernel density estimator performs in a similar fashion, except that each block (the *kernel*) is centered on the observation, rather than on a discrete bin. Summing these blocks results in a step-wise function. In practice, rather than using a discrete block-like kernel, KDE will use a smoother function, often the normal distribution. An example of this is depicted in Figure 3. The five observations are denoted with circles. At each observation, we have centered a normal kernel (dashed lines), and summed them to yield the estimated distribution (solid line). The selection of the shape and bandwidth of the kernel affect the resulting distribution. In the case of a normal kernel, the bandwidth is the standard deviation of the kernel distribution. If it is too narrow, the result will have too many modes; too wide, and the distribution will become an undifferentiated mass. In these experiments, we have found that a bandwidth of 2.5 time units yields reasonable results.

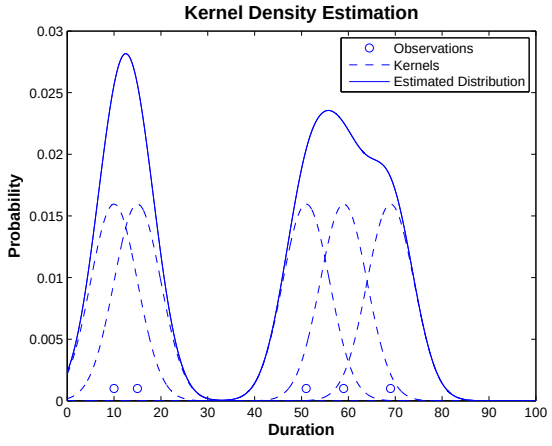

Figure 3: A simple example of kernel density estimation. Kernels (dashed lines) are centered at each of the five duration observations (plotted as 'o's), then the kernels are summed to build the estimated distribution (solid line).

Denote the bandwidth as h, the kernel function as $K(x,h)$, and let there be n observations with duration values x_i. The density of the distribution at a duration x is then the sum of the density contributed by the n kernels: $f(x) = \frac{1}{n}\sum_{i=1}^{n} K(x - x_i, h)$, where in our case $K(x,h) = \frac{1}{h\sqrt{2\pi}} exp\left(-\frac{x^2}{2h^2}\right)$ and $h = 2.5$.

We actually use a weighted form of KDE in order to represent the belief that observations from points near the task's current state are more relevant. In this version of KDE, each observation is assigned a relative weight w_i, where $\sum_{i=1}^{n} w_i = 1$. This allows observations "closer" to the task's current state to be weighted more. The density function is nearly identical to the canonical KDE approach, simply replacing the uniform weighting with the observation-specific weight: $f(x) = \sum_{i=1}^{n} w_i K(x - x_i, h)$

Using KDE, it is straightforward to transform a weighted set of durations into the desired distribution. However, the question of how to select and weight the appropriate set remains.

Let us refer to the current task state as the *query point*, a tuple Q of length d, where d is the dimensionality of the task's state space and Q_j is the current value of the jth state variable. Note that each observation is a tuple of length $d+1$, consisting of the state and the duration observed there. We must now determine the set of observations and associated weights that KDE will use to build the duration distribution. We do so by applying a *query kernel* along each dimension of the state space (Figure 4) and combining the resulting weights for each observation. A query kernel is a normal distribution centered at Q_j, with bandwidth h_j, that is used to calculate the weight of each observation for dimension j. The values of h_j are empirically selected, and depend upon the characteristics of the task. For instance, a continuous dimension may have a relatively large h_j, while a discrete dimension that represents a few very different cases may use a very small h_j to keep the cases segregated.

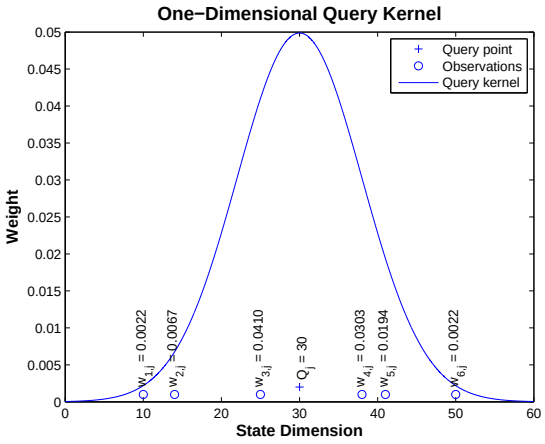

Figure 4: The query point (the current value of this dimension's state variable) is denoted with a '+', candidate observations with 'o's, and the query kernel as the solid curve. The weight of an observation i for this dimension, $w_{i,j}$, is the likelihood that the observation would be drawn randomly from the query kernel.

The weight in dimension j of the ith observation is simply the likelihood that $x_{i,j}$ (the observation's value for dimension j) would be randomly drawn from the query kernel: $w_{i,j} = K(x_{i,j} - Q_j, h_j)$. To limit computation, we consider observations only where $w_{i,j} > \epsilon$.

This weight calculation is performed for all j dimensions, resulting in a set of weights $w_{i,j}$ for the i observations. The final weight of an observation to be used for KDE is the normalized product of these per-dimension weights: $w_i = \frac{\prod_{j=1}^{d} w_{i,j}}{\sum_{i=1}^{n}\prod_{j=1}^{d} w_{i,j}}$. Once these per-observation weights are calculated, building the duration distribution is simply a matter of performing KDE as outlined above with the

weights w_i and the observed durations $x_{i,d+1}$.

Planner Integration

We have integrated duration prediction with the repair-based ASPEN planner (Chien *et al.* 2000b) and CASPER executive (Chien *et al.* 2000a). Although ASPEN lacks explicit multi-agent support, its capabilities are a good fit to the needs of duration prediction during execution. Given a set of goals, a plan (initially empty), and a set of resource timelines representing agents and their locations, ASPEN performs iterative plan repair to resolve conflicts and other flaws in the schedule. Repair and optimization steps may be interleaved as the state of the system is updated, using a most-commitment strategy (all variables are grounded as early as possible). This strategy makes the evaluation of metrics and projections of resource usage much simpler, but reduces the plan's flexibility.

We have tightened the integration between planner and executive by providing a conduit for state updates to flow to the planner at every timestep. As the updates arrive, new duration predictions are triggered, and ASPEN's schedule is updated. ASPEN is then able to repair any resulting conflicts or take advantage of opportunities that have become apparent. ASPEN represents tasks as having a single duration, so we utilize the mean of the predicted distribution as the expected duration for a task. We do not currently use the duration distributions to perform multi-metric optimization or live task modification, but these are active areas of research.

Execution Modeling

Task execution is stochastically modeled using a representation similar to Augmented Transition Networks (Woods 1970). The models introduce a degree of uncertainty akin to that found in real-world robotic teams. They model only the high-level progress of a task, including nonterminal failures, and report task-level state to the CASPER executive. For instance, while individual components of agents (such as manipulators or sensors) are not simulated, events such as an agent becoming stuck in the sand are represented. Figure 5 depicts the model used for the *Move* task. Execution begins in the Moving state, and one state transition is made per time unit. During normal operation, the distance traveled is incremented by a value drawn from a normal distribution, and there is a 1% chance of the agent becoming stuck during any given time step. Once the agent becomes stuck, it must recover before further progress can be made towards the goal.

Such models are used both during execution and to provide the initial observations needed to build the KDEs used by duration prediction. The amount of training data provided varies according to the uncertainty associated with the task, ranging from an average of 13 to 60 data points per cell, with no guarantee that any given cell will contain data. Note that the actual variables are continuous, and KDE does not require discretization; these reference figures are derived by discretizing the state space into cells with the same size as the average progress made during a time step. For instance, the Move task makes an average of 1 unit of progress per time step, both when moving and recovering. Since the two state variables (distance traveled and distance to recovery) lie in the range $[0, 50]$, this yields $50 * 50 = 2500$ cells. 34,776 training observations were made available for Move, yielding an average coverage of 13.9 observations per cell. While this may appear to be a large amount of training data, it is unevenly distributed, and nearly all of the duration distributions are highly multi-modal, thus requiring more data to achieve even a rough approximation. We will be examining the effects of varying the available amount of training data in the near future.

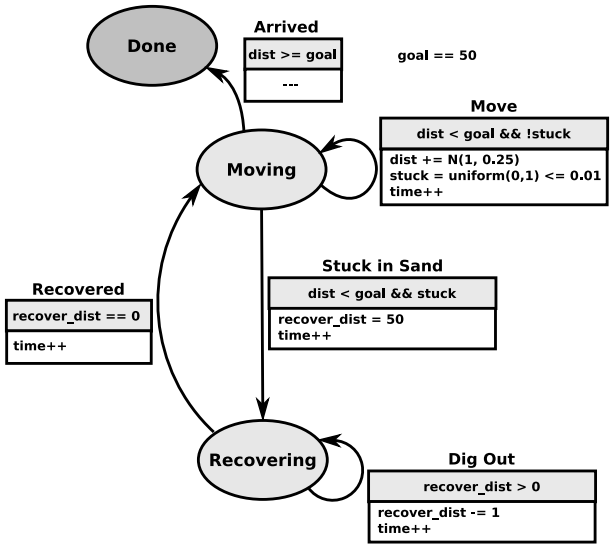

Figure 5: The stochastic simulation model for *Move*. This is used to simulate execution, and provides the state that is used for duration prediction during execution. In addition, this model is used offline to generate the observations used by the KDE-based predictor.

Results

We have evaluated the effectiveness of duration prediction for proactive replanning by performing a series of experiments in simulation.

Scenario

The scenario represents a subset of the construction of a lunar outpost, and includes three agents, three sites, and eight types of tasks (Table 1, Figure 6). Agents are assumed to be homogeneous, and can participate in only one task at a time. Each task has an associated reward, may require any number of agents, may be constrained to occur at a given site, and/or may involve the moving of agents from one site to another. In general, there are two classes of tasks: cooperative and solo. Cooperative tasks require more than one agent, and generally have higher rewards and durations than solo tasks.

The objective is to maximize total reward within a fixed horizon, which roughly equates to maximizing the number of reward-laden tasks that are executed. Duration prediction improves the final executed schedule by allowing tasks to be

Task	Reward	Agents Needed	Location	State Variables	Progress Per Timestep	Mean Duration from Start
Move	0	1	Start: Anywhere End: Anywhere	Distance moved: [0,50] Error recovery: [0,50]	$N(1, 0.25)$	50
Sky Observation	15	1	Anywhere	Progress: [0,1]	$N(0.05, 0.01)$	18
Soil Observation	15	1	Anywhere	Progress: [0,1]	$N(0.025, 0.01)$	38
Habitat Maintenance	30	1	Habitat	Progress: [0,1] Error recovery: [0,20]	$N(0.025, 0.01)$	50
Materials: Lander → Habitat	300	3	Start: Lander End: Habitat	Distance moved: [0,50] Error recovery: [0,50]	$N(0.5, 0.1)$	102
Lay Cable	120	2	Start: Habitat End: Comm	Distance moved: [0,50] Error recovery: [0,50]	$N(0.3, 0.25)$	258
Materials: Lander → Comm	100	2	Start: Lander End: Comm	Distance moved: [0,50] Error recovery: [0,100]	$N(2, 0.25)$	26
Comm Setup	50	2	Comm	Progress: [0,1] Error recovery: [0,200]	$N(0.05, 0.01)$	18

Table 1: Scenario tasks and relevant statistics. All state variables are continuous. In this scenario, all variables begin at zero, with the task being completed when the 'Progress' or 'Distance moved' variable reaches its maximum. At each time step, the progress made or distance traveled for each task is incremented by a value drawn from the indicated normal distribution. Tasks with an 'Error recovery' variable include nonterminal failures, as diagrammed in Figure 5.

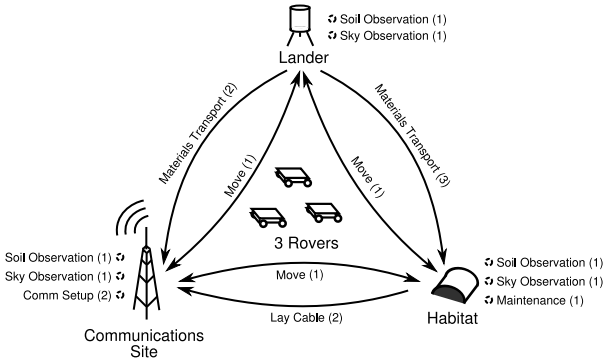

Figure 6: The Lunar Outpost scenario. Numbers in parentheses denote the number of agents needed to perform each task. Tasks during which the agent must remain at a site are denoted with a dashed circle.

opportunistically scheduled when over-runs and under-runs occur during execution.

Experimental Conditions

We evaluated two experimental conditions: baseline and duration prediction. We generated 20 initial schedules using ASPEN, which we have augmented with heuristics that address multi-agent tasks. We executed each of the schedules five times under each condition, resulting in 100 runs each.

Baseline In the baseline case, no duration prediction is performed. Instead, when a task finishes early, its duration on the schedule is instantly changed from the expected to the final value. When a task over-runs, its duration is increased incrementally, until it completes.

At every timestep, ASPEN evaluates whether there are any conflicts in the schedule. If there are, it repairs them: if the conflicts are small in duration (likely resulting from the incremental increase of over-running tasks), it will attempt to right-shift tasks to resolve them. If this fails, ASPEN's general repair algorithm will be invoked until the schedule is once again conflict-free. After all conflicts have been repaired, ten iterations of optimization are performed, with repair of any new conflicts occurring after each. This allows the planner to repair any inefficiencies introduced by the initial repair cycle.

If there are no initial conflicts, ASPEN performs one iteration of a limited optimization algorithm at each timestep. In particular, it checks for agents that are free at the current time and have sufficient time prior to their next task to perform a solo task with non-zero reward. If such an agent can be found, a task is heuristically selected and scheduled.

Duration Prediction The duration prediction condition is identical to the baseline, except that the durations of all currently executing tasks are updated at every timestep, as discussed above. When tasks are predicted to over-run or under-run, this provides opportunities for the optimization algorithm to schedule additional tasks.

Data and Discussion

When using duration prediction, the planner is able to schedule 28.6% more tasks on average than the baseline and achieve a 28.5% greater total reward (Table 2). When unrewarding tasks are excluded, we can see that duration prediction almost exclusively adds tasks with non-zero reward: the number of additional executed tasks and executed rewarded tasks are very similar.

While the average increases are promising, the data is quite noisy, as we can see from the large standard deviations. This is due to the stochastic nature of both execution and ASPEN's heuristic approach to repair and optimiza-

	Reward	Executed Tasks	Rewarded Executed Tasks	Execution Time	Runs
Baseline	1517.80 (847.09)	78.79 (44.68)	57.43 (33.78)	1382.03s (1041.28s)	100
Duration Prediction	1951.15 (833.58)	101.30 (47.41)	78.07 (38.66)	2424.87s (1178.02s)	100
Prediction - Baseline	433.35 (+28.5%)	22.51 (+28.6%)	20.64 (+35.9%)	1042.84s (+75.5%)	—
p-value	0.000075	0.0000067	0.000056	—	—

Table 2: Experimental results. All data is reported as *mean (standard deviation)*. The difference between "Executed Tasks" and "Executed Rewarded Tasks" is the number of zero-reward tasks performed; in this scenario, these consisted of the Move tasks. p-values are the result of a paired T-test comparing the baseline and prediction data.

tion. However, we performed a paired T-test comparing the baseline with duration prediction using the reward, executed tasks, and executed rewarded tasks data. In all three cases, the use of duration prediction resulted in statistically significant increases, at a confidence level of $p = 0.01$. See the last row of Table 2 for specific p-values.

These improvements come at a cost, however: overall planning time increases by 75% on average. We have not yet attempted to optimize our approach, and expect this cost to decrease significantly as research continues.

Future Work

Duration prediction, while useful in its own right, is also a tool that can enable much greater improvements in planning and execution. In particular, we have previously proposed *live task modification* (Sellner & Simmons 2006), in which the composition of teams currently executing tasks may be adjusted in response to the realities of execution. For instance, if two tasks are being performed in parallel, but we predict that one will finish much later than the other, agents should be reassigned to balance the tasks and reduce the overall makespan of the schedule. Live task modification requires duration distributions, rather than simple scalar estimates, to reason about factors such as the likelihood of a transfer being useful.

In reasoning about live task modification, we must predict the effect of transferring an agent. During actual execution, such transfers are not instantaneous, nor deterministic. As a result, we will need to predict the effect on task duration of an additional agent arriving at an uncertain time in the future. Calculating this exactly requires projecting the task's state into the future, estimating the duration distribution at each possible state, and aggregating those distributions to produce a usable estimated distribution. We are currently developing an approach that allows approximate estimation without any knowledge of future states.

We will also examine whether the predicted distributions can be integrated more tightly into the planner. Our current approach represents the duration in ASPEN as the mean of the predicted distribution, but this is clearly suboptimal in some cases. For instance, when the distribution is multimodal, the mean often falls between modes. It may be useful to use the mean of the most likely mode, or modify ASPEN to support a discrete distribution across durations. However, the latter is likely to result in an unacceptable computation / reward tradeoff. In either case, we will continue to maintain duration distributions for use in multi-metric optimization calculations and live task modification.

Finally, we plan to characterize how the effectiveness of duration prediction varies as the amount of available training data is changed. This should allow us to determine a reasonable lower bound on the number of observations necessary to provide a given level of execution-time improvement.

Conclusion

This paper has examined the utility of duration prediction, especially with respect to its use as part of a proactive replanning system. We also have discussed the capabilities enabled by predicting distributions, rather than point estimates, and presented our approach to doing so from relatively sparse training data in a continuous state space. We experimentally evaluated an initial implementation of duration prediction, and found it yielded statistically significant gains of 28.5% to 35.9%, depending on the metric being evaluated. While duration prediction is an improvement in its own right, it is a stepping stone toward more flexible and effective proactive replanning systems that can predict problems during execution and fluidly reallocate agents in response.

References

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

Belker, T.; Hammel, M.; and Hertzberg, J. 2003. 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, 4136–4141.

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

Bookstein, F. L. 1989. Principal warps: Thin plate splines and the decomposition of deformations. *IEEE Transations on Pattern Analysis and Machine Intelligence* 11:567–585.

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

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

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

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

Friedman, J. H. 1991. Multivariate adaptive regression splines (with discussion). *Annals of Statistics* 19:1–141.

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

Kay, S. M. 1993. *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice Hall. chapter 7.

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

Merkwirth, C.; Mauser, H.; Schulz-Gasch, T.; Roche, O.; Stahl, M.; and Lengauer, T. 2004. Ensemble methods for classification in cheminformatics. *Journal of Chemical Information and Modeling* 44(6):1971–1978. DOI: 10.1021/ci049850e.

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

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

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

Sellner, B., and Simmons, R. 2006. Towards proactive replanning for multi-robot teams. In *Proceedings of the 5th International Workshop on Planning and Scheduling in Space 2006*.

Silverman, B. W. 1986. *Density estimation for statistics and data analysis*. London, UK: Chapman and Hall.

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

Vijayakumar, S., and Schaal, S. 2000. Locally weighted projection regression. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, volume 1, 288–293.

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