Combining Stochastic Task Models with Reinforcement Learning for Dynamic Scheduling

Malcolm J. A. Strens
QinetiQ Limited
Cody Technology Park
Ively Rd, Farnborough, GU14 0LX, U.K.
mjstrens@QinetiQ.com

Abstract

We view dynamic scheduling as a sequential decision problem. Firstly, we introduce a generalized planning operator, the stochastic task model (STM), which predicts the effects of executing a particular task on *state*, *time* and *reward* using a general procedural format (pure stochastic function). Secondly, we show that effective planning under uncertainty can be obtained by combining adaptive horizon stochastic planning with reinforcement learning (RL) in a hybrid system. The bene£ts of the hybrid approach are evaluated using a repeatable job shop scheduling task.

Introduction

In logistics and robotics applications the requirement is often to allocate assets (robots or vehicles) to tasks that are spatially distributed. In production scheduling there is a need to allocate assets (human and machine) to the tasks (customer orders) in order to achieve timeliness, ef£ciency and cost objectives. Simpli£ed models for these systems have been de£ned for planning and AI, such as *job shop scheduling* and *multi-robot task allocation* (MRTA). However these simpli£ed problem descriptions do not necessarily capture intrinsic characteristics of the problem such as stochastic performance and a dynamic stream of new tasks. Regardless of the domain, we can identify two quite distinct sources of uncertainty.

The most well-understood form of uncertainty is that which arises in the execution of tasks. For example, the time taken for a robot to traverse terrain may be intrinsically unpredictable, often because we do not have accurate and reliable models for the robot's dynamics and the terrain itself. There is some statistical distribution over task execution times rather than a deterministic outcome. The same uncertainty extends to the *state* that results from execution of a task as well as its duration. In general, the possible outcomes reside in a high-dimensional continuous state or belief space.

The second (and less-studied) form of uncertainty is the stream of tasks that enter the system. This is often ignored completely in conventional approaches that formulate a *static optimization problem* to address only the currently known tasks. In real applications, the stream of tasks that enter a system are highly unpredictable because they are the result of human decision-making processes that have taken place in a much broader context (than the planning system's domain knowledge).

In a system where very high priority tasks may occur at any time, the *readiness* of the system to service these (as yet unseen) tasks must be accounted for. However, this is not possible if we do not acknowledge the dynamic nature of the domain, and estimate/learn the effects of our decisions on future returns from unseen tasks. For example, in a production environment, it may be best to keep one machine free in readiness for high priority customer orders, even at the cost of incurring a timeliness penalty on low priority orders. If resources (fuel, energy, etc) are consumed in the process of executing tasks, the cost of our decisions (in terms of diminished future returns from unseen tasks) must be estimated/learnt, to adjust the value of a proposed plan. For example, in MRTA it may be better to prefer assigning a robot with large energy reserves to new task, even at the cost of taking more time to complete it.

Hybrid approach

We propose a solution that combines a very general stochastic planning process with value-function RL. This hybrid solution exploits the strengths of planning (specifying domain structure and resolving complex constraints) and the complementary strengths of RL (estimating the long term effects of actions), in a single architecture founded in sequential decision theory. Boutilier *et al.* (Boutilier, Dean, & Hanks 1999) survey existing work in this topic area, which links operations research (OR), RL and planning.

First, we de£ne a *Markov Decision Process*, a discrete-time model for the stochastic evolution of a system's state, under control of an external agent's actions. Formally, an MDP is given by $< X, A, T, R >$ where X is a set of states and A a set of actions. T is a stochastic transition function de£ning the likelihood the next state will be $x' \in X$ given current state $x \in X$ and action $a \in A$: $P_T(x'|x, a)$. R is a stochastic reward function de£ning the likelihood the immediate reward will be $r \in \mathbb{R}$ given current state $x \in X$ and action $a \in A$: $P_R(r|x, a)$. Combining an MDP with a deterministic control policy $\pi(x)$, that generates an action for every state, gives a closed system. A useful measure of policy performance, starting at a particular state, is the expected value of the *discounted return*: $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$. $\gamma < 1$ is a discount factor that ensures (slightly) more weight is given to rewards received sooner. For a given policy π and starting state x, we write $V_\pi(x)$ for this expectation. *RL* attempts to £nd the optimal policy for a MDP with unknown transition and reward functions by trial-and-error experience (Sutton & Barto 1998).

Dynamic Scheduling with Stochastic Tasks

We focus on the concepts of *task* and *asset* (e.g. machine, robot, human resource). Speci£cally, a plan must consist of an ordered assignment of the currently known tasks to assets. We assume that a task is an object that once speci£ed (or created by a higher-level planning process) must be allocated to at most one asset. (This is not as great a restriction as it may seem; more complex goals can be decomposed into a coupled sequences of tasks.) The set of possible plans is the discrete set of possible ordered assignments, augmented by *allocation parameters* associated with each assignment.

This plan structure forms a basis on which to attempt to *factor* the "full" MDP representing the multi-agent, multi-task system into many smaller sub-problems that can be individually solved. In a *weakly-coupled* system, once the allocation has been made the tasks essentially become independent in terms of rewards received and end states. For example, if asset A is assigned to task 3 and asset B to task 5, the effectiveness of A in task 3 is assumed to be independent of the individual steps taken by B in performing task 5. This suggests a hierarchical decomposition in which one level (allocation of assets to tasks) is "globally" aware, but lower level planning (execution of individual tasks) can be achieved in a *local* context (a particular asset and particular task).

Stochastic Task Model

In this problem structure, we introduce the notion of a stochastic task model (STM). This is a predictive model for the outcome of executing a particular task with a particular asset, from a particular starting state. The outcome is a sample for the end state (after execution of that task), the time taken, and the (discounted) reward. We assume that tasks are divided into a discrete set of task types, indexed by k. For each k, the STM consists of a pair of functions $< S_k, P_k >$. The £rst of these functions is analogous to the preconditions of a planning operator, indicating whether the STM is applicable in any particular setting:

$$S_k : (t, x_j, \beta, \alpha) \rightarrow true|false$$

where t is the current time (discrete or continuous), vector x_j is the current state of the chosen asset, β is a vector of parameters describing the particular task instance and α is a vector of allocation parameters (decided when the task was added to the plan). If S_k returns true, then the second function samples a new state (x'_j), duration τ, and discounted return r:

$$P_k(x'_j, \tau, r | x_j, \beta, \alpha)$$

Note that the overall state of the system is, as a minimum, the list of individual asset states $(x_1, \ldots, x_m)$, but may contain other state information. A more general formulation of an STM allows modi£cations to the full joint state x, rather than only the x_j component. Therefore the weak-coupling assumption can be relaxed.

Scoring a short-term plan

Suppose we have a set of candidate plans for comparison, at some time t_0. We wish to decide which is best according to the *expected discounted return* criterion. The part of this return resulting from the *known* tasks can be predicted by executing task models. To score a particular plan, we apply the STMs for the sequence of allocated tasks, keeping track of the asset's sampled state, time and discounted return $< x_j, t_j, d_j >$. (Initially, $t_j = t_0$ and $d_j = 0$.) We repeat the following sequence of steps:

1. Find an asset j for which the preconditions of its £rst unexecuted task (with known parameters α and β) are met. i.e. $S_k(t_j, x_j, \beta, \alpha) =$ true
2. Draw $< x'_j, \tau, r >$ from $P_k(x'_j, \tau, r | x_j, \beta, \alpha)$
3. Update time $t_j \leftarrow t_j + \tau$
4. Update plan state $x_j \leftarrow x'_j$
5. Accumulate discounted return $d_j \leftarrow d_j + \gamma^{t_j - t_0} r$

If the £rst step fails, the plan is invalid and is removed from the list of candidates; otherwise, all tasks are evaluated and the sampled short-term plan score (from this one "sweep") is $\sum_j d_j$. Because sweeping through a plan in this way generates d_j stochastically, we can represent it by a random variable D_j (for each asset j). A quantity of interest is the expected discounted return $E[\sum_j D_j]$, which can be estimated by averaging the values of d_j realized from multiple sweeps. This provides us with a mechanism for comparing candidate plans, even where the execution of tasks in the plan is a non-deterministic process.

Planning horizon

This process of comparing plans using returns predicted by task models will be termed "short-term planning" because it maximizes the discounted return from the known tasks, rather than future (unseen) tasks. Furthermore, there may be no bene£t in planning ahead beyond some *planning horizon*: a time beyond which uncertainty or computational costs will have grown so as to make planning worthless. If this is at some constant duration from the current time, it is termed a *receding horizon* approach. In scoring a plan, no further tasks are scored once t_j passes the horizon. We have shown how to obtain a score representing the expected discounted return of a candidate plan, but limited to rewards received from *known* tasks, realized *before* the planning horizon. We now consider formulations for RL of a value function representing returns that are realized from unseen future tasks, or beyond the planning horizon.

Our main result is applicable when (i) it is computationally feasible to score a plan containing all the (known) tasks; and (ii) the sequence of tasks assigned to each asset must be

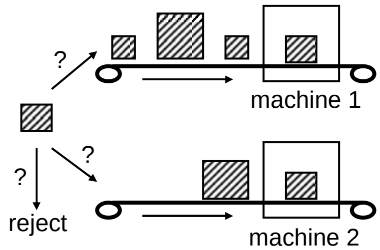

Figure 1: A stochastic job shop scheduling problem.

executed in the order de£ned by the plan before any new tasks that subsequently enter the system. These assumptions ensure that future returns are conditionally independent of the known tasks, given the time-stamped plan end-states ($< x_j, t_j >$) of all m assets. Therefore the expected discounted future return can be expressed in terms of only these time-stamped end states, ignoring the known tasks. In the next section we will show how a value function V with these inputs can be acquired by batch learning, and used in policy iteration. Given the outcome $< x_1, t_1, d_1, \ldots, x_m, t_m, d_m >$ from a single scoring sweep of a candidate plan, we can sample a total score by adding the value function to the plan return:

$$v = (\sum_j d_j) + V(x_1, t_1 - t_0, \ldots, x_m, t_m - t_0)$$

The expectation of this quantity (estimated by averaging for repeated sweeps of the plan) is the expected discounted return for known and future tasks. It will yield an optimal decision policy if V is accurately represented. We call this approach *adaptive horizon* because the end times $\{t_m\}$ can *vary* across assets, according to when that asset will become free for new tasks.

In contrast, a *receding horizon* approach (Strens & Windelinckx 2005) de£nes the value function in terms of the joint state of all assets $< x_1, x_2, \ldots, x_m >$ at the planning horizon. It cannot account for returns from *unseen* tasks that are actually obtained before the planning horizon (as a result of a re-plan). While this approximation implies sub-optimality, a receding horizon can be very helpful to achieve bounded real-time performance, and a simpler value function is achieved because all parts of the state have the same time-stamp. Both methods yield decision-making behavior that accounts for not only short-term rewards, but also readiness for new tasks, including long term considerations such as consumption of resources.

Job shop scheduling evaluation

We introduce a dynamic scheduling task in which the arrivals process is highly uncertain, task execution is stochastic, and readiness for high priority tasks is essential to achieve effective performance. This enables us to show the bene£ts of a sequential decision theoretic approach. We will compare three solutions: a heuristic policy that assigns each customer request to the shortest queue, a planner that makes use of stochastic task models, and a hybrid approach that combines this planner with RL of a value function, capturing readiness for future (unseen) tasks. A factory has 2 identical machines, each with a £rst-in-£rst-out queue of tasks. Each queue can hold a maximum of 10 tasks, including the one being processed. Each task that enters the system (indexed in order by k) has a size s_k which determines the *expected number* of discrete time steps the task will take for the machine to process. However, the actual time taken is geometrically distributed with parameter $p_k \equiv 1/s_k$:

$$P(t_k) = p_k(1 - p_k)^{t_k - 1}$$

At each discrete time t, a new customer request arrives with probability $2/5$, and has size drawn uniformly from the set $\{1, 2, \ldots, 9\}$. Therefore if every task is successfully processed, the two machines will be 100% utilized. The reward for successfully completing a task is 1, unless the task spends more than 10 time steps in total (waiting in a queue and being processed) in which case the reward is 0. For example, a task that arrives at time 0 must complete before or during time step 9 to obtain a reward. Once a task is in a queue, it cannot be abandoned, even if it will receive no reward.

For each new task, a decision must be made instantaneously on its arrival (Figure 1). There are three choices: reject the task (with reward 0) or add it to the queue for one or other machine. The baseline (heuristic) policy assigns each new task to the machine that has the smallest queue, in terms of the total of the sizes of all tasks on that queue (including the one currently being processed). This measure of queue-length is identical to the *expected* time until the machine becomes available. Therefore this policy yields the same result as would a deterministic planning process that works with mean durations to minimize waiting times. (Queues which are full are not considered in the comparison, and if both queues are full the task is rejected.)

A single stochastic task model is used repeatedly in evaluating proposed plans (allocations of tasks to machines). It is executed to predict the time taken to process a particular task, once it reaches a machine. For a task of size s_k, it draws t_k from the geometric distribution with parameter $p_k = 1/s_k$. It then generates a reward of 1 or 0 according to whether the task has been completed by its deadline, speci£ed as a task parameter. To sample the outcome for a particular queue of tasks, the STM is evaluated sequentially for all tasks in the queue, to provide a (sampled) duration at which the machine becomes free for new tasks. In this process, the discounted rewards are also accumulated. Therefore the £nal outcome is a sample of time taken and discounted return. The discounted return is used directly for scoring the plan, but the sampled duration is used as an input to the value function (when using RL). On arrival of a new task k we have to decide whether to assign this to machine 1, assign it to machine 2, or to reject it. Suppose the existing queues (lists of tasks) are Q_1 and Q_2. We therefore have three candidate plans:

1. Place task k on queue for machine 1: $\Pi_1 \equiv (k :: Q_1, Q_2)$
2. Place task k on queue for machine 2: $\Pi_2 \equiv (Q_1, k :: Q_2)$
3. Reject task k: $\Pi_3 \equiv (Q_1, Q_2)$

The scoring procedure is then applied: sequential application of the STM for each queue (j) in plan Π_i, yields a sampled discounted return d_j (for rewards received from known tasks) and sampled duration until the machine becomes free f_j. Repeating this process to obtain a large sample (size 32) allows us to obtain an estimate of the expected discounted return for that plan. The stochastic planning algorithm then makes a decision (from the 3 options) by selecting the plan with the highest estimated discounted return.

Value function

Suppose that we can learn a *value function* $V(f_1, f_2)$ that accounts for the expected discounted returns for future (*unseen*) tasks, given that machine j will become free after a duration f_j. Then the value of a candidate plan is the sum of the expected discounted returns for known tasks and the expected discounted return for unseen tasks given by application of V, (assuming a £xed policy):

$$E[D_1] + E[D_2] + E[V(F_1, F_2)]$$

where D_1, D_2, F_1 and F_2 are random variables representing outcomes of the above plan scoring process (generating d_1, d_2, f_1 and f_2 respectively). The expectations are effectively taken over the geometric distributions in the STMs. Using this criterion for decision-making allows us to account not only for the known tasks, but how ready the machines are *likely* to be for future unseen tasks. In practice, we draw a sample (size 32) of tuples $< d_1, d_2, f_1, f_2 >$ and the expectation is approximated by an average over this sample. A major advantage of this decomposition of the discounted return is that future returns are conditionally independent of the known tasks, given f_1 and f_2. Therefore V has a much simpler (2-dimensional form) than that which would be required for a RL-only approach, in which V would depend on all known tasks.

We use an approach analogous to policy iteration RL to obtain V. The difference from conventional policy iteration is that our policy is expressed implicitly through the combination of a short-term planning process with a value function. Batch learning took place at the end of each epoch (100,000 steps). For the £rst epoch, decisions are not affected by the value function and so average performance is the same as for a planning-only approach. Subsequently the learnt value function from each epoch is used for decision-making in the next epoch (as part of the plan scoring process). Therefore from the second epoch onward, decisions are capable of taking account of the impact of task allocation decisions on readiness for unseen tasks. The policy iteration process typically converges very rapidly and shows no further improvement after 4 epochs. A full description of the algorithm is beyond the scope of this paper and does not prevent the reader from repeating the results: any method that acquires the true value function through on-line experience is acceptable.

Results

Table 1 summarizes performance of the 3 algorithms, recorded over 16 repetitions. We chose the discount factor $\gamma = 0.99$. The heuristic policy (equivalent to a deterministic planner that minimizes waiting time) provides 21% of available rewards. We note that it has the lowest rejection rate, because it only rejects a new task when both queues are full. The resulting poor performance is caused by long queues which make most tasks exceed their deadline. In comparison the stochastic planner achieves 37% of available rewards. It achieves this by choosing rationally between available actions based on the STM predictions of discounted return for the known tasks. This predictive insight allows it to reject tasks that are not promising when they arrive in the system, avoiding long queues from becoming established.

Algorithm	**Performance**	**Reject**
Deterministic planner	20.9±0.3%	5%
Stochastic planner	37.2±0.3%	8%
Stoch. planner + RL (epoch 4)	62.9±0.2%	22%

Table 1: Job shop scheduling performance comparison (with standard errors).

RL of a value function has a major impact on performance, improving it to 63%. Taking into account the impact on future returns of the candidate plans provides it with a strategic advantage, and completely changes decision behavior. We observe a very high rejection rate (22%), but of the remaining 78% of tasks, most (81%) are successfully processed within the time limit. Therefore there is very clear evidence that it is bene£cial to take account of the likely impact of current plans on future returns, using a learnt value function.

Conclusion

We have shown that dynamic scheduling can be addressed by combining a stochastic planning approach with a value function acquired by RL. The planning element makes use of stochastic tasks models that sample (or predict) the outcome of executing a particular task with a particular asset. The learnt value function is added to the score of each plan, to take account of the long-term advantage or disadvantage of the plan's end state. This can express the readiness of the system for new tasks, and penalize the consumption of resources. Experimental results from a simple, repeatable job shop scheduling example clearly show the bene£ts of both elements, compared with a heuristic strategy.

References

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Arti£cial Intelligence Research* 11:1–94.

Strens, M. J. A., and Windelinckx, N. 2005. Combining planning with reinforcement learning for multi-robot task allocation. In Kudenko, D.; Kazakov, D.; and Alonso, E., eds., *Adaptive Agents and Multi-Agent Systems II*, volume 3394. Springer-Verlag. 260–274.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning*. Cambridge, MA: MIT Press.