# Trading Plan Cost for Timeliness in Situated Temporal Planning

**Shahaf S. Shperberg**[1], **Andrew Coles**[2], **Erez Karpas**[3], **Eyal Shimony**[1] and **Wheeler Ruml**[4]
[1]Ben-Gurion University, Israel
[2]King's College London, UK
[3]Technion, Israel
[4]University of New Hampshire, USA
shperbsh@post.bgu.ac.il, andrew.coles@kcl.ac.uk, karpase@technion.ac.il,
shimony@cs.bgu.ac.il, ruml@cs.unh.edu

## Abstract

If a planning agent is considering taking a bus, for example, the time that passes during its planning can affect the feasibility of its plans, as the bus may depart before the agent has found a complete plan. Previous work on this situated temporal planning setting proposed an abstract deliberation scheduling scheme for maximizing the probability of finding a plan that is still feasible at the time it is found. In this paper, we extend the deliberation scheduling approach to address problems in which plans can differ in their cost. Like the planning deadlines, these costs can be uncertain until a complete plan has been found. We show that finding a deliberation policy that minimizes expected cost is PSPACE-hard and that even for known costs and deadlines the optimal solution is a contingent, rather than sequential, schedule. We then analyze special cases of the problem and use these results to propose a greedy scheme that considers both the uncertain deadlines and costs. Our empirical evaluation shows that the greedy scheme performs well in practice on a variety of problems, including some generated from planner search trees.

## 1 Introduction

Situated temporal planning [Cashmore *et al.*, 2018] is a model for the planning problem faced by an agent for whom significant time passes as it plans. In this setting, external temporal constraints (e.g., deadlines) can be introduced depending on the actions included in a plan. For example, taking the 9:00 bus introduces a new constraint that the agent must be at the bus stop by 9:00. These plan-specific constraints make the problem different than real-time search (e.g., [Koenig and Sun, 2009; Sharon *et al.*, 2014; Cserna *et al.*, 2017; Cserna *et al.*, 2018]), deadline-aware search [Dionne *et al.*, 2011], or Best-first Utility Guided Search [Burns *et al.*, 2013].

Situated temporal planning calls for a search strategy different from traditional offline search algorithms, as the choice of which node to expand must account for the fact that time spent exploring one part of the search space passes in the real world and may invalidate other partial plans. Shperberg *et al.* [2019] suggested a rational metareasoning [Russell and Wefald, 1991] scheme for situated temporal planning. They formalized the problem as an MDP (called AE2 for Allocating Effort when Actions Expire) whose actions allocate a unit of time to one of $n$ running processes, showed that solving this MDP optimally is NP-hard, and suggested a greedy decision rule (denoted P-Greedy henceforth) that worked well in their empirical evaluation. However, P-Greedy attempts merely to maximize the chance of finding a timely plan, without considering plan cost. For example, if taking a taxi does not introduce a deadline but is much more expensive than taking the bus, P-Greedy always chooses to take the taxi, even if there is very little uncertainty about whether the agent could catch the bus on time.

In this paper, we extend the metareasoning problem for situated temporal planning to include plan cost. First, we provide a formal characterization of the metareasoning problem with plan costs as an MDP (which we call ACE2 for Allocating Computational Effort when Actions with Costs Expire), and show that optimally solving this MDP is PSPACE-complete. We also show that even when the deadlines and costs are known with certainty, an optimal solution for this problem requires a contingent policy, in contrast with AE2 with known deadlines, where previous work showed that an optimal policy can be linear. Finally, we provide an analytical solution to the special case where only one process may be scheduled, and use this solution to construct a greedy decision rule for the general case. Our empirical evaluation suggests that the new greedy scheme performs significantly better than various baseline algorithms and the P-Greedy scheme on benchmarks featuring several families of distributions, including distributions taken from actual runs of the OPTIC planner [Benton *et al.*, 2012]. This work brings situated temporal planning closer to practical applicability, as plan cost is often an important factor in applications.

## 2 Problem Statement: Cost vs Timeliness

The desire to achieve one's goals as cheaply as possible, as well as in a timely manner, induces a tradeoff between plan cost and the probability of successfully finding a plan that is still executable at the time it is found (except in the rare case where all potential plans have equal cost). In order to make the metareasoning problem well defined, we introduce a cost

of failure $c_f$ and require minimization of the expected cost over outcomes representing either costs of timely plans or the cost of failure. When the value of $c_f$ is high, optimal policies will aim to maximize the probability of finding a plan, while for low values of $c_f$, optimal policies will focus on finding a cheap plan, at the risk of not finding a plan at all. Moreover, the simpler problem of maximizing the probability of success is a special case where the cost of all correct and timely plans is zero and $c_f > 0$.

Following Shperberg *et al.* [2019], we abstract away from any specific planning methodology and merely posit the existence of $n$ computational processes, all attempting to solve the same problem, and a single computing core for running all of the processes. When a process completes, it delivers a solution plan with a known execution cost. However, every solution also has a (possibly unknown) deadline, and if the solution is delivered at a time that is later than that deadline, it cannot be used. A solution delivered at or before its deadline is called *timely*.

We can now define the ACE2 problem, in which the objective is to minimize the expected cost of the timely solution found by the set of processes. The following distributions are assumed to be known: **(i)** $D_i(t)$, the cumulative distribution function (CDF) over wall clock times of a random variable denoting the deadline for each process $i$. **(ii)** $M_i(t)$, the CDF giving the probability that process $i$ will terminate when given an accumulated computation time of $t$ or less. **(iii)** $C_i$, a probability mass function (PMF) over solution costs for process $i$. We denote by $d_{max_i}$ the time of the latest possible deadline for process $i$, i.e. the smallest $t$ for which $D_i(t) = 1$.

The true values of the deadline for process $i$ and the cost of plan $i$ are revealed only when process $i$ completes its computation. We call such a process *completed*, otherwise it is *incomplete*. The cost of a plan of an incomplete process, as well as a completed process that has failed to find a timely solution, is assumed to be $c_f$. Thus, the probability of a process failing to find a plan at all can be incorporated into its cost distribution.

The problem is to find a policy for allocating the computing core's time among the processes, as well as optionally stopping deliberation and executing a complete plan already computed by one of the processes, so as to minimize expected cost of the executed plan. Note that in AE2 there is no need to include an explicit decision to start executing a plan, as once the first feasible plan is found, there is no benefit in searching for better plans. However, when cost is considered, even after the first timely plan is found, we may want to delay executing it in the hope of finding a better plan.

When modeling processes solving the same problem, the distributions over costs and deadlines may have dependencies, complicating both the distribution estimation in practice and the deliberation scheduling. In fact we show below that optimally solving ACE2 with dependencies is PSPACE-hard. Therefore, we make the metareasoning assumption that all these distributions are independent, which in certain simple special cases allows for efficient solutions. Since these simple cases do not cover our desired scenarios, we also present a greedy algorithm for effectively handling the problem, based on intuitions from the special cases.

## 3 Deliberation Scheduling MDP with Costs

Following Shperberg *et al.* [2019], we formulate a discrete-time version of the problem, DACE2, allowing us to model the problem as an MDP. We define $m_i(t) = M_i(t) - M_i(t-1)$, the probability that process $i$ completes after exactly $t$ steps of computation, and $d_i(t) = D_i(t) - D_i(t-1)$, the probability that the deadline for process $i$ is exactly at time $t$.

We formalize the DACE2 MDP as an indefinite duration MDP with terminal states, where we keep track of time as part of the state. The state variables are the wall clock time $T$, and one state variable $T_i$ for each process, with domain $\mathbb{N}$, which represents the cumulative time assigned to process $i$ so far. In addition, we have state variables $ct_i$ and $dl_i$, the cost and deadline (respectively) of each process that has completed its computation. We also have a special terminal state DONE. Since the $ct_i$ and $dl_i$ variables are irrelevant for incomplete processes, and the time assigned to a process is irrelevant to completed processes, the state space can be stated as:

$$\mathcal{S} = \{\text{DONE}\} \cup \left( dom(T) \times \bigtimes_{1 \le i \le n} (dom(T_i) \cup (dom(ct_i) \times dom(dl_i))) \right)$$

There are $2n$ actions in the MDP, $\{a_1, \ldots, a_n\} \cup \{g_1, \ldots, g_n\}$. Action $a_i$ **a**ssigns the next unit of computation time to incomplete process $i$. Action $g_i$ denotes giving the plan computed by completed process $i$ the **g**o-ahead to execute, and transitioning into a terminal state. Note that for every process $i$, either $a_i$ or $g_i$ is applicable but not both.

The initial state $S_0$ has $T = 0$ and $T_i = 0$ for all $1 \le i \le n$. We use the notation $T[S_0] = 0$ and $T_i[S_0] = 0$ (i.e. state variable as a function of the state) as a shorthand to denote this. The transition distribution of the action $a_i$ is determined by the $M_i$ and $D_i$ distributions. If a process has just completed its computation in the transition from state $S$ to state $S'$, then $ct_i[S']$ and $dl_i[S']$ are assigned according to the actual deadline and cost of the solution obtained by process $i$. The state variables for other processes remain unchanged.

More precisely, when transitioning from state $S$ to $S'$ by applying action $a_i$: **(1)** The current time $T[S'] = T[S] + 1$. **(2)** The computation time of every other process remains unchanged, that is $\forall j \neq i : T_j[S'] = T_j[S]$. **(3)** The probability that process $i$'s computation completes in this transition is $P_{term} = \frac{m_i(T_i[S]+1)}{1-M_i(T_i[S])}$. Therefore, with probability $1 - P_{term}$, process $i$ does not complete and we have $T_i[S'] = T_i[S] + 1$. Conversely, with probability $P_{term}$, process $i$ completes. In this case, $ct_i[S']$ and $dl_i[S']$ are assigned values according to distributions $C_i$ and $D_i$, respectively. For example, in the independent cost case, for all $x \le d_{max_i}$, $dl_i[S'] = x$ with probability $d_i(x)$; if $x < T[S']$, then $ct_i[S'] = c_f$, otherwise, for all $y$, $ct_i[S'] = y$ with probability $C_i(y)$.

The reward for executing action $a_i$ before the last deadline of process $i$ has passed ($T[S] < d_{max_i}$) is always 0, but when $a_i$ is applied in state $S$ with $T[S] \ge d_{max_i}$, the reward is $-c_f$. In the latter case, transition into $S' =$ DONE with probability 1. This exception is in order to avoid useless allocation of time to processes where certainly no timely plan can be found, as well as infinite allocation action sequences.

When applying action $g_i$ in state $S$, that is, executing the plan found by completed process $i$, the transition is always to terminal state $S' = \text{DONE}$. The reward in this case is $-ct_i[S]$ if $dl_i \geq T[S]$ and $-c_f$ otherwise.

Although the MDP is a formal statement of the DACE2 problem, we assume that it is specified implicitly through a compact representation of the distribution (including any possible dependencies), as the state space of the MDP is exponential in the number of processes.

# 4 Theoretical Analysis of ACE2

## 4.1 Complexity of ACE2

Optimally solving the ACE2 problem is intractable. In fact, for the general case with unrestricted dependencies we have:

**Theorem 1.** *Finding the first action in an optimal policy for a DACE2 problem is PSPACE-hard.*

*Proof.* By reduction from quantified Boolean formulas (QBF). Let $\alpha$ be a QBF specified in prenex normal form, i.e.:

$$\alpha = \exists x_1 \forall y_1 ... \exists x_n \forall y_n \Phi(x_1, y_1, ...x_n, y_n) \quad (1)$$

with $\Phi$ being a 3-CNF Boolean formula in the $x_i, y_i$ variables. Determining the truth of $\alpha$, or equivalently the existence of a policy for assigning the $x_i$ such that $\Phi$ is always satisfied, is PSPACE-hard (Garey and Johnson [1979], problem LO11).

We transform an instance of QBF into an instance of DACE2 with $2n+2$ processes, whose optimal allocation policy reveals the truth status of $\alpha$. Among these processes, there are $2n$ 'variable setting' processes $p^j_{x_i}$, one for each of the $n$ existential variables and for each $j \in \{t, f\}$. Each $p^j_{x_i}$ always takes exactly 1 time unit. Allocating time to $p^j_{x_i}$ represents assigning the value $j$ to $x_i$. The uncontrollable $y_i$ variables are modeled using the stochastic cost outcomes of the $p^j_{x_i}$ processes. Let $c_f = 2^{n+2}$. Each $p^j_{x_i}$ has three possible costs: 1) $\frac{c_f}{2}$, representing an outcome where the universally quantified variable $y_i$ that follows $x_i$ is false, 2) $\frac{c_f}{2} + 1$ representing an outcome in which $y_i$ is true, and 3) $c_f$, representing a "failure" or that the process is incomplete (because it was not allocated any computation time yet). As a shorthand, we denote the event $C(p^j_{x_i}) = \frac{c_f}{2}$ by $p^j_{x_i,0}$, the event $C(p^j_{x_i}) = \frac{c_f}{2} + 1$ by $p^j_{x_i,1}$, and $C(p^j_{x_i}) = c_f$ by $p^j_{x_i,I}$.

The remaining two processes, denoted by $p_{default}$ and $p_{success}$ are designed to be selected according to whether $\alpha$ is true. $p_{default}$ always takes time $2n+1$ to complete and delivers a solution with a known cost of 1. We show below that the optimal policy schedules the $a$ action followed by the $g$ action for $p_{default}$ just when $\alpha$ is false.

When $\alpha$ is true, we show that it is optimal to schedule the $p^j_{x_i}$ corresponding to a satisfying assignment of $\alpha$ followed by $p_{success}$, which always takes time $n+1$, has a cost distribution depending deterministically on the $p^j_{x_i}$, with cost 0 or $c_f$, depending on other costs, as defined below. If $p_{success}$ is run and delivers a solution of cost 0, the optimal policy then executes that solution (i.e. do the $g$ action for $p_{success}$). The deadline for all processes is $D = 2n+1$, so by construction one can either run $p_{default}$, or run $n$ variable-setting processes and then $p_{success}$ before the deadline.

To test the truth of $\alpha$, we have two tasks: enforcing ordered setting of the $x_i$, and letting the cost of $p_{success}$ be 0 if $\alpha$ is true. We now define the details of the probability distribution over the costs for the $p^j_{x_i,1}$ and $p_{success}$. First, we essentially force one of each $p^j_{x_i}$ pair to be allocated in order to make $p_{success}$ achieve cost 0. For $j \in \{f, t\}$ and $n \geq i \geq 2$,

$$P(p^j_{x_i,I} | p^f_{x_{i-1},I} \wedge p^t_{x_{i-1},I}) = 1$$

Note the dependence of the cost distribution on the observed costs of the preceding $p^j_{x_i}$ processes. The cost distribution for completed processes (for conciseness, we omit this conditioning event in the equations) is $P(p^j_{x_1,0}) = P(p^j_{x_1,1}) = 0.5$ and for $n \geq i \geq 2$, we have:

$$P(p^j_{x_i,0} | \neg(p^f_{x_{i-1},I} \wedge p^t_{x_{i-1},I})) = P(p^j_{x_i,1} | \neg(p^f_{x_{i-1},I} \wedge p^t_{x_{i-1},I})) = 0.5$$

The cost of $p_{success}$ given the costs of the $p^j_{x_i}$ is defined as follows. If for any $1 \leq i \leq n$ we have $p^f_{x_i,I} \wedge p^t_{x_i,I}$, then $C(p_{success}) = c_f$ with probability 1. In all other cases we have exactly one of $C(p^t_{x_i}) < c_f$ or $C(p^f_{x_i}) < c_f$ for all $i$. Thus, the costs of all the $p^j_{x_i}$ encode a unique assignment $A$ to all the Boolean variables. Let $C(p_{success}) = 0$ if assignment $A$ satisfies $\Phi$, and $C(p_{success}) = c_f$ otherwise. So the optimal policy is to choose $p_{default}$ (for a cost of 1) just when $\alpha$ is false. That is because in this case there is a probability of at least $2^{-n}$ of getting a non-satisfying assignment, thus a cost of at least $\frac{c_f}{2}$, and an expected cost $\geq \frac{2^{n+2}}{2^{n+1}} = 2$ if the default is not selected. However, if $\alpha$ is true, there exists a policy that always satisfies $\Phi$, and thus the optimal scheduling policy begins with $p^f_{x_0}$ or $p^t_{x_0}$ (and concludes with $p_{success}$), delivering a cost of 0 with probability 1. □

If the last possible deadline is polynomial in the problem description size, then finding the optimal first action is in PSPACE, since this is a 'game against nature' of polynomial length. Thus, under this restriction, the DACE2 problem is PSPACE-complete. Note that since the DACE2 problem is PSPACE-hard, it follows immediately that ACE2is also PSPACE-hard, as the discrete problem is a special case of the more general model of ACE2.

Although of theoretical significance, membership in PSPACE is no help for practical solution of this scheduling problem, especially as it was conceived as a metareasoning problem, which must be solved in negligible computation time. Henceforth, we make the metareasoning assumption that all the random variables are jointly independent.

## 4.2 The Case of Known Costs

The special case of DACE2 where all plan costs are 0 and the cost of failure is 1 is equivalent to the discrete AE2 model of Shperberg *et al.* [2019] (denoted by SAE2). Further simplifying the SAE2 model to known deadlines, the problem was shown to be NP-hard, even though the optimal schedule is a linear sequence. However, with different costs the optimal schedule is not even necessarily linear.

**Theorem 2.** *Optimal solutions for DACE2 must sometimes consist of a contingent schedule (that depends on previous results delivered by processes), even for the restricted case of known deadlines and known costs.*

*Proof.* Let $c_f = 100$ and consider processes $\{1, 2, 3\}$, with $C_1 = 0, C_2 = 10, C_3 = 15$, all with the same deadline $d_i = 2$. Let the completion-time distributions be: $m_1 = [0.1 : 1, 0.9 : 10], m_2 = [0.5 : 1, 0.5 : 10], m_3 = [1 : 1]$. That is, all processes have some probability of completing computation in one time unit, and otherwise do not complete within the deadline, with process 3 surely completing and delivering a plan costing 15. Thus, by construction, any optimal policy will run a process for at most one time unit, and only 2 processes can be scheduled to run before the (common) deadline.

The optimal policy $P^*$ here is to run process 2 first (action $a_2$). If process 2 completes (thus delivering a cost 10 plan), then run process 1 (action $a_1$), resulting in an expected cost of $(10 \cdot 0.9 + 0 \cdot 0.1 = 9)$ in this branch, because if process 1 completes we have a plan with cost 0 (do action $g_1$), otherwise we have a plan with cost 10 (do action $g_2$). If process 2 does not complete, the best option is $a_3$ to get a plan with cost 15 (and then do $g_3$ to execute it). Expected cost of this policy is thus $E[C(P^*)] = 0.5 \cdot 9 + 0.5 \cdot 15 = 12$. Note that this is a strictly conditional (i.e. non-linear), policy.

To see that $P^*$ is strictly better than any other policy, consider starting with $a_3$, which delivers a plan with cost 15. Now doing $a_1$, we get a 0.1 probability of improving this to 0, so the expected cost is 13.5, while doing $a_2$ at this point we get a probability of 0.5 of improving the cost to 10, and thus a cost 12.5 in expectation, in both cases worse than $P^*$. Alternately, starting with $a_1$, if process 1 completes we get cost 0, but otherwise the best we can do is $a_3$ to get a plan with cost 15. This is also suboptimal (expected cost of this policy is 13.5). □

### 4.3 Simple Special Case: One Running Process

Looking for tractable cases, we consider the case where we are allowed to allocate time to only **one** incomplete process, denoted by $i$. However, we also posit $m - 1$ **completed plans** with known costs $c_j$ and known deadlines $d_j$ for $1 \leq j \leq m - 1$. These plans are results from processes that have completed computation in the past, if any. For convenience, we add an additional completed dummy plan $m$ with $c_m = c_f$ and $d_m = \infty$, denoting the failure case. Without loss of generality, let $0 = d_1 < d_2 < ... < d_m = \infty$ and also $c_1 < c_2 < ... < c_m = c_f$, as a plan $c_j$ that has a cost greater than or equal to $c_{j+1}$ is dominated and can be removed. We can find an optimal stopping policy for the incomplete process $i$ by considering all possible stopping points $d_1, ...d_m$.

**Theorem 3.** *If only one specific incomplete process can be scheduled, the optimal schedule (consisting of the optimal stopping time) can be computed in polynomial time.*

*Proof.* Let policy $\pi_{i,k}(t_d)$ be to do no computation for delay time $t_d$ (needed later on), then run process $i$ until time $d_k$ (if $d_k > t_d$). Finally, execute the cheapest timely plan available when $i$ completes or "times out". Computing the expected cost of $\pi_{i,1}(t_d)$, denoted by $E_{\pi_{i,1}}(t_d)$, is easy because $d_1 = 0$, so no computation time is allowed for process $i$. The best timely plan at time $t_d$ has cost $c_l$ for the first $l$ for which $d_l \geq t_d$, and thus $E_{\pi_{i,1}}(t_d) = c_l$. In particular, $E_{\pi_{i,1}}(0) = c_1$.

The value of $\pi_{i,2}(t_d)$ is more complex. Let us denote by $s_i(t, t_d)$ the probability that process $i$, starting at time $t_d$ and running with no interruption, will complete its computation in at most $t$ additional time units, and succeed in finding a timely solution. Likewise, denote by $f_i(t, t_d)$ the probability that process $i$ will complete computing under the same conditions, and fail to find a solution (or find a solution, but discover that the deadline has passed). That is, we have:

$$s_i(t, t_d) = \sum_{t'=1}^{t} m_i(t')(1 - D_i(t_d + t' - 1))$$

$$f_i(t, t_d) = \sum_{t'=1}^{t} m_i(t') D_i(t_d + t' - 1)$$

Note that $f_i(t, t_d) + s_i(t, t_d) = M_i(t)$ because at any point in time, for a completed process, success and failure (in finding a timely solution) are mutually exclusive events.

If we decide to stop computing at $d_2$, we get cost $c_2$ unless something better came up earlier. That is, we get cost $c_2$ either if process $i$ is incomplete, or if it is complete but failed; however, if process $i$ finds a timely solution, we get the better among the solution cost returned by process $i$ and $c_2$. Thus:

$$E_{\pi_{i,2}}(t_d) = (1 - M_i(d'_2)) \cdot c_2 + (f_i(d'_2, t_d) - f_i(d'_1, t_d)) \cdot c_2 + (s_i(d'_2, t_d) - s_i(d'_1, t_d)) \cdot E[min(C_i, c_2)]$$

where $d'_j = d_j - t_d$. Note that we may actually get a result with expected cost less than $c_1$ even if $E[C_i] > c_1$.

Generalizing to compute the expected cost of $\pi_k(t_d)$: with probability $1 - M_i(t - t_d)$ process $i$ does not complete before $t$, so we pick plan $k$ with cost $c_k$. The probability that the computation stops with failure between $d_{j-1}$ and $d_j$ is: $f_i(d'_j, t_d) - f_i(d'_{j-1}, t_d)$, in which case we pick plan $j$ with cost $c_j$. With probability $s_i(d'_j, t_d) - s_i(d'_{j-1}, t_d)$ the computation generates a timely plan after $d_{j-1}$ but before $d_j$, resulting in expected cost $E[min(C_i, c_j)]$. All in all, we get expected cost:

$$E_{\pi_{i,k}}(t_d) = (1 - M_i(d'_k)) \cdot c_k + \sum_{j=2}^{k} ((f_i(d'_j, t_d) - f_i(d'_{j-1}, t_d)) \cdot c_j + (s_i(d'_j, t_d) - s_i(d'_{j-1}, t_d)) \cdot E[min(C_i, c_j)])$$

Thus in the case of one incomplete process to be scheduled, one can simply compute the values of all the $E_{\pi_{i,k}}(0)$, and select the optimal stopping point. □

Unfortunately, this result cannot be easily extended to schedule more than one incomplete process. One can still solve the MDP in time that is 'only' exponential in the number of incomplete processes that may be scheduled. Obviously this is not reasonable for a large number of such processes, and certainly not for the case where processes actually stand for search nodes and must be scheduled in negligible time. Instead, we use the above ideas to generate a greedy rule that performs well in practice.

## 5 A Greedy Scheme With Costs

When considering only one incomplete process $i$ for scheduling, there was no benefit in allocating runtime other than $d_k$ (for some $k$), as we are not allowed to use time not allocated to process $i$ towards other processes. However, when multiple processes may be scheduled, it is desirable to use as little time as necessary for the current process, as any remaining time can be used by other incomplete processes. Therefore, we generalize the notion of the value of a stopping policy $\pi_k$ to stopping at $d_k$ *subject to the constraint* that at most $t$ time is allocated to the current process (after a delay of $t_d$). The expected value of the constrained policy, denoted by $\pi_k(t, t_d)$, is given by:

$$\begin{aligned} E_{\pi_{i,k}}(t, t_d) = (1 - M_i(\min(t, d'_k))) \cdot c_k + \\ \sum_{j=2}^{k} ((f_i(\min(t, d'_j), t_d) - f_i(\min(t, d'_{j-1}), t_d)) \cdot c_j + \\ (s_i(\min(t, d'_j), t_d) - s_i(\min(t, d'_{j-1}), t_d)) \cdot \\ E[min(C_i, c_j)]) \end{aligned}$$

Although $E_{\pi_{i,k}}(t, t_d)$ is monotonically decreasing in $t$, there is some $t$ at which the expected returns (minus expected costs) per time unit is maximized, and this type of quantity is the useful basis of numerous greedy algorithms. Using this idea, we define the *most-effective reward gain (i.e. cost reduction) rate* for process $i$, relative to the current best valid plan cost $c_c$ as:

$$ecr_i(t_d) = \max_{t,k} \frac{c_c - E_{\pi_{i,k}}(t, t_d)}{t} \tag{2}$$

with $c_c = c_f$ if there is no currently valid plan. This definition resembles the idea behind the returns rate at the 'most effective computation time' ($e_i$) of Shperberg *et al.* [2019], but now in terms of expected cost, rather than logarithm of probability of failure.

It is common to use the value $ecr_i(0)$ (highest rate of returns for process $i$) to select the process $i$ that has the highest returns rate among all processes. However, some processes are more time-critical than others, and this can be measured by how much the returns rate drops due to delaying the start of the processing for process $i$ by $t_d$. The returns rate for a delayed process $i$ is given by $ecr_i(t_d)$. Processes for which this decrease in returns rate $ecr_i(0) - ecr_i(t_d)$ is high should get priority. Trading off high returns rate with loss in returns rate due to delay, we get the following criterion:

$$Q_i(t_d) = ecr_i(0) - \gamma ecr_i(t_d) \tag{3}$$

Note that $\gamma$ is an empirically determined constant that can be used to balance between immediate reward (reward from allocating time to process $i$ now) and future loss (due to delaying time allocation to process $i$ by $t_d$). The value $\gamma = 0.5$ gives these factors equal weight, so one might expect a value not far from that to provide a good balance.

We defined a greedy scheme called Delay-Aware Greedy (DAG) based on Equation 3. This scheme allocates time to a process $i$ that maximizes $Q_i(t_d)$, we used $t_d = 1$ in the experiments. If for each $i$, the optimal individual policy for process $i$ is to stop processing, then the algorithm terminates and executes the first valid plan.

Each iteration of DAG requires computing the $ecr_i$ values of each process. Therefore, a naïve implementation would compute $E_{\pi_{i,k}}(t, t_d)$ for every completed plan, for every process, and for every $1 \leq t \leq d_{max_i}$. Computing each $E_{\pi_{i,k}}(t, t_d)$ takes time $O(m \cdot d_{max_i})$. Thus, a single iteration of DAG requires $O(n \cdot m^2 \cdot d^2_{max})$ time, where $d_{max} = \max_{1 \leq i \leq n} d_{max_i}$. Since there are at most $d_{max}$ iterations, and $m$ is bounded by the number of processes, the execution of DAG takes $O(n^3 \cdot d^3_{max})$. However, DAG can be optimized. First, the values of $s_i$, $f_i$, $E_{\pi_{i,k}}$ and $ecr_i$ can be pre-computed for every $1 \leq t, t_d \leq d_{max}$ in $O(d^2_{max})$ time, and can be used across all iterations. Afterwards, each iteration would only take $O(n)$ time in order to obtain the process with the highest $Q_i$ value. Hence, the overall runtime of DAG can be bounded by $O(n \cdot d^2_{max})$. Finally, if the $M_i$ and $D_i$ distributions are given implicitly, the computations of $ecr_i$, $s_i$ and $f_i$ do not need to consider every $1 \leq t \leq d_{max}$, but rather specific points of interest. For example, if the distributions are piecewise linear, one may only need to examine transition points between segments and some other segment intersections.

## 6 Empirical Results

We tested the new DAG method on DACE2 problems whose performance profiles, cost distributions, and deadline distributions had a variety of forms. Following Shperberg *et al.* [2019], we used: 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 $R$; and Planner (P), which are distributions collected from search trees of the OPTIC planner when run on problems from the Robocup Logistics League [Niemueller *et al.*, 2015] domain. To acquire the planner distributions, A* was executed from each node of a 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 set $V$ was formed by selecting nodes randomly from the tree, 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.

We used the following scheduling algorithms as baselines for the evaluation: **(i) MDP:** solution computed using the Bellman equation (whenever computationally feasible). **(ii) P-Greedy:** the greedy log-probability of failure minimization scheme of Shperberg *et al.* [2019]. **(iii) Random:** allocate time to a random process that has not already failed. **(iv) Most promising process (MPP):** allocate time to the

| Dist | # pr | MDP | | MPP | | RR | | Random | | P-Greedy | | DAG(1/2) | | DAG(1) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | C | T | C | T | C | T | C | T | C | T | C | T | C | T |
| B | 2 | **23.43** | **90,860** | 69.22 | 0 | 80.46 | 0 | 108.74 | 0 | 30.93 | 0 | 26.13 | 0 | **25.88** | 0 |
| | 5 | **19.23** | **$7.9 \times 10^7$** | 40.78 | 0 | 63.73 | 0 | 75.78 | 0 | 29.07 | 0 | 26.61 | 0 | **20.91** | 0 |
| | 10 | | | 31.46 | 0 | 50.98 | 0 | 65.18 | 0 | 19.94 | 0 | 20.93 | 0 | **16.39** | 0 |
| | 100 | | | 26.15 | 0 | 37.89 | 0 | 43.57 | 0 | 14.81 | 10 | 12.21 | 80 | **12.07** | 80 |
| N | 2 | **41.16** | **107,730** | 104.82 | 0 | 166.98 | 0 | 205.69 | 0 | 68.17 | 0 | 63.67 | 0 | **53.19** | 0 |
| | 5 | **40.97** | **$1.2 \times 10^8$** | 99.72 | 0 | 135 | 0 | 171.95 | 0 | 72.38 | 0 | 59.92 | 0 | **47.37** | 0 |
| | 10 | | | 81.1 | 0 | 130.63 | 0 | 151.47 | 0 | 62.87 | 0 | **38.47** | 0 | 40.89 | 0 |
| | 100 | | | 74.6 | 0 | 119.93 | 0 | 153.97 | 0 | 45.18 | 10 | 41.98 | 80 | **38.61** | 70 |
| U | 2 | **30.54** | **112,030** | 86.72 | 0 | 111.61 | 0 | 147.85 | 0 | 55.53 | 0 | 40.11 | 0 | **39.52** | 0 |
| | 5 | **35.18** | **$1.3 \times 10^8$** | 84.21 | 0 | 120.48 | 0 | 152.39 | 0 | 50.93 | 0 | **37.55** | 0 | 38.26 | 0 |
| | 10 | | | 79.9 | 0 | 104.67 | 0 | 138.63 | 0 | 47.42 | 0 | 36.89 | 0 | **35.19** | 0 |
| | 100 | | | 68.19 | 0 | 93.04 | 0 | 121.39 | 0 | 44.14 | 10 | 33.65 | 90 | **30.74** | 80 |
| P | 2 | **193.55** | **328,960** | 456.32 | 0 | 628.26 | 0 | 750.53 | 0 | 299.39 | 0 | 210.44 | 0 | **200.01** | 0 |
| | 5 | | | 386.35 | 0 | 545.11 | 0 | 690.26 | 0 | 275.42 | 0 | 192.47 | 0 | **181.49** | 0 |
| | 10 | | | 369.35 | 0 | 462.62 | 0 | 606.33 | 0 | 228.59 | 10 | 169.87 | 80 | **160.78** | 80 |
| | 100 | | | 254.43 | 10 | 380.15 | 0 | 475.6 | 0 | 171.45 | 50 | 127.01 | 460 | **126.22** | 420 |
| Average | | | | 144.58 | 0 | 201.97 | 0 | 253.71 | 0 | 94.76 | 10 | 71.12 | 50 | **66.72** | 50 |

Table 1: Solution cost and metareasoning runtime (ms) of the algorithms on different types of benchmark problems.

process with the highest probability to find a timely solution. In case of failure, the algorithm chooses the next most promising process. **(v) Round-robin (RR):** allocate time units to each non-failed process in equal portions and in circular order. In addition to a scheduling scheme, we must specify a stopping scheme used by the candidate algorithms. The last three algorithms choose the best (least-cost) timely completed plan $i$ available at any given time, but continue scheduling processes until the (now known) deadline for $i$ arrives, in case a better plan is found. Once the deadline for $i$ arrives, this plan is executed. We compared all the above algorithms to the DAG method using $\gamma \in \{0, 0.2, 0.5, 0.75, 1, 2, 10, 100\}$. However, the reported results include only $0.5$ and $1$, which were the best parameter values.

Evaluating the quality of a solution (policy) is not a trivial task, especially for adaptive policies that 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 average cost over all runs. Since this policy evaluation process introduced noise, we have measured the standard deviation (std) of the solution quality (not reported in the table); the overall std was small ($\pm 3.27$), therefore, the introduced noise does not affect the trends reported below. The results are shown in Table 1. The C column indicates the average cost achieved by the policy created by the algorithms and the T column is the metareasoning runtime in milliseconds. The last rows gives average solution cost and geometric mean of the runtimes. Generally, the DAG scheme achieved the lowest costs among all algorithms (except for the MDP, which is optimal), and demonstrates a significant advantage over the naive schemes in terms of solution cost; $\gamma = 1$ seems to be the best balance between loss due to delay and reward slope. Although DAG was the slowest among all non-MDP algorithms, it is still several orders of magnitude faster than the MDP and required less than 1 second in all cases.

## 7 Conclusion

Situated temporal planning can benefit from metareasoning about the unknown deadlines and search process runtimes. An abstract deliberation scheduling scheme modeling such search processes, aimed at maximizing the *probability* of finding a timely solution, was developed by Shperberg *et al.* [2019], but it did not model the cost of the computed solution. This paper extends the latter scheme to handle plan costs.

An MDP model of the extended deliberation scheduling problem was defined and its complexity was analyzed. We showed that the incorporation of costs significantly complicates the metareasoning problem in several ways. First, even when everything except algorithm runtimes is known (deadlines, costs), the optimal schedule requires contingent policies, rather than just a linear schedule as in the case without costs. Second, the introduction of costs now necessitates a stopping policy, trading off execution of an already computed solution vs. attempting to find better solutions, at the risk of making the current solution(s) expire. Finally, with costs (and dependencies) we proved that finding even the first action in an optimal schedule is PSPACE-hard.

As the MDP has exponential size and is provably intractable, we examined special cases and approximations. We gave a polynomial-time optimal solution for a simple case in which there is only one running process (and a set of completed processes). We present a greedy algorithm (DAG) based on intuitions from this simple case, as well as the greedy algorithm described by Shperberg *et al.* [2019] (P-Greedy). Finally, we compared the new algorithm empirically against P-Greedy and other base-line algorithms; the results of DAG outperform the other methods. For very small instances, where the MDP solution was possible, the DAG scheme was near-optimal.

As more and more agents make plans that interact with the external world in all its temporal complexity, this work will help provide a foundation allowing situated planning to see use in applications where costs play a significant role.

## 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. 821988 (ADE).

## References

[Benton *et al.*, 2012] J. Benton, Amanda Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*, page 2–10, 2012.

[Burns *et al.*, 2013] Ethan Burns, Wheeler Ruml, and Minh Binh Do. Heuristic search when time matters. *J. Artif. Intell. Res.*, 47:697–740, 2013.

[Cashmore *et al.*, 2018] Michael Cashmore, Andrew Coles, Bence Cserna, Erez Karpas, Daniele Magazzeni, and Wheeler Ruml. Temporal planning while the clock ticks. In *ICAPS*, pages 39–46, 2018.

[Cserna *et al.*, 2017] Bence Cserna, Wheeler Ruml, and Jeremy Frank. Planning time to think: Metareasoning for on-line planning with durative actions. In *ICAPS*, pages 56–60, 2017.

[Cserna *et al.*, 2018] Bence Cserna, William J. Doyle, Jordan S. Ramsdell, and Wheeler Ruml. Avoiding dead ends in real-time heuristic search. In *AAAI*, pages 1306–1313, 2018.

[Dionne *et al.*, 2011] Austin J. Dionne, Jordan Tyler Thayer, and Wheeler Ruml. Deadline-aware search using on-line measures of behavior. In *SoCS*, pages 39–46, 2011.

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

[Koenig and Sun, 2009] Sven Koenig and Xiaoxun Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Auton. Agents Multi Agent Syst.*, 18(3):313–341, 2009.

[Niemueller *et al.*, 2015] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. The RoboCup Logistics League as a Benchmark for Planning in Robotics. In *WS on Planning and Robotics (PlanRob) at ICAPS*, 2015.

[Russell and Wefald, 1991] Stuart J. Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49(1-3):361–395, 1991.

[Sharon *et al.*, 2014] Guni Sharon, Ariel Felner, and Nathan R. Sturtevant. Exponential deepening A* for real-time agent-centered search. In *AAAI*, pages 871–877, 2014.

[Shperberg *et al.*, 2019] Shahaf S. Shperberg, Andrew Coles, Bence Cserna, Erez Karpas, Wheeler Ruml, and Solomon Eyal Shimony. Allocating planning effort when actions expire. In *AAAI*, pages 2371–2378, 2019.