The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Allocating Planning Effort When Actions Expire

Shahaf S. Shperberg,[1] Andrew Coles,[2] Bence Cserna,[3]
Erez Karpas,[4] Wheeler Ruml,[3] Solomon E. Shimony[1,5]
[1]Ben-Gurion University, Israel; [2]King's College London, UK;
[3]University of New Hampshire, USA; [4]Technion, Israel; [5]UMass Lowell, USA
shperbsh@post.bgu.ac.il, andrew.coles@kcl.ac.uk, {bence,ruml}@cs.unh.edu,
karpase@technion.ac.il, shimony@cs.bgu.ac.il

Abstract

Making plans that depend on external events can be tricky. For example, an agent considering a partial plan that involves taking a bus must recognize that this partial plan is only viable if completed and selected for execution in time for the agent to arrive at the bus stop. This setting raises the thorny problem of allocating the agent's planning effort across multiple open search nodes, each of which has an expiration time and an expected completion effort in addition to the usual estimated plan cost. This paper formalizes this metareasoning problem, studies its theoretical properties, and presents several algorithms for solving it. Our theoretical results include a surprising connection to job scheduling, as well as to deliberation scheduling in time-dependent planning. Our empirical results indicate that our algorithms are effective in practice. This work advances our understanding of how heuristic search planners might address realistic problem settings.

Introduction

Agents that plan and act in the real world must deal with the fact that time passes as they are planning. For example, an agent that needs to get to the airport may have two options: take a taxi, or take a bus. Each of these options can be thought of as a *partial plan* to be elaborated into a complete plan before execution can start. Clearly, the agent's planner should only elaborate the partial plan that involves taking the bus if it can be elaborated into a complete plan before the bus leaves. Furthermore, consider a second example. When faced with two partial plans that are each estimated to require five minutes of computation to elaborate into complete plans, if only six minutes remain until they both expire, then we would want the planner to allocate all of its remaining planning effort to one of them, rather than to fail on both.

Cashmore et al. (2018) recognized the problem of node expiration in the context of temporal planning with timed initial literals (TIL) (Cresswell and Coddington 2003; Edelkamp and Hoffmann 2004), where the TILs occur at times that are relative to when *planning* starts, rather than to when execution starts. However, their approach to addressing it is relatively superficial in that, after estimating the latest time when execution can start for each search node, this information is used merely to prune nodes that become infeasible. Such a planner, while handling the first example given above, can fail in our second example. In this paper, we investigate the problem more deeply, explicitly using rational metareasoning (Russell and Wefald 1991) to choose which node to expand. We formalize this metareasoning problem, which we call *Allocating Effort when Actions Expire* (AE2), as an MDP, allowing us to define the optimal solution in a manner similar to Hansen and Zilberstein (2001). We establish close connections between AE2, deliberation scheduling in time-dependent planning (Boddy and Dean 1994), and a seemingly unrelated problem in job scheduling with deadlines (Yedidsion 2012). We describe several efficient ways of solving the metareasoning problem, although not necessarily optimally, and evaluate them empirically over several types of distributions. The empirical results suggest that taking estimated node expiration times into account can lead to a better planning strategy.

In this paper, we examine only the one-shot version of the metareasoning problem. Integrating the solutions we present here into a temporal planner can involve solving this problem repeatedly, possibly after each node expansion, in addition to gathering the requisite statistics. These issues are beyond the scope of the current paper. Nevertheless, we devote attention to developing effective metareasoning algorithms that can be useful in practice. When testing our algorithms, we use scenarios based on realistic search trees generated by OPTIC (Benton, Coles, and Coles 2012), the same planner that was adapted in the experiments of Cashmore et al. (2018). Our work provides a firm basis for further efforts to design planners for agents that interact with a wider world containing exogenous processes and other agents, one in which time passes and opportunities can be fleeting.

Problem Statement

To formalize AE2, we abstract away from any particular planning methodology and merely posit the existence of n computational processes, all attempting to solve the same problem. For example, these may represent promising partial plans for a certain goal, implemented as nodes on the frontier of a search tree. There is a single computing thread or processor to run all the processes, so it must be shared. When process i terminates, it will, with probability P_i, deliver a solution or, otherwise, indicate its failure to find one. For each process, there is a deadline, defined in absolute wall

clock time, by which the computation must be completed in order for any solution it finds to be valid, although that deadline may only be known to us with uncertainty. For process i, let $D_i(t)$ be the CDF over wall clock times of the random variable denoting the deadline. Note that the actual deadline for a process is only discovered with certainty when its computation is complete. This models the fact that, in planning, a dependence on an external timed event might not become clear until the final action in the plan is added. If a process terminates with a solution before its deadline, we say that it is *timely*. The processes have performance profiles described by CDFs $M_i(t)$ giving the probability that process i will terminate given an accumulated computation time on that process of t or less. Although some of the algorithms we present may work with dependent random variables, we assume in our analysis that all the variables are independent. Given the $D_i(t)$, $M_i(t)$, and P_i, the objective of AE2 is to schedule processing time over the n processes such that the probability that at least one process finds a solution before its deadline is maximized. This is the essential metareasoning problem in planning when actions expire.

Previous Work

There is much related work on planning under time constraints, such as that by Dean et al. (1995). For an appropriate early survey see Garvey and Lesser (1994). In this section we refer only to existing work that is directly used in developing results for AE2: work in deliberation scheduling (Boddy and Dean 1994) and job scheduling (Yedidsion 2012). In both of these problems, we have a set of n computational processes that need to be allocated processing time on a single processor. Each process $1 \leqslant i \leqslant n$ has a known deadline d_i by which computation in that process must be completed.

Deliberation Scheduling

In deliberation scheduling for time-dependent planning (Boddy and Dean 1994), typically what is being scheduled are anytime algorithms, which exhibit a trade-off between runtime and solution quality (utility). We are thus given a *performance profile*, a mapping $v_i(t_i)$ from the total processing time t_i allocated to process i to the expected utility value generated by that process. The problem is to find a schedule, mapping time to process number, such that the total expected utility $U = \sum_{i=1}^{n} v_i(t_i)$ is maximized, subject to the constraint that all processing allocated for process i occurs no later than d_i. The objective function here is not the same as for AE2, and there is no notion of complete failure to find a solution. However, there is a direct mapping between this version of deliberation scheduling and AE2 with known deadlines, which we mention and exploit later on.

Although the problem is NP-hard, in the special case of *diminishing returns*, it can be solved optimally in polynomial time. The diminishing returns requirement is that the returns slope $\frac{dv_i(t)}{dt}$ be non-increasing in t. We define a similar notion of diminishing returns for AE2 below.

Boddy and Dean (1994) present a deliberation scheduling algorithm for diminishing returns piecewise linear performance profiles. It scans from the **latest** deadline backwards. In the current inter-deadline segment, select, from all profiles whose deadline has **not** expired, the profile i with the greatest slope. Then allocate to process i time sufficient to exhaust this slope segment or to reach the previous deadline, whichever is first. When an earlier deadline is reached, additional profiles become relevant, which may introduce profiles with a better slope. Upon scan is completion, re-arrange the schedule into contiguous segments, in order of deadlines.

Job Scheduling

The minimum tardiness job scheduling problem (Yedidsion 2012) differs from deliberation scheduling in that all processes must be run to completion, and no uncertainty is involved. However, a process i can run faster by paying a cost c_i, modeling the allocation of additional resources. The total job i runtime is a known function $T_i(c)$ of the cost. The goal is to find costs and a schedule such that the sum of the costs is minimized, while ensuring that all jobs finish before their respective deadlines. (Actually, Yedidsion (2012) allows processes to run beyond the deadline by a certain *tardiness* value that is either to be constrained or minimized, but for our purposes we need only refer to the variant that constrains the maximum tardiness to be zero.)

In job scheduling, a polynomial-time scheme is possible if the the speedup is a diminishing-returns function of the cost, i.e. if $T_i'(c) = \frac{dT_i(c)}{dc}$ is an always non-decreasing function of c. Note that $T_i(c)$ is non-negative and $T_i'(c) < 0$.

In the case of diminishing returns, it was shown that there exists an optimal schedule such that

A) time allocations are contiguous, i.e. the schedule can be represented as a list of start-length pairs (s_i, l_i);

B) the allocations are in order of the respective deadlines. We assume w.l.o.g. that $s_1 \leqslant s_2 \leqslant ... \leqslant s_n$; and

C) the performance slopes $T_i'(c)$ obey $T_i'(c_i) \geqslant T_{i+1}'(c_{i+1})$ for all $1 \leqslant i < n$.

As discussed below, these properties also hold for diminishing returns in AE2 (with property C adapted as discussed below). Using these properties, Yedidsion (2012) provides an algorithm to find an optimal schedule in polynomial time, with a complexity depending on speedup profiles T_i representation. An analytical representation manipulable in $O(1)$ is assumed therein, resulting in a runtime $O(n)$.

The Deliberation Scheduling MDP

We now address the AE2 problem of deliberation scheduling with uncertain deadlines. For simplicity, we initially assume that time is discrete and the smallest unit of time is 1. Allowing continuous time is more complex because one needs to define what is done if some time-slice is allocated to a process i, and that process terminates before the end of the time-slice. Discretization avoids this complication.

We can now define our deliberation scheduling problem as an MDP, with distributions represented by their discrete probability function (pmf). Denote $m_i(t) = M_i(t) - M_i(t-1)$, the probability that process i completes after exactly t time units of computation time, and $d_i(t) = D_i(t) - D_i(t-$

1), the probability that the deadline for process i is exactly at time t. Without loss of generality, we can assume that $P_i = 1$: otherwise modify the deadline distribution for process i to have $d_i(-1) = 1 - P_i$, simulating failure of the process to find a solution at all with probability $1 - P_i$, and multiply all other $d_i(t)$ by P_i. This simplified problem we call SEA2. We formalize the SEA2 MDP as an indefinite duration MDP with terminal states, where we keep track of time as part of the state. (An alternate definition would be as a finite-horizon MDP, given a finite value d for the last possible deadline.)

The actions in the MDP are: assign the next time unit to process i, denoted by a_i with $i \in [1, n]$. We allow action a_i only if process i has not already failed.

The state variables are the wall clock time T and one state variable T_i for each process, with domain $\mathcal{N} \cup \{F\}$. T_i denotes the cumulative time assigned to each process i until the current state, or that the process has completed computation and resulted in failure to find a solution within the deadline. We also have special terminal states SUCCESS and FAIL. Thus the state space is:

$$\mathcal{S} = (dom(T) \times \bigtimes_{1 \leqslant i \leqslant n} dom(T_i)) \cup \{\text{SUCCESS, FAIL}\}$$

The initial state is $T = 0$ and $T_i = 0$ for all $1 \leqslant i \leqslant n$.

The transition distribution is determined by which process i has last been scheduled (the action a_i), and the M_i and D_i distributions. If all processes fail, transition into FAIL with probability 1. If some process is successful, transition into SUCCESS with probability 1. More precisely:

- The current time T is always incremented by 1.
- Accumulated computation time is preserved, i.e. for action a_i, $T_j(t+1) = T_j(t)$ for all processes $j \neq i$.
- $T_i(t) = F$ always leads to $T_i(t+1) = F$.
- For action a_i (assign time to process i), the probability that process i's computation is complete given that it has not previously completed is $P(C_i) = \frac{m_i(T_i+1)}{1-M_i(T_i)}$. If completion occurs, the respective deadline will be met with probability $1 - D_i(T_i)$. Therefore, transition probabilities are: with probability $1 - P(C_i)$ set $T_i(t+1) = T_i(t) + 1$, with probability $P(C_i)D_i(T_i)$ set $T_i(t+1) = F$ (process i failed to meet its deadline), and otherwise (probability $P(C_i)(1 - D_i(T_i))$ transition into SUCCESS (the value of T_i in this case is 'don't care').
- If $T_i(t+1) = F$ for all i, transition into FAIL.

The reward function is 0 for all states, except SUCCESS, which has a reward of 1.

Solution Complexity and Approximations

Solving the SEA2 MDP implies finding an optimal conditional (also called *adaptive*) policy, which is a hard computational problem because the state space of the MDP is exponential in n. We show below that the problem is NP-hard, but conjecture that it is PSPACE-complete. As this is a meta-reasoning problem, we cannot afford to devote a lot of computational resources to solving it; instead we attempt to solve easier problems that may give approximately optimal solutions, or optimal solutions to special cases. We begin by considering the optimal *linear* policy. A linear allocation policy is simply a (possibly infinite) sequence A of integers, where $A[t] = i$ means 'assign time slice t to process i.'

There are two possible ways to execute the linear policy. The *basic* scheme (also called "batch", or "non-adaptive") simply executes process $A[t]$ at time t, except when that process has already failed, in which case we leave the processor idle. Upon success, we stop all the computations. Clearly, a better expected performance would result from reallocating the time allocated to an already failed process to the next process in the sequence that has not failed yet. We call this execution method the *semi-adaptive* execution scheme. The semi-adaptive scheme is easy to implement during execution, but hard to analyze, therefore our analysis below is for the basic execution scheme. In some special cases discussed below, the basic and semi-adaptive schemes are equivalent.

As shown below, finding the optimal linear policy is also NP-hard, although it may be easy to approximate using a greedy scheme. However, even if the optimal linear policy can be found, it is not necessarily an optimal solution to the MDP, as shown in the following counter-example.

Example 1: We are given processes $\{1, 2, 3\}$ with deadline distributions $d_1 = [0.5 : -1,\ 0.5 : 2]$, $d_2 = [0.5 : -1,\ 0.5 : 4]$, and $d_3 = [0.4 : -1,\ 0.6 : 4]$. The computation completion time distributions are: $m_1 = [0.1 : 1,\ 0.9 : 2]$, $m_2 = [1 : 2]$, and $m_3 = [1 : 3]$, the latter two being degenerate distributions, i.e. known runtimes times.

The optimal linear policy is $A_{12} = (1, 1, 2, 2, ...)$, where the subscript denotes the processes to which time is allocated. A_{12} delivers a timely solution if either process 1 succeeds (its deadline is not -1) or if process 1 fails yet process 2 succeeds, so we get $P_{12} = 0.5 + (0.5 \times 0.5) = 0.75$. To see that A_{12} is optimal, consider the alternatives. Sequence $A_{13} = (1, 1, 3, 3, 3, ...)$ succeeds only if process 1 succeeds, or if it fails after 1 time unit and process 3 succeeds. If process 1 takes too long to fail, process 3 will not meet its deadline. We get: $P_{13} = 0.5 + 0.5 \times 0.1 \times 0.6 = 0.53$. Any sequence A_3 starting with process 3 succeeds only if process 3 succeeds, with $P_3 = 0.6$. All other sequences are dominated by A_3, A_{13}, or A_{12}, thus A_{12} is the (linear) optimum.

However, the following *adaptive* policy is better than A_{12}: Run process 1 for one time unit. If it terminates with success, then we are done. If it terminates with failure, then run process 3. If it has not terminated, run process 1 for one more unit and allocate the two subsequent time units to process 2. The probability of a timely successful solution for this policy is greater than P_{12}, because in the case where process 1 fails in one time unit, we take advantage of this knowledge to run process 3 instead of process 2 because it has a higher probability of success and can still finish within its deadline.

Whether linear plans are *near-optimal* is an open problem. However, we show below that in the special case where the deadlines are known, the optimal linear policy is also an optimal solution to the MDP. Unfortunately, finding an optimal linear policy remains NP-hard. But if, in addition, the performance profile obeys a certain diminishing returns criterion, the optimal linear policy (and thus the true optimal

policy) can be found in polynomial time. Although realistic cases do not necessarily exhibit these properties, we will see that the solution to the special case can be used effectively as part of an efficient heuristic solution to the MDP.

We begin by formulating the objective function in the cases of a linear policy and a linear *contiguous* policy. For simplicity, we provide the equations for the *basic* execution scheme. For a given scheduled sequence A, we define an *individual allocation* for a task i as a function A_i from non-negative integers to $\{0, 1\}$. $A_i(t)$ is 1 if time slot t allocated for process i and 0 otherwise. Denote by $S_i(t)$ the total number of time slots allocated to i on or before t, i.e.: $S_i(t) = \sum_{t'=1}^{t} A_i(t')$. Then the probability that process i under allocation A_i will find a timely solution is

$$PS_i(A_i) = \sum_t A_i(t) m_i(S_i(t))(1 - D_i(t))$$

In a *contiguous* linear policy, $A_i(t)$ contains only a single contiguous block of 1's. A policy can then be represented as an array of (s_i, l_i) pairs denoting (start, length) of the processing time, respectively, and we can re-write PS_i as

$$PS_i(A_i = (s_i, l_i)) = \sum_{t=0}^{l_i} m_i(t)(1 - D_i(t + s_i))$$

With this representation, the problem becomes: Find an array of pairs $A_i = (s_i, l_i)$ maximizing

$$P_{SUCC} = 1 - \prod_i (1 - PS_i(A_i))$$

Optimization of a product term is inconvenient. We can equivalently minimize probability of failure P_{FAIL}, which is $1 - P_{SUCC}$. Taking logarithms, we seek to minimize:

$$log(P_{FAIL}) = log(\prod_i (1 - PS_i(A_i))) = \sum_i log(1 - PS_i(A_i)) \quad (1)$$

An important special case is known deadlines, where we can re-write the probability of success for process i as:

$$PS_i(A_i) = \sum_{t \leqslant d_i} A_i(t) m_i(S_i(t)) = \sum_{t \leqslant S_i(d_i)} m_i(t) = M_i(S_i(d_i)) \quad (2)$$

because $1 - D_i(t) = 1$ for all $t \leqslant d_i$ and zero for $t > d_i$. Using this probability of success in Equation 1 and using $LF_i(.)$ to denote the log probability of failure, $log(1 - M_i(.))$, the objective function to minimize becomes

$$log(P_{FAIL}) = \sum_i LF_i(S_i(d_i))) \quad (3)$$

Note that in this case of known deadlines, we can map SEA2 into an equivalent (time-dependent planning) deliberation scheduling problem simply by creating performance profiles $v_i(t) = -LF_i(t)$, and keeping the same deadlines. Additionally, if all the known dealines are also *equal*, we now have a problem equivalent to that of allocating runtimes in *algorithm portfolios* (Gomes and Selman 2001), where we wish to maximize the probability that some algorithm in the portfolio can find a solution before the (common) time limit per problem instance.

Theorem 1. *In SEA2 with known deadlines, the optimal linear contiguous policy is an optimal policy for the MDP.*

Proof (outline): Consider any SEA2 MDP policy for known deadlines, represented as a decision tree. Each state-node has a single action edge, leading to a stochastic (chance) node. Each chance node has one or more outgoing edges, each leading to a state node. Given a (state, action) pair $((T, T_1 ... T_n), a_i)$, there are two possible cases. If $T \geqslant d_i$, the deadline for process i is past, and process i fails to find a timely solution (with probability 1): the respective chance node is degenerate, and has only *one* next next state with $T_i = F$. If $T < d_i$, the following chance node has only *two* outgoing edges: either process i terminates successfully (reaching a terminal state SUCCESS), or does not terminate (and the next state has T and T_i incremented by 1). That is, only the latter *one* edge leads to a node that can have outgoing action edges. Thus, a maximum-length path in the tree determines a unique sequence of actions A, which is a linear policy equivalent to this MDP policy.

Now consider any linear policy. The probability of success for process i for known deadlines (Equation 2) depends only on the total processing time given to process i before its deadline d_i. Thus, any schedule that allocates processing time beyond the deadline (called a *tardy* schedule) can be improved (or at least made no worse) by re-allocating such processing time to some other process. We thus consider below only non-tardy schedules.

Under our SEA2 simplifying assumption that $P_i = 1$, if a process completes the computation this ends in SUCCESS. Since Equation 2 depends only on the *total* amount of computation time (before the deadline) for process i, rearranging the schedule by moving allocations makes no difference as long as the resulting schedule remains non-tardy. Therefore the schedule can be rearranged to make allocations contiguous, by making them appear in the same order as the deadlines (i.e. $s_i \leqslant s_j$ is $d_i \leqslant d_j$). Thus the optimal contiguous linear sequence is also an optimal solution to the MDP. □

Note that for non-tardy schedules with certain deadlines, since a completed computation never fails, the simple execution scheme and the semi-adaptive execution scheme are equivalent. Although it is nice to know the form of optimal policies for known deadlines, we have:

Theorem 2. *For a compact representation of the distributions, finding the optimal contiguous linear SEA2 policy in the case of known deadlines is NP-hard.*

Proof (outline): by reduction from the optimization version of knapsack ((Garey and Johnson 1979) problem [MP9]):

Definition 1 (Knapsack problem)**.** *Given a set of items $\mathcal{S} = \{s_1, ...s_n\}$, each with a positive integer weight w_i and value v_i, a weight limit W, find a subset S of $\mathcal{S}$ such that the total weight of S is at most W with a maximal total value.*

In the reduction, each process represents an item in the Knapsack problem. We only need degenerate performance profiles: for each process i, let its deadline be $d_i = W$, and

its performance profile M_i be a piecewise constant function:

$$M_i(t) = \begin{cases} 0, & t < w_i \\ \epsilon v_i, & w_i \leqslant t \leqslant W \\ 1, & W < t \end{cases}$$

ϵ is chosen such that the probability of success of at least one process is "almost" as much as the sum of success probabilities of all the selected processes. Let $H = \max_{i=1}^{n} v_i$. Setting $\epsilon \leqslant \frac{1}{2H^2n^3}$, we can show that every set of items (processes) S we have:

$$\epsilon \sum_{s_i \in S} v_i \geqslant P_{SUCC}(S) = 1 - \prod_{s_i \in S} (1 - \epsilon v_i) > \epsilon((\sum_{s_i \in S} v_i) - 1)$$

Let S be an optimal contiguous schedule. W.l.o.g. we can assume that the processing time assigned to each process $s_i \in S$ is equal to w_i, and that the sum of processing times is at most W. Abusing the notation, we use S to also denote the set of items s_i in the Knapsack problem corresponding to the processes assigned time w_i in S. The Knapsack value of S is $V = \sum_{s_i \in S} v_i$. By construction, the sum of weights of the items in S is at most W, so S is a Knapsack solution.

Assume, in contradiction, that S is suboptimal. Then exists an (integer) Knapsack solution S', with value $V' \geqslant V + 1$. Taking S' as a schedule (assigning time w_i to each process $s_i \in S'$) creates a schedule where all processes run before time W as well, with success probability:

$$P_{SUCC}(S') = 1 - \prod_{s_i \in S'} (1 - \epsilon v_i) > \epsilon((\sum_{s_i \in S'} v_i) - 1)$$
$$\geqslant \epsilon(\sum_{s_i \in S} v_i) \geqslant P_{SUCC}(S)$$

that is, schedule S' has a greater success probability than S, a contradiction. □

Note that the seemingly obvious proof of using the mapping between our deliberation scheduling and the NP-hard equivalent deliberation scheduling in time-dependent planning does not generate a correct NP-hardness proof, as the mapping involves an exponent. This results in number descriptions of exponential size given the size of the description of the problem. Due to Theorem 1, this result (Theorem 2) also implies NP-hardness of the linear (not necessarily contiguous) policy, as well as the NP-hardness of solving the general SEA2 MDP. Observe that a compact representation of the MDP is assumed, rather than a complete transition distribution array, and that solving the MDP in the general case may even be PSPACE-hard (conjecture).

As the linear policy optimization is NP-hard, we now consider further restrictions, and in particular consider the case of diminishing logarithm of returns in the performance profile. That is, suppose that $LF_i(t+1) - LF_i(t)$ is a non-decreasing function of t for all i.

Theorem 3. *The optimal SEA2 policy for the case of known deadlines and diminishing logarithm of returns can be computed in polynomial time.*

Proof (outline): Although it seems to be a different problem entirely, the theorem follows from results on job scheduling with diminishing returns (Yedidsion 2012). The problems are equivalent when we map log probability of failure into costs, both of which need to be minimized. We set the speedup function such that $T_i(LF_i(t)) = T_i(log(1 - M_i(t))) = t$, i.e. set $T_i(c) = exp(1 - M_i^{-1}(c))$, where M_i^{-1} is the inverse function of M_i. Then use any algorithm for the job scheduling problem to get an optimal solution and map it back to SEA2. □

If we have an analytical representation of $LF_i(t)$, we can use the above conversion and the algorithm of Yedidsion (2012) directly. Likewise, if our distribution $LF_i(t)$ is piecewise linear diminishing returns, we can use the algorithm for deliberation scheduling in time-dependent planning (Boddy and Dean 1994).

SEA2 with Diminishing Returns

As in general the performance profile may not be provided analytically, and may also not be piecewise linear, we adapt the above ideas to apply to the discrete distribution representation that we have in SEA2. Properties A and B of an optimal job schedule hold in SEA2. The solution property C for the discrete case is slightly different, the inequality needed here is:

$$LF_i(l_i) - LF_i(l_i - 1) \geqslant LF_{i+1}(l_{i+1} + 1) - LF_{i+1}(l_{i+1})$$

Proving this condition is similar to the proof for the continuous case. That is, if condition C does not hold, then we have for some i:

$$LF_i(l_i) - LF_i(l_i - 1) < LF_{i+1}(l_{i+1} + 1) - LF_{i+1}(l_{i+1})$$

where modifying the deliberation schedule to have: $l_i \leftarrow l_i - 1, l_{i+1} \leftarrow l_{i+1} + 1, s_{i+1} \leftarrow s_{i+1} - 1$, decreases logarithm of probability of failure, without causing it to be tardy.

In the algorithm, which we call ScheduleDR, we keep a queue Q containing the current active profiles, and for each profile the currently allocated time l_i. Q is kept sorted in non-decreasing order of the gain $g_i(t) = LF_i(t) - LF_i(t - 1)$. Psuedo-code is given in Algorithm 1.

Algorithm 1: Scheduling for Diminishing Returns

```
ScheduleDR(Profiles)
For all 1 ⩽ i ⩽ n, let l_i = 0
Let Q = ∅; Order = ∅
for t = max_{i=1}^{n} d_i downto 0 do
    for each i for which d_i = t do
        Insert profile i into Q
        Prepend i to Order
    Retrieve profile j from Q
    Increment l_j, and re-insert into Q
Let t = 0
for j from 1 to n do
    Let s_{Order[j]} = t
    Let t = t + l_{Order[j]}
Return the pairs (s_i, l_i) of all profiles.
```

By construction, the complexity of ScheduleDR is $O((n + d) \log n)$, where $d = \max_{i=1}^{n} d_i$, the latest deadline. Note

that this algorithm is essentially the same as the deliberation scheduling for piecewise linear profiles (Boddy and Dean 1994), adapted to the discrete case.

Non-diminishing returns

In general, as well as in the planning application, the M_i distributions do not have diminishing returns, as we expect $M_i(t)$ to be near zero until some critical value of t (the expected planning time for process i, also called "startup time"), and then quickly increase, followed by a region that behaves according to a diminishing returns rule. An additional complication is that the deadlines can be stochastic. Therefore, we cannot directly use ScheduleDR.

Nevertheless, if the time allocated to each process is at least equal to the critical value, thereby reaching the diminishing returns region, it is still possible to use the algorithm for diminishing returns, as long as we make sure that the algorithm does not ignore processes that have a significant startup time. Therefore, we can convert such profiles that have a startup time into diminishing returns by modifying them to have a rampup starting at 0 (see below).

Under the assumption that all processing time is allocated to process i, starting at time 0, the success distribution for process i is:

$$f_i(t) = PS_i(A_i = (0,t)) = P_i \sum_{t'=0}^{t} m_i(t')(1 - D_i(t'))$$

Define the *most effective computation time* for process i under this assumption to be:

$$e_i = \operatorname*{argmin}_t \frac{log(1 - f_i(t))}{t}$$

Now can modify the function M_i in the region from 0 to e_i to have linear slope in probability of failure. That is, set:

$$LF_i(t) = t\frac{log(1 - M_i(e_i))}{e_i}$$

for all $0 \leqslant t \leqslant e_i$. We now have performance profiles with initially diminishing returns. (For some unimodular distribution this actually results in diminishing returns for all t.)

We also need to handle the uncertain deadlines. We do so by setting a deadline proxy value $d_i(th)$ as a function of D_i and a "confidence level threshold" $0 \leqslant th \leqslant 1$, defined by: $d_i(th)$ = the first t such that $D_i(t) \geqslant th$. Another possibility is to use the expected value of the deadline as if it were a known deadline.

After making these modifications, we can use the algorithm for diminishing returns to create a schedule that is optimal for the modified profiles and the proxy deadlines. However, the resulting schedule may be very far from optimal, as in fact the time allocated for many processes can be significantly lower than its most effective computation time. We have attempted such a solution, and the empirical results were not good. Therefore, other than mentioning these negative results for completeness, we do not further elaborate this method. Nevertheless, some of the intuitions and definitions from this section are still usable in the better-performing real-time greedy scheme we describe below.

Real-time Deliberation Scheduling

Faster meta-reasoning is achievable if we try to allocate computation time just for the first process to run, and defer the rest of the scheduling. This makes sense as in practice the initial computations actually may provide additional information, which the full scheduling does not take into consideration. Note that algorithm ScheduleDR( ) tends to allocate computation time for a process that has an early deadline, but may decide to throw it out (allocate it zero processing time) if its performance slope is too low or it is unlikely to complete computing before its deadline.

The intuition from the diminishing returns optimization is thus to prefer the process i that has the best utility per time unit. However, it is also important to allocate the time now to a process i that has a deadline as early as possible, as this is most critical. We therefore suggest the following greedy algorithm. Whenever assigning computation time, allocate t_d units of computation time to process i that maximizes:

$$Q(i) = \frac{\alpha}{E[D_i]} - \frac{log(1 - f_i(e_i))}{e_i} \tag{4}$$

where α and t_d are positive empirically determined parameters, and $E[D_i]$ is the expectation of the D_i distribution, which we use as a proxy for "deadline of process i". The α parameter trades off between preferring earlier expected deadlines (large α) and better performance slopes (small α).

Empirical Evaluation

In order to empirically evaluate our scheduling methods, we generated several types of performance profiles and deadline distributions based on the following distributions: Uniform (U), with minimal range value $a = 1$ and maximal range value b uniformly drawn from $\{[5,10],[50,100],[100,200],[150-300]\}$, we denote the set of possible $[a-b]$ ranges by R; Boltzmann (B), truncated exponential distribution with the diminishing return property, using a $\lambda \in \{0.1,1,2\}$ and range drawn from R; Truncated Normal Distribution (N) with $\mu \in \{5,50,100,150\}, \sigma \in \{1,5,10\}$, and range drawn from R;

Finally, we used distributions collected from search trees of the Robocup Logistics League (Niemueller, Lakemeyer, and Ferrein 2015) domain generated by the OPTIC planner (denoted by P in the table). To acquire the distribution, A* was executed from each node of the dumped search tree. The result of each of these searches provides the number $N(v)$ of expansions necessary to find the goal under a node v. These numbers were binned separately for each $(h(v), g(v))$ pair. Then, a number of nodes V in the tree was selected randomly, each one standing for a process. For each such $v \in V$, the list of numbers of expansion in the bin corresponding to $g(v)$ and $h(v)$ was treated as a distribution over completion times (in terms of number of expansions). Likewise for creating the latest start times for the resulting plan (the deadline distribution). When using our method as part of a planner, one would need to create such statistics on-the-fly.

Experiments were both with unknown deadline (UK) or with a known deadline (K) which was randomly drawn from the corresponding distribution before the execution.

K / UK	Dist	# pr	MDP		Greedy		ScheduleDR		MPP		RR		Random	
			Q	T	Q	T	Q	T	Q	T	Q	T	Q	T
K	B	2	**0.67**	0.05	0.61	0.01	**0.67**	0.00	0.70	0.00	0.55	0.00	0.54	0.00
		5			0.72	0.04	**0.82**	0.00	0.61	0.00	0.54	0.00	0.57	0.00
		10			0.60	0.07	**0.88**	0.00	0.71	0.00	0.48	0.00	0.51	0.00
		100			0.81	0.74	**0.99**	0.01	0.91	0.03	0.62	0.00	0.60	0.00
	N	2	**0.61**	7.49	**0.56**	0.01	0.45	0.00	0.33	0.00	0.04	0.00	0.07	0.00
		5			**0.83**	0.02	0.72	0.01	0.27	0.00	0.01	0.00	0.03	0.00
		10			**0.93**	0.04	0.41	0.01	0.09	0.00	0.00	0.00	0.03	0.00
		100			**1.00**	0.20	0.70	0.03	0.23	0.02	0.00	0.00	0.00	0.00
	U	2	**0.68**	1.20	0.61	0.04	**0.65**	0.01	0.50	0.00	0.47	0.00	0.48	0.00
		5			**0.90**	0.13	0.88	0.01	0.75	0.00	0.49	0.00	0.47	0.00
		10			**0.98**	0.28	**0.98**	0.01	0.66	0.01	0.44	0.00	0.44	0.00
		100			**1.00**	2.36	**1.00**	0.02	0.80	0.03	0.42	0.00	0.45	0.00
	P	2			0.72	0.02	**0.79**	0.00	0.01	0.00	0.01	0.00	0.04	0.00
		5			0.78	0.06	**0.81**	0.07	0.79	0.02	0.38	0.02	0.54	0.02
		10			**1.00**	0.05	0.87	0.10	0.99	0.00	0.85	0.01	0.82	0.01
		100			**1.00**	0.42	0.91	0.25	0.86	0.04	0.00	0.03	0.04	0.05
Known Average					**0.82**	0.08	0.78	0.01	0.58	0.00	0.33	0.00	0.35	0.00
UK	B	2	**0.68**	147.34	0.61	0.07	0.35	0.00	**0.64**	0.00	0.59	0.00	0.57	0.00
		5			**0.65**	0.18	0.36	0.00	0.63	0.01	0.60	0.00	0.60	0.00
		10			**0.70**	0.45	0.45	0.00	0.66	0.04	0.62	0.00	0.62	0.00
		100			**0.70**	5.45	0.44	0.01	0.65	0.68	0.58	0.00	0.61	0.00
	N	2	**0.66**	26.95	**0.63**	0.07	0.37	0.01	0.20	0.00	0.14	0.00	0.13	0.00
		5			**0.70**	0.19	0.35	0.01	0.09	0.00	0.02	0.00	0.06	0.00
		10			**0.65**	0.41	0.30	0.01	0.15	0.01	0.00	0.00	0.02	0.00
		100			**0.76**	4.02	0.32	0.05	0.06	0.05	0.00	0.00	0.00	0.00
	U	2	**0.73**	103.28	**0.68**	0.33	0.39	0.01	0.53	0.01	0.54	0.00	0.55	0.00
		5			**0.70**	1.25	0.43	0.01	0.57	0.03	0.43	0.00	0.45	0.00
		10			**0.78**	2.07	0.46	0.02	0.59	0.05	0.47	0.00	0.44	0.00
		100			**0.86**	16.56	0.52	0.03	0.59	0.16	0.43	0.00	0.44	0.00
	P	2			**0.61**	0.01	0.24	0.00	0.46	0.00	0.47	0.00	0.52	0.00
		5			**0.90**	0.05	0.54	0.04	0.45	0.03	0.56	0.03	0.59	0.06
		10			**0.90**	0.45	0.32	0.06	0.62	0.06	0.60	0.04	0.62	0.07
		100			**0.85**	3.54	0.77	0.13	0.38	0.01	0.20	0.89	0.33	0.50
Unknown Average					**0.73**	0.47	0.41	0.01	0.45	0.02	0.39	0.00	0.41	0.00
Total Average					**0.77**	0.20	0.60	0.01	0.52	0.01	0.36	0.00	0.38	0.00

Table 1: Solution quality and runtime of the algorithms on different settings

We compared the results of ScheduleDR and the greedy scheme to several naive schemes: (1) random - allocate time to a random process that did not already fail; Most promising plan (MPP) - allocate time to the plan with the highest probability to finish successfully and meet the deadline, in case of failure, the algorithm chooses the next most promising plan and repeats the process; Round-robin (RR) - allocate time units to each non-failed process in equal portions and in circular order. Whenever possible, we also compared with the optimal MDP solution computed using value-determination of the Bellman equation. Unfortunately the space required for the MDP is $O(d^n)$ just for enumerating the state space, even with a compact representation of the transition distribution. Although the runtime is $O(d^{n+1})$, space was the main limiting factor, so we could only compute the optimal score for a few of the smallest instances. We tested ScheduleDR using $th \in \{0.2, 0.5, 0.7, 1\}$ and greedy with $\alpha \in \{0, 0.2, 0.5, 1, 20\}, td \in \{1, 5, 10\}$. However, the reported results include only the best configuration for each algorithm: $th = 0.5$, $alpha = 0$, and $td = 1$.

Evaluating the quality of a solution (policy) is not a trivial task, especially for adaptive policies which depend on the state to make a decision. In order to tackle this issue, we ran the algorithms on each setting for 500 attempts and reported the fraction of successful runs out of the total number of attempts as the solution quality. The results are shown in Table 1. The Q column indicates the solution quality (probability of success using the policy created by the algorithms); the T column is the runtime in seconds. The rows denoted "average" are average solution quality and geometric mean of the runtimes. As expected, ScheduleDR had the best performance when using known deadlines with diminishing returns performance profiles (B). However, in most other cases ScheduleDR performed poorly, especially when the deadlines were unknown. Greedy resulted in the best policy in most cases, and the best average solution quality for both known and unknown deadlines. Overall, the greedy scheme demonstrates a significant improvement over the naive schemes in terms of solution quality. Nonetheless, greedy had the worst runtime out of all other algorithms (except for the MDP) with an average runtime of 1.24 seconds. Note that the time reported is the total time for an *entire* pol-

icy evaluation, therefore, it includes hundreds of decisions at different points. Thus, despite being the slowest of the efficient algorithms, the greedy scheme is sufficiently fast to be used for metareasoning.

Discussion

This paper defined a deliberation scheduling problem that models optimization of the probability of success for finding a timely plan that depends on external events. This dependence can cause a plan of action to expire, and thus our deliberation scheduling problem is similar to that of time-dependent planning, though with a different objective function. In fact, there is a direct mapping between these deliberation scheduling problems that we can exploit. However, in our case the time at which the actions expire (also called deadlines) are uncertain, adding complexity to our version of the problem. An additional surprising connection exists to job scheduling, where some results similar to those in time-dependent planning are useful.

We introduce a formal MDP model of our deliberation scheduling problem, and analyze its complexity. As solving the MDP is computationally hard, we examine the possibility to provide simple ("linear") schedules as a solution, rather than a full MDP policy. Such solutions are shown by counter-example to be suboptimal, except when the problem is restricted to known deadlines, in which case the optimal constant schedule is also an optimal solution to the MDP. Unfortunately we show that even finding the optimal simple schedule is NP-hard. By examining the relationship to time dependent planning and to job scheduling, we can use similar results for the further restricted case of an appropriate form of diminishing returns, where an optimal solution is possible in low-order polynomial time (Boddy and Dean 1994; Horvitz 2001; Yedidsion 2012).

As the restrictions that allow polynomial-time optimal solutions usually do not hold in practice, we develop algorithms that use intuitions from the special case. These are evaluated empirically; one of them, a greedy scheduling algorithm, seems to be close to optimal for many distributions.

Nevertheless, several issues remain open, both theoretical and practical. Some immediate theoretical questions are: Can the optimal policy be approximated in polynomial time within a small constant factor (whether multiplicative or additive)? What is the actual complexity class of the deliberation-scheduling MDP? On the practical side, faster algorithms with good practical results, are needed. **Dynamic** algorithms are especially important due to the main motivation of our problem, which comes from allocating time for search. For example, the different "processes" could actually represent different nodes in a planner's search for a timely plan. In this case, however, nodes are added (and possibly pruned) during the search, thereby adding and deleting processes that need to be scheduled, in some cases modifying node statistics. The allocation effort thus needs to be very fast, but may take advantage of there being only a few changes in the setup each time the search effort is reallocated. An adaptation of our greedy scheme is likely to be applicable, but additional research is required before it can be fully integrated into an existing planner.

Acknowledgements

Partially supported by ISF grant #844/17, and by the Frankel center for CS at BGU. Project also funded by the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No. 730086 (ERGO).

References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*.

Boddy, M., and Dean, T. L. 1994. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence* 67(2):245 – 285.

Cashmore, M.; Coles, A.; Cserna, B.; Karpas, E.; Magazzeni, D.; and Ruml, W. 2018. Temporal planning while the clock ticks. In *ICAPS*, 39–46.

Cresswell, S., and Coddington, A. 2003. Planning with timed literals and deadlines. In *Proceedings of 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, 23–35.

Dean, T. L.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. E. 1995. Planning under time constraints in stochastic domains. *Artif. Intell.* 76(1-2):35–74.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, University of Freiburg.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability, A Guide to the Theory of NP-completeness*. W. H. Freeman and Co. 190.

Garvey, A., and Lesser, V. R. 1994. A survey of research in deliberative real-time artificial intelligence. *Real-Time Systems* 6(3):317–347.

Gomes, C. P., and Selman, B. 2001. Algorithm portfolios. *Artif. Intell.* 126(1-2):43–62.

Hansen, E. A., and Zilberstein, S. 2001. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artif. Intell.* 126(1-2):139–157.

Horvitz, E. 2001. Principles and applications of continual computation. *Artif. Intell.* 126(1-2):159–196.

Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2015. The RoboCup Logistics League as a Benchmark for Planning in Robotics. In *WS on Planning and Robotics (PlanRob) at Int. Conf. on Aut. Planning and Scheduling (ICAPS)*.

Russell, S. J., and Wefald, E. 1991. Principles of metareasoning. *Artificial Intelligence* 49(1-3):361–395.

Yedidsion, L. 2012. Bi-criteria and tri-criteria analysis to minimize maximum lateness makespan and resource consumption for scheduling a single machine. *J. Scheduling* 15(6):665–679.