Duration Prediction for Proactive Replanning

Brennan Sellner Reid Simmons

***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. The addition of *duration prediction* resulted in a 11.7% improvement in average reward. Compared with an omniscient planner, this is 45.0% of the maximum possible improvement.**

I. 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 completion time throughout its execution, a component we refer to as *duration prediction*. Duration prediction estimates a task's remaining run time, given a measurement of its current state. Rather than computing a single estimate of the duration, we estimate a distribution across the possible task durations.

B. Sellner and R. Simmons are with the Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213 bsellner@andrew.cmu.edu, reids@cs.cmu.edu

By predicting a duration distribution, it becomes possible for the planner to engage in several new strategies that are unattainable if we simply compute 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 distributions of the arrival time and the duration of the receiving task. Scalar estimates provide insufficient information to accurately reason about 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.

We need execution data in order to estimate the duration distribution, since most realistic tasks involve stochasticity that cannot be accurately foreseen. This training data is necessarily thin, 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. We have developed a kernel density estimation-based approach to duration prediction that enables the estimation of duration distributions given relatively sparse training data.

We have evaluated our approach using the ASPEN planner [1] and a high-level stochastic execution simulator. Our experimental results indicate that the use of duration prediction increases the total reward over the baseline by a statistically significant degree. The increase amounts to 45.0% of the possible improvement, as measured by building plans with complete prior knowledge of the tasks' final durations.

II. Related Work

A. 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 training data and 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 [2]. 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 [3] 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 [4], locally weighted projection regression [5], and neural networks are capable of approximating functions from high-dimensional input spaces. While all could be utilized for duration prediction, when we applied them to our domain fitting times tended to be long and over-fitting often occurred.

We have selected a modified form of kernel density estimation (KDE) [6] as our prediction method. KDE is a nonparametric method that is able to estimate an arbitrary distribution from training data without making assumptions about the structure of the underlying distribution. KDE is not subject to over-fitting, as it operates directly from the data, but is able to interpolate to a degree between data points. KDE will be discussed in detail in the Approach section.

B. 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, and be able to quickly replan or repair a plan in response to feedback from the executive. While a number of planners meet these requirements (e.g. EUROPA [7] and IxTeT [8]), we have used ASPEN [1] 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 [9] and ATLANTIS [10] generally have planning/executive ties that are too loose to support duration prediction. LAAS [11] and CLARAty [12] 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 [13] and CLEaR [14]. CASPER uses the ASPEN planner and a simple executive, while CLEaR extends CASPER with a TDL-based [15] 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 [16] makes use of EUROPA and also is flexible enough to support our work.

III. Approach

Duration prediction allows the planner to recognize future scheduling problems and opportunities in time to address or take advantage of them. Specifically, it allows the prediction of two classes of execution anomalies: over-runs and under-runs.

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 idling 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.

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. Fig. 1 depicts a canonical example of an under-running task. Here, 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 Fig. 1(a).

If task A completes early (Fig. 1(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 (Fig. 1(c)). However, this is inefficient, as BPrep may be executed in parallel with A. If the 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, allowing B to be scheduled immediately after A, and BPrep to be executed entirely in parallel with task A (Fig. 1(d)). By providing the planner with forewarning

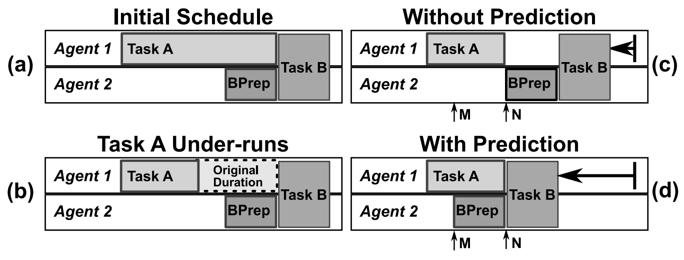


Fig. 1. Duration prediction allows the planner to start setup tasks early when a preceding task is predicted to under-run.

of under-runs and over-runs such as this, duration prediction enables the execution of more efficient schedules.

In addition to enabling more efficient execution, duration prediction 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.

A. Prediction Method

We use a form of kernel density estimation (KDE) [6] to predict duration distributions. KDE is a nonparametric method related to histograms that is used to estimate an arbitrary distribution from training data without making *a priori* assumptions about the form of the underlying distribution. 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 (or *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 utilizes a smoother function, such as the normal distribution. An example of this is depicted in Fig. 2. The five observations are denoted with circles. At each observation, a normal kernel is centered (dashed lines), and summed to yield the estimated distribution (solid line). The selection of the shape and bandwidth of the kernel affects 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 the experiments reported here, we have found that a bandwidth of 2.5 time units yields reasonable results.

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: $f(x) = \frac{1}{n}\sum_{i=1}^{n} K\left(x - x_i, h\right)$, 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 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 $\sum_{i=1}^{n} w_i = 1$. The density function is nearly identical to the canonical KDE

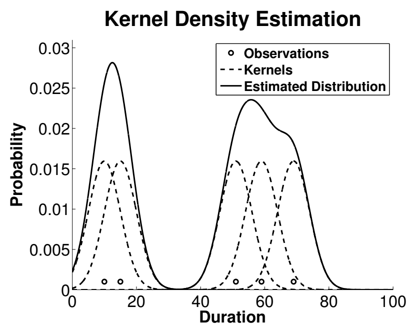


Fig. 2. A simple example of kernel density estimation. Kernels (dashed lines) are centered at each of the five duration observations (plotted as 'o's), then the kernels are summed to build the estimated distribution (solid line).

approach, simply replacing the uniform weighting with the observation-specific weight: $f(x) = \sum_{i=1}^{n} w_i K\left(x - x_i, h\right)$

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 uses to build the duration distribution. We do so by applying a *query kernel* along each dimension of the state space (Fig. 3). 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.

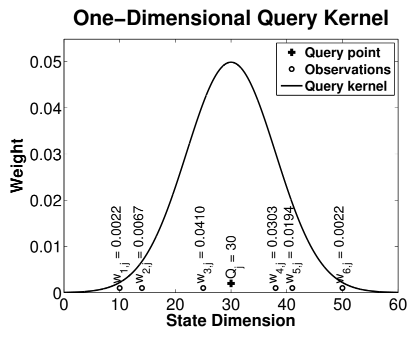


Fig. 3. The query point (the current value of this dimension's state variable) is denoted with a '+', candidate observations with 'o's, and the query kernel as the solid curve. The weight of an observation i for this dimension, $w_{i,j}$, is the likelihood that the observation would be drawn randomly 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: $w_{i,j} = K\left(x_{i,j} - Q_j, h_j\right)$. To limit computation, we consider observations only where $w_{i,j} > \epsilon$.

This weight calculation is performed for all dimensions, resulting in a set of weights $w_{i,j}$ for the 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 $x_{i,d+1}$.

B. Planner Integration

We have integrated duration prediction with the repair-based ASPEN planner [1] and CASPER executive [13]. 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 (potentially empty) plan, 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.

C. Execution Modeling

Task execution is stochastically modeled using a representation similar to Augmented Transition Networks [17]. 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 non-terminal 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. Fig. 4 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. Models such as this are used both during execution and to provide the training data needed to build the KDEs used by duration prediction.

IV. Results

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

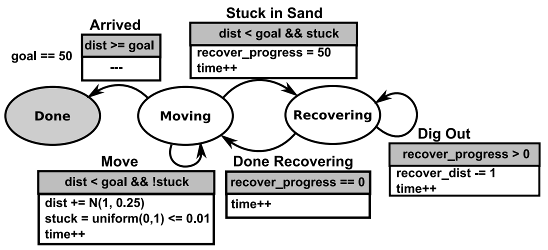


Fig. 4. The stochastic simulation model for *Move*. This is used to simulate execution, and provides the state that is used for duration prediction during execution. In addition, this model is used offline to generate the observations used by the KDE-based predictor.

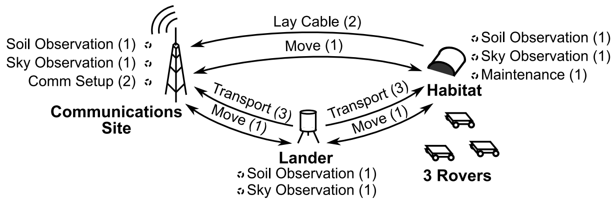


Fig. 5. The Lunar Outpost scenario. Numbers in parentheses denote the number of agents needed to perform each task. Tasks during which the agent must remain at a site are denoted with a dashed circle.

A. Scenario

The scenario represents a subset of the construction of a lunar outpost, and includes three agents, three sites, and eight types of tasks (Fig. 5, Table I). Agents are assumed to be homogeneous, and can participate in only one task at a time. Each task has an associated reward, may be constrained to occur at a given site, and/or may involve the moving of agents from one site to another. Different tasks require different numbers of agents. 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. During execution, ASPEN evaluates at every timestep whether there are any conflicts in the schedule. If there are, it repairs the conflicts by first right-shifting tasks. If this fails, ASPEN's general repair algorithm is invoked until the schedule is 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 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.

B. Experimental Conditions

We evaluated three experimental conditions: a baseline, an "oracle" case, 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 the baseline condition and fifty

TABLE I
SCENARIO TASKS AND RELEVANT STATISTICS.

Task	Reward	Agents Needed	Location	State Variables	Progress Per Timestep	Duration from Start (μ, σ)
Move	0	1	Start: Anywhere End: Anywhere	Distance moved: [0,50] Recovery progress: [0,50]	$N(1, 0.25)$	59.12 (39.59)
Sky Observation	15	1	Anywhere	Progress: [0,1]	$N(0.05, 0.01)$	18.16 (4.50)
Soil Observation	15	1	Anywhere	Progress: [0,1]	$N(0.025, 0.01)$	33.11 (7.21)
Habitat Maintenance	30	1	Habitat	Progress: [0,1] Recovery progress: [0,20]	$N(0.025, 0.01)$	41.65 (15.75)
Materials: Lander $\rightarrow$ Habitat	300	3	Start: Lander End: Habitat	Distance moved: [0,50] Recovery progress: [0,50]	$N(0.5, 0.1)$	113.72 (27.82)
Lay Cable	120	2	Start: Habitat End: Comm	Distance moved: [0,50] Recovery progress: [0,50]	$N(0.3, 0.25)$	205.53 (45.64)
Materials: Lander $\rightarrow$ Comm	100	2	Start: Lander End: Comm	Distance moved: [0,50] Recovery progress: [0,100]	$N(2, 0.25)$	34.06 (70.71)
Comm Setup	50	2	Comm	Progress: [0,1] Recovery progress: [0,200]	$N(0.05, 0.01)$	49.54 (83.80)

times under the duration prediction condition, resulting in 100 and 1000 runs, respectively. In the oracle condition, the planner was provided with complete foreknowledge of the task durations, making execution extraneous. Instead, each of the 20 schedules was stochastically optimized five times, again yielding 100 data points.

1) 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.

2) 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 in Section IV-A. When tasks are predicted to over-run or under-run, this provides opportunities for the optimization algorithm to schedule additional tasks.

3) Oracle: In the oracle condition, the planner is provided with the precise durations of each task, obviating any need for prediction. The specific durations were generated by simulating the task once for each task instance. As a result, the actual task durations were stochastic across runs, but the durations for each run were known by the planner at plan time. Under this condition, the planner updated an initial schedule (built using average task durations) with the actual durations, repaired any resulting conflicts, then performed 5000 iterations of the same optimization routines as used during execution in the other two experimental conditions. This was done five times for each of the 20 initial schedules.

C. Data and Discussion

The results of our experiment are detailed in Table II, where the data are reported as *mean (standard deviation)* or *difference (percent difference)*. The data is the difference between the initial and final schedules – this removes the variation due to differences in the initial schedules. "Executed Tasks" includes zero-reward tasks, such as *Move*, while "Rewarded Executed Tasks" includes only rewarding tasks (see Table I). The reported planning time is the time needed to perform all repairs and optimizations during the execution or construction of a schedule.

When using duration prediction, the planner is able to schedule 25.2% more tasks on average than the baseline and achieve a 11.7% greater total reward (Table II, row 4). In addition to scheduling more rewarding tasks, some unrewarding tasks were replaced with rewarding ones (the increase in rewarded tasks is greater than the increase in executed tasks).

Under the oracle condition, which represents the best that the planner could accomplish with the provided heuristics, 6.6% more tasks were scheduled than the baseline, yielding 25.9% more reward (Table II, row 5). This shows that by predicting task durations, the planner was able to achieve 45.0% of the possible improvement (Table II, row 6). Note that under the oracle condition significantly fewer tasks were scheduled, yet more reward was garnered: with the provided foreknowledge, more of the long, high-reward tasks were scheduled. In contrast, when using prediction, many smaller low-reward tasks were added as small opportunities presented themselves. The specific results will vary according to the composition of the scenario, but the gains due to prediction are a function of how early over- and under-runs can be predicted, the duration of tasks available for addition, and the reward per unit time of the available tasks.

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 optimization. In particular, note the large standard deviations in expected duration for tasks such as *Comm Setup* and *Materials: Lander $\rightarrow$ Comm* (Table I). We performed a repeated measures ANOVA with the initial schedule as the repeated sample to compare the baseline, duration prediction, and oracle results. There were statistically significant differences between all combinations (at a confidence level of $p = 0.0001$), except for rewarded tasks between the baseline and oracle cases. As can be seen from Table II, the number of rewarded executed tasks was very similar between these conditions, although which tasks were executed varied significantly, as can be seen by the difference in reward.

These improvements come at a cost, however: overall planning time increases by 37.2% on average when using

TABLE II
EXPERIMENTAL RESULTS.

	Reward [a]	**Executed Tasks** [a]	**Rewarded Executed Tasks** [a]	**Planning Time**	**Runs**
1. **Baseline**	2041.73 (475.53)	122.96 (28.26)	118.18 (31.24)	20.14s (7.79s)	100
2. **Oracle**	2571.10 (419.59)	131.10 (114.32)	114.32 (19.71)	66.16s (11.18s)	100
3. **Duration Prediction** [b]	2280.17 (433.25)	153.89 (27.73)	161.80 (32.22)	27.63s (17.57s)	1000
4. **PB = Prediction - Baseline**	238.44 (+11.7%)	30.93 (+25.2%)	43.62 (+36.9%)	7.49 (+37.2%)	—
5. **OB = Oracle - Baseline**	529.37 (+25.9%)	8.14 (+6.6%)	-3.86 (-3.3%)	46.02s (+228.5%)	—
6. **OB - PB** ($\frac{PB}{OB}$)	290.93 (45.0%)	-22.79 (380.0%)	-47.48 (—)	38.53s (16.3%)	—

[a] Values are relative to the initial schedules. [b] Using 32 training runs per task.

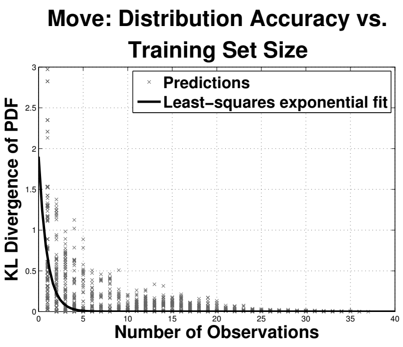


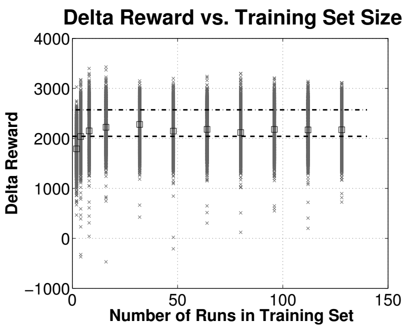
Fig. 6. As the number of observations used to form the duration distribution increases, the resulting distribution becomes more accurate. The Y axis indicates the K-L divergence from a distribution built from all available data at the relevant query point.

prediction (Table II, row 4). This is due to an increased number of repair and optimization attempts made possible by the predictions, as well as the cost of prediction itself. In order to evaluate only the effects of duration prediction, execution was suspended during planning and optimization. When operating in the real world, the number of optimization passes and what portion of the schedule is optimized can be adjusted to allow continuous execution. ASPEN includes the concept of a "commitment window", which is a window of time, starting with the current time, in which the planner will not change the schedule. This gives ASPEN a known "lead time" in which repair and optimization calculations can be performed without affecting execution.

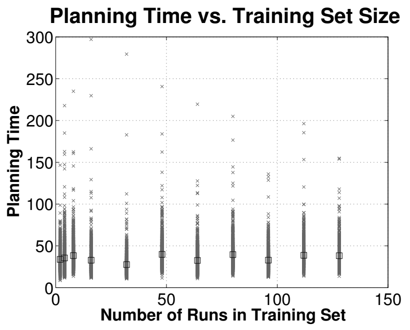
D. Duration Prediction Characterization

In addition to evaluating the utility of duration prediction, we characterized its accuracy as a function of the amount of available training data. To do so, we built a corpus of data for each of the eight tasks, consisting of 512 runs from the initial state. This yielded between 11,021 and 177,795 observations per task, depending on the task's inherent length and stochasticity. We then created 17 sample sets for each task, each with an ever larger portion of the corpus, beginning with 2 runs and proceeding to 512 runs in increments of 32.

We built a test duration distribution from each sample set at every point in a finely-spaced grid spanning the state space of each task, and compared them with the corresponding distribution constructed from the 512-run reference set. Points with no supporting data were discarded. This comparison was made by computing the Kullback-Leibler (K-L) divergence [18] between the test and reference distributions.



(a)



(b)

Fig. 7. Reward achieved and planning time needed as functions of the number of runs in the training set for each task. Each point corresponds to an execution of a schedule, with the mean of each size of training set plotted as a square. In 7(a), the average delta reward for the baseline case is denoted with a dashed line, while the oracle condition is plotted as a dashed-dotted line.

The divergence value increases as the distributions being compared are more different, and identical distributions have a divergence of zero.

Because a run does not provide data spread evenly across the state space, Fig. 6 plots the number of kernels (observations) used to construct a prediction against the resulting K-L divergence for all test points from all sample sets for the Move task. In general, test points from larger sample sets will have more observations, but the relationship depends heavily on the task's structure: a task may spend most of its time in one portion of the state space, yielding few observations in the outlying state space, even with large training sets. The fitted curve is the least-squares fit of $y = a*exp(b*x)$ to the available data. The exponential distribution also fits the data from the other seven tasks, although a and b will of course vary. As can be seen from Fig. 6, little utility is gained from having more than 5 data points in the vicinity of the query point. Ideally, 5 observations would be within the query kernel bandwidth of every point in the state space. Given a kernel bandwidth and knowledge of the task's structure, this can provide guidance as to how many training runs are useful for a particular scenario.

We then repeated the duration prediction portion of our initial experiment while varying the amount of available training data, from 2 to 128 runs per task. A run consists of the data points generated by a single simulated execution of the task. As can be seen in Fig. 7(a), reward increases as the amount of training data increases, asymptoting at roughly 32 runs of training data per task, although there is significant variation as the training set continues to expand. The first experiment (Table II) utilized data sets equivalent to the 32-run sample set. The prediction condition improves significantly over the baseline as long as more than 4 training runs are available: the difference between the baseline and every prediction experiment (except for the 4-run case) is statistically significant at a confidence level of 0.0001, according to a repeated measures ANOVA.

Fig. 7(b) shows that planning time increases very slightly, if at all, with the amount of training data. While we use k-d trees to quickly perform queries, prediction time will always increase as the size of the training corpus increases. This is offset by a reduction in the amount of repair needed, due to the more accurate predictions.

V. 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* [19], 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.

VI. Conclusion

This paper has examined the utility of duration prediction, especially with respect to its use as part of a proactive replanning system. We have discussed the capabilities enabled by predicting distributions, rather than scalars, and presented our approach to doing so from relatively sparse training data in a continuous state space. We experimentally evaluated an implementation, and found it achieved a statistically significant 45.0% of the maximum possible improvement, when compared with an omniscient planner. In addition, we established a relationship between the amount of training data and both prediction accuracy and proactive replanning performance. While duration prediction is an improvement in its own right, it is also a stepping stone towards more flexible and effective proactive replanning systems that can predict problems during execution and fluidly reallocate agents in response.

References

[1] 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.

[2] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice Hall, 1993, ch. 7.

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

[4] J. H. Friedman, "Multivariate adaptive regression splines (with discussion)," *Annals of Statistics*, vol. 19, pp. 1–141, 1991.

[5] S. Vijayakumar and S. Schaal, "Locally weighted projection regression," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, vol. 1, 2000, pp. 288–293.

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

[7] J. Frank and A. Jónsson, "Constraint-based attribute and interval planning," *Journal of Constraints, Special Issue on Constraints and Planning*, vol. 8, no. 4, October 2003.

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

[9] 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*, vol. 9, no. 2-3, pp. 237–256, 1997.

[10] 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.

[11] 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*, vol. 17, no. 4, 1998.

[12] I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and W. S. Kim, "Claraty: An architecture for reusable robotic software," in *Proceedings of the SPIE Aerosense Conference*, Orlando, Florida, April 2003.

[13] 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.

[14] 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*, 2001.

[15] 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.

[16] N. 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. [Online]. Available: http://citeseer.ist.psu.edu/593897.html

[17] W. A. Woods, "Transition network grammars for natual language analysis," *CACM 13*, vol. 10, pp. 591–606, October 1970.

[18] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 79–86, 1951.

[19] B. Sellner and R. Simmons, "Towards proactive replanning for multi-robot teams," in *Proceedings of the 5th International Workshop on Planning and Scheduling in Space 2006*, Baltimore, MD, October 2006.