Situated Temporal Planning Using Deadline-aware Metareasoning

Shahaf S. Shperberg,[1] **Andrew Coles,**[2] **Erez Karpas,**[3] **Wheeler Ruml,**[4] **Solomon E. Shimony**[1]
[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, ruml@cs.unh.edu, shimony@cs.bgu.ac.il

Abstract

We address the problem of situated temporal planning, in which an agent's plan can depend on scheduled exogenous events, and thus it becomes important to take the passage of time into account during the planning process. Previous work on situated temporal planning has proposed simple pruning strategies, as well as complex schemes for a simplified version of the associated metareasoning problem. Although even the simplified version of the metareasoning problem is NP-hard, we provide a pseudo-polynomial time optimal solution to the case with known deadlines. We leverage intuitions emerging from this case to provide a fast greedy scheme that significantly improves upon previous schemes even for the case of unknown deadlines. Finally, we show how this new method can be applied inside a practical situated temporal planner. An empirical evaluation suggests that the new planner provides state-of-the-art results on problems where external deadlines play a significant role.

1 Introduction

This paper addresses the problem of situated temporal planning, where an agent plans online in the presence of external temporal constraints such as deadlines. For example, if a promising partial plan involves taking a particular train, then it might be worth ensuring that the planning process finishes soon enough that the agent can get to the station in time. In other words, a plan must be found quickly enough that it is possible to execute that plan after planning completes. In this setting, search decisions and temporal constraints interact in complex ways, as choosing to include some action in a plan can introduce a temporal constraint for the subtree of the search tree that includes that action; time spent searching within other subtrees affects the applicability of that action. This differs from other time-aware planning settings, such as real-time heuristic search (Korf 1990), in that each open search node might have a different deadline.

The first planner to address situated temporal planning (Cashmore et al. 2018) uses temporal reasoning (Dechter, Meiri, and Pearl 1991) to prune search nodes for which it is provably too late to start execution. It also uses estimates of remaining search time (Dionne, Thayer, and Ruml 2011) together with information from a temporal relaxed planning graph (Coles et al. 2010) to estimate whether a search node is likely to be *timely*, i.e. likely to lead to a solution that will be executable when planning finishes. As these estimates are not admissible, it uses dual open lists: one only for timely nodes, and another for all nodes (including nodes for which it is likely too late to start execution). However, the planner still uses standard heuristic search (Weighted A*) with these open lists, while noting that this is the wrong thing to do; leaving for future work finding the right search strategy.

Inspired by the situated planning setting, Shperberg et al. (2019) proposed a rational metareasoning (Russell and Wefald 1991) approach for a simplified version of the search problem faced by a situated planner. The problem was simplified in several ways: first, only an abstract version of the metareasoning problem was addressed, and second, distributions over the remaining search time and deadlines were assumed known. The metareasoning problem was formulated as an MDP with the objective of maximizing the probability of finding a timely plan. This was proved to be NP-hard, even when the deadlines are known. However, the reduction was from the Knapsack problem, suggesting the possibility of a pseudo-polynomial time optimal solution algorithm. Shperberg et al. (2019) also suggested a greedy, and somewhat ad-hoc, decision rule (denoted hereafter as basic greedy), which worked well in an empirical evaluation with various types of distributions.

In this paper, we first show that indeed the known deadline case can be solved in pseudo-polynomial time through dynamic programming (DP). Despite being optimal when deadlines are known, the DP approach does not perform well with unknown deadlines. Our second contribution is an alternate greedy decision rule, called DDA, that is better justified theoretically than basic greedy; we show empirically that DDA delivers better results than the basic greedy scheme in the same abstract setting of the problem.

Finally, our third contribution is to integrate the new metareasoning scheme as the search strategy for the situated temporal planner of Cashmore et al. (2018). An empirical evaluation shows that the new approach leads to timely solutions for significantly more problems than using standard heuristic search, even with pruning late nodes and dual open lists. This is an important step in bringing situated temporal planning closer to practical utility.

2 Background

We start by reviewing formal models of situated temporal planning and the associated metareasoning problem. Although heuristic search with external temporal constraints can arise in many settings, we focus in this paper on the problem of situated domain-independent temporal planning.

2.1 Problem Statement

Following Cashmore et al. (2018), we formulate situated temporal planning as propositional temporal planning with Timed Initial Literals (TIL) (Cresswell and Coddington 2003; Edelkamp and Hoffmann 2004). Such problems are specified by a tuple $\Pi = \langle F, A, I, T, G \rangle$, where:

- F, a set of Bool. propositions describing the world state.
- A is a set of durative actions; each action $a \in A$ has:
  - Duration in the range $[dur_{\min}(a), dur_{\max}(a)]$
  - Start condition $cond_{\vdash}(a)$, invariant condition $cond_{\leftrightarrow}(a)$, and end condition $cond_{\dashv}(a)$, all of which are subsets of F, and
  - Start effect $\mathit{eff}_{\vdash}(a)$ and end effect $\mathit{eff}_{\dashv}(a)$, both of which specify which propositions in F become true or false when a starts or ends, respectively.
- $I \subseteq F$ is the initial state, and $G \subseteq F$ specifies the goal.
- T is a set of timed initial literals (TIL). Each TIL $l = \langle time(l), lit(l) \rangle \in T$ consists of a time $time(l)$ and a literal $lit(l) \in F$, specifying a proposition that becomes true (or false) at $time(l)$.

A solution to a situated temporal planning problem is a schedule σ: a sequence of triples $\langle a, t_a, d_a \rangle$, where $a \in A$ is an action, $t_a \in \mathbb{R}^{0+}$ is the time when action a is started, and $d_a \in [dur_{\min}(a), dur_{\max}(a)]$ is the duration chosen for a. To define a valid schedule, we view it as a set of instantaneous *happenings* (Fox and Long 2003) that occur when an action starts, when an action ends, and when a timed initial literal is triggered. For each triple $\langle a, t, d \rangle$ in σ, we have action a starting at time t (requiring $cond_{\vdash}(a)$ to hold a small amount of time ϵ before t, and applying the effects $\mathit{eff}_{\vdash}(a)$ right at t), and ending at $t + d$ (requiring $cond_{\dashv}(a)$ to hold ϵ before $t + d$, and applying the effects $\mathit{eff}_{\dashv}(a)$ at $t + d$). For TIL l we have the effect specified by $lit(l)$ triggered at $time(l)$. We require the invariant condition $cond_{\leftrightarrow}(a)$ to hold over the open interval between t and $t + d$, and the goal G to hold after all happenings have occurred.

The difference from standard temporal planning is that here we interpret the TILs as encoding temporal constraints in absolute time since planning started. Thus, we require the schedule σ to start only after planning is completed. That is, if the planner started at time 0 and took t_p time for its planning, we require that $t_a \geq t_p$ for all $\langle a, t_a, d_a \rangle \in \sigma$.

2.2 Metareasoning in Situated Planning

The requirement $t_a \geq t_p$ implies that the plan must be fully generated before the minimum t_a, which may be unknown until planning completes. For a partial plan available at a search node i in the planner, this can be modeled by a random variable d_i, denoting the unknown deadline by which a potential plan expanded from node i must be generated. Thus, the planner faces the metareasoning problem of deciding which nodes on the open list to expand in order to maximize the chance of finding a plan before its deadline.

Shperberg et al. (2019) propose a model of this problem called (AE)^2 ('allocating effort when actions expire') which abstracts away from the planning problem and merely assumes n independent processes. Each process attempts to solve the same problem under time constraints. In the context of situated temporal planning using heuristic search (which we explore further below), each process may represent a promising partial plan for the goal, implemented as a node on the open list eager to have its subtree explored. But the abstract problem may also be applicable to other settings, such as algorithm portfolios or scheduling candidates for job interviews. For simplicity, we assume a single processor, so the core of the metareasoning problem is to determine how to schedule the n processes on the single processor.

When process i terminates, it delivers a solution with probability P_i or, otherwise, indicates its failure to find one. For each process, there is a deadline defined in absolute wall clock time by which its computation must be completed in order for any solution it finds to be valid. The deadline may be uncertain and is provided as a probability distribution. For process i, let $D_i(t)$ be the CDF over wall clock times of the random variable denoting the deadline. The actual deadline for a process is only discovered with certainty when the process completes. This models the fact that a dependence on an external timed event might not become clear until the final action in a plan is added. If a process terminates with a solution before its deadline, we say that it is *timely*. Given $D_i(t)$, we assume w.l.o.g. that P_i is 1, otherwise one can adjust $D_i(t)$ to make the probability of a deadline that is in the past (thus forcing the plan to fail) equal to $1 - P_i$.

The processes have known search time distributions, (performance profiles (Zilberstein and Russell 1996)) described by CDFs $M_i(t)$, the probability that process i needs total computation time t or less to terminate. Although some of the algorithms we present can handle dependencies, we make the typical metareasoning assumption in our analysis that all random variables are independent. Given the $D_i(t)$ and $M_i(t)$ distributions, the objective of (AE)^2 is to schedule processing time between the n processes maximizing the probability of at least one process finding a timely solution.

A simplified discrete-time version of the problem, called S(AE)^2, can be cast as a Markov decision process. The MDP's actions are to assign (schedule) the next time unit to process i, denoted by a_i with $i \in [1, n]$. Action a_i is allowed only if process i has not already failed. A process is considered to have failed if it has terminated and discovered that its deadline has already passed, or if the current time is later than the last possible deadline for the process.

The state variables are the wall clock time T and one state variable T_i for each process, with domain $\mathbb{N} \cup \{F\}$, although in practice the time domains of T, T_i are bounded by the latest possible deadlines. T_i denotes the cumulative time assigned to each process i until the current state, or that the process failed (indicated by F). We also have special termi-

nal states SUCCESS and FAIL. Thus the state space is:

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

The initial state has $T = 0$, and $T_i = 0$ for all $1 \le i \le n$. The transition distribution is determined by which process i has last been scheduled (the action a_i), the M_i distribution (which determines whether currently scheduled process i has completed its computation), and D_i (which determines the revealed deadline for a completed process, and thus whether it has succeeded or failed). If all processes fail, transition into FAIL (with probability 1). If some process is successful, transition into SUCCESS. The reward is 0 for all states except SUCCESS, for which the reward is 1.

The size of the state-space of the S(AE)^2 MDP is exponential in the number of processes, so it is impractical to fully specify the MDP explicitly or to solve it directly. Furthermore, the S(AE)^2 problem is NP-hard, even for known deadlines (denoted KDS(AE)^2) (Shperberg et al. 2019).

2.3 Basic Greedy Scheme

As S(AE)^2 is NP-hard, Shperberg et al. (2019) used insights from a diminishing returns result to develop a greedy scheme. They restrict their attention to linear contiguous allocation policies: schedules where the action taken at time t does not depend on the results of the previous actions, and where each process receives its allocated time contiguously. Using the p.m.f. $m_i(t') = M_i(t') - M_i(t'-1)$, the probability that process i finds a timely plan when allocated t_i consecutive time units *beginning* at time t_{b_i} is:

$$s_i(t_i, t_{b_i}) = \sum_{t'=0}^{t_i} m_i(t')(1 - D_i(t' + t_{b_i})) \quad (1)$$

Example 1. *Let $m_1 \sim [0.5 : 2; 0.5 : 5]$, i.e. process 1 requires 2 time units or 5, equally likely; and $d_1 = 2$ with probability 1 (known deadline). Then $s_1(2, 0) = 0.5$, and $s_1(2, 1) = s_1(1, 0) = 0$. Thus, process 1 delivers a timely solution with probability 0.5 given 2 time units at $t_{b_1} = 0$; and is useless with $t_1 < 2$ or $t_{b_1} \ge 1$.*

When considering linear contiguous policies, we need to allocate t_i, t_{b_i} pairs to all processes (with no allocation overlap). Note that overall a timely plan is found if at least one process succeeds, that is, overall failure occurs only if all processes fail. Thus, to maximize the probability of overall success P_s (over all possible linear contiguous allocations), we need to allocate t_i, t_{b_i} pairs so as to maximize:

$$P_s = 1 - \prod_i (1 - s_i(t_i, t_{b_i})) \quad (2)$$

Using $LPF_i(\cdot)$ ('logarithm of probability of failure') as shorthand for $log(1 - s_i(\cdot))$, we note that P_s is maximized if the sum of the $LPF_i(t_i, t_{b_i})$ is minimized and that $-LPF_i(t_i, t_{b_i})$ behaves like a utility that we need to maximize. For known deadlines, we can assume that no policy will allocate processing time after the respective deadline. We will use $LPF_i(t)$ as shorthand for $LPF_i(t, 0)$.

To bypass the problem of non-diminishing returns, the notion of *most effective computation time* for process i under the assumption that it begins at time t_b and runs for t time units was defined as:

$$e_i(t_b) = \operatorname*{argmin}_t \frac{LPF_i(t, t_b)}{t} \quad (3)$$

$e_i(t_b)$ is a generalization of e_i from Shperberg et al. (2019) which equals $e_i(0)$ here. We use e_i to denote $e_i(0)$ below.

Example 2. *For process 1 from Example 1 we have (log base 2): $LPF_1(t_1, t_{b_1}) = -1$ for all $t_1 \ge 2$ and $t_{b_1} = 0$, and $LPF_1(t_1, t_{b_1}) = 0$ for all other t_1 and t_{b_1} configurations; so $e_1(0) = 2$.*

Since not all processes can start *now*, intuitions from diminishing returns are: to prefer process i that has the best utility per time unit, i.e. such that $-LPF_i(e_i))/e_i$ is greatest. Still, allocating time now to process i delays other processes, so it is also important to allocate the time now to processes with an early deadline. Shperberg et al. (2019) thus suggested the following greedy algorithm: Iteratively allocate t_u units of computation time to process i maximizing:

$$Q(i) = \frac{\alpha}{E[D_i]} - \frac{LPF_i(e_i)}{e_i} \quad (4)$$

where α and t_u are positive-valued parameters, and $E[D_i]$ is the expectation of the random variable with CDF D_i (a slight abuse of notation). α trades off between preferring earlier deadlines (large α) and better performance slopes (small α).

Example 3. *Add to Example 2 process 2 with $d_2 = 4$ and $m_2 \sim [0.75 : 2; 0.25 : 20]$. Note that process 1 cannot be delayed as $d_1 = 2$, but process 2 can be delayed by 2 time units. Thus, the optimal policy (which results in $P_s = 7/8$) is to start with process 1 and then move to process 2. However, $LPF_2(2, t_{b_2}) = -2$ and $e_2(t_{b_2}) = 2$ for $t_{b_2} \le 2$, with $-\frac{LPF_2(e_2)}{e_2} = 1$, better than $-\frac{LPF_1(e_1)}{e_1} = 0.5$. Nonetheless, with $\alpha = 10$, we have initially $Q(1) = 10/3 + 0.5 = 23/6$ and $Q(2) = 10/4 + 1 = 21/6$, so start with process 1, as required.*

3 New Metareasoning Schemes

Our new results for the S(AE)^2 are a (pseudo) polynomial-time algorithm for the case of known deadlines (dropping the diminishing returns requirement), and a better justified greedy scheme that also works better in practice.

3.1 DP Solution for Known Deadlines

Although S(AE)^2 is NP-hard, Shperberg et al. (2019) showed that under the additional restriction of diminishing returns (non-decreasing logarithm of probability of failure) an optimal schedule can be found in (pseudo) polynomial time. However, planning processes do not have diminishing returns. We therefore examine deliberation scheduling when the diminishing returns assumption is removed.

For KDS(AE)^2 (known deadlines S(AE)^2), it is sufficient to examine linear contiguous allocation policies (Shperberg et al. 2019). We extend this result by showing that restricting the schedules to ones with processes sorted by an increasing order of deadlines is still optimal:

Theorem 1. *Given a KDS(AE)^2 problem, there exists a linear contiguous schedule with processes sorted by a non-decreasing order of deadlines that is optimal.*

Proof. Given an optimal linear contiguous policy P, we show that it can be rearranged to have non-decreasing order of deadlines. Let consecutive processes $i, i+1$ in P be the first two processes with deadlines $d_{i+1} < d_i$. Let P' be the same as P but with the starting times of i and $i+1$ exchanged. Since P' differs from P only in the order of i and $i+1$, all processes allocated either before i or after $i+1$ are unaffected, as they maintain the same starting time and duration allocations. In addition, since the time allocation for both processes is unchanged, the only way to decrease the solution quality is by violating the deadline of either i or $i+1$. Making process $i+1$ start earlier cannot cause it to violate its deadline. Furthermore, $d_i > d_{i+1}$ and d_{i+1} was not violated in P, therefore, d_i also cannot be violated in P'. Thus, the success probability for P' is not less than that of P. These steps can be repeated until an optimal schedule sorted by a non-decreasing order of deadlines is obtained. □

Theorem 1 can be used to obtain a DP scheme.

Theorem 2. *For known deadlines, an optimal schedule can be found in time polynomial in n, d_n using DP according to*

$$OPT(t,l) = \max_{0 \le j \le d_l - t} (OPT(t+j, l+1) - LPF_l(j)) \qquad (5)$$

Proof outline. We show by induction that $OPT(t,l)$ is the utility of the optimal linear contiguous ordered (LCO) schedule for processes l through n for the 'remaining' time $d_n - t$. The base cases are when $l = n+1$ (no processes to be assigned), thus $OPT(\cdot, n+1) = 0$; or no time remaining, thus $OPT(d_n, \cdot) = 0$. Assume that $OPT(t', l')$ is the utility of the optimal LCO schedule for every $t' \ge t$ and $l' > l$. To compute the utility of the optimal LCO schedule for processes l through n for the "remaining" time $d_n - t$, we need to consider all time allocations j for process l. For each time allocation j, we add the reward resulting from the allocation, which is $-LPF_l(j)$, to the best utility over every possible optimal schedule of processes $l+1$ through n, given that j time was already allocated. The latter is exactly $OPT(t+j, l+1)$ according to the inductive hypothesis. Note that any allocation beyond the deadline of process l (d_l) is wasteful and cannot contribute to the utility. Thus, we can only consider time allocations j between 0 and d_l–t. This computation is exactly the right-hand side of Equation 5.

Thus, $OPT(0,1)$ is the maximal utility among all LCO schedules for processes from 1 to n starting at $t = 0$. Due to Theorem 1, $OPT(0,1)$ is the maximal utility among all schedules, indicating an optimal solution to KDS(AE)^2. □

If the representation of the M_i is explicit, the algorithm evaluating Equation 5 in descending order runs in polynomial time. Otherwise, it is pseudo-polynomial and can be approximated in polynomial time.

3.2 Delay-Damage Aware Greedy Scheme

Although the pseudo-polynomial algorithm for KDS(AE)^2 is reasonably fast, it is still too slow for metareasoning in a planner. Furthermore, it is not applicable when deadlines are unknown. Thus, we next examine a new greedy scheme, beginning with pointing out shortcomings of the greedy scheme from Shperberg et al. (2019); namely, using the proxy $E[D_i]$ in the value $Q(i)$ is somewhat ad-hoc and fails when the deadline distribution has a large variance.

Example 4. *Modify Example 3 so that process 2 deadline $d_2 \sim [0.75 : 2; 0.25 : 10]$, thus $E[D_2] = 4$ as before. Then $Q(1), Q(2)$ are unaffected and basic greedy behaves the same. However, now both process 1 and process 2 cannot be delayed. Therefore, the optimal policy now is to schedule only process 2 for $P_s = 9/16$.*

A more principled scheme can use the utility per time unit as in $Q(i)$, but with a first term that is better justified theoretically. The first term of $Q(i)$ is used for prioritizing processes with early deadlines, as any delay might prevent them from completing their computation before their deadline, even if they *would* have been timely had they been scheduled for processing immediately. Therefore, instead of the first term, it makes sense to provide a measure of the 'utility damage' to a process i due to delaying its processing start time from time 0 to time t_d. Consider the case of two processes and allocating contiguous processing time to each of them, each equal to their most effective computation time e_i. For simplicity assume $e_1 = e_2 = e_{1,2}$. In this case, we can write down the 'utility' (negative logarithm of probability of failure) for first running process 1 and then process 2, as:

$$U(1,2) = -LPF_1(e_{1,2}, 0) - LPF_2(e_{1,2}, e_{1,2})$$

where the second term is for LPF for process 2 delayed to the end of the run of process 1. Likewise, for process 2 first:

$$U(2,1) = -LPF_2(e_{1,2}, 0) - LPF_1(e_{1,2}, e_{1,2})$$

Since $e_1 = e_2 = e_{1,2}$, we can re-normalize the utility terms by adding $LPF_2(e_{1,2}, e_{1,2}) + LPF_1(e_{1,2}, e_{1,2})$ to get:

$$U'(1,2) = -LPF_1(e_{1,2}, 0) + LPF_1(e_{1,2}, e_{1,2}) \qquad (6)$$
$$U'(2,1) = -LPF_2(e_{1,2}, 0) + LPF_2(e_{1,2}, e_{1,2})$$

U' is the difference in utility we could get for a process when it is run at time 0, minus the utility it can achieve if delayed by $e_{1,2}$ time units, and thus we call U' the 'utility loss of delaying the process by $e_{1,2}$ time units'. The advantage of using U' as a measure of process i is that its value in equation 6 depends only on process i. Although this optimality argument does not necessarily extend to more than 2 processes or to non-contiguous schedules, it has the advantage of correctly addressing the deadline distributions.

Example 5. *In Example 3, $LPF_2(2,0) = LPF_2(2,2) = -2$ and $LPF_1(2,0) = -1$ but $LPF_1(2,2) = 0$. So $U'(1,2) = 3 > U'(2,1)$ and process 1 is first as needed. In Example 4, $LPF_2(2,2) = 0$, $LPF_2(2,0) = log(7/16)$, so $U'(2,1) > U'(1,2)$ and process 2 is run as required.*

In practice, a greedy scheme assigns some $t_u < e_i$ time units to process i at a time. As the most effective time e_i is not assigned in one chunk, it makes sense to assign time in order of the utility slope available if we allow process i to run now, rather than the total utility. In other words, we will use the utility slope as a proxy for the potential gain. This was done in the second term in Equation 4 in the basic greedy scheme, and thus we still prefer a process i that maximizes:

$$slopenow_i = {}^{-LPF_i(e_i, 0)}/_{e_i}$$

However, as suggested by Equation 6, a measure of the urgency of process i w.r.t. the deadlines is in certain cases proportional to the utility loss due to delaying the process. The utility slope after delay of process i by t_u is:

$$slopelater_i = {}^{-LPF_i(e_i(t_u),t_u)}/_{e_i(t_u)}$$

The higher the slope after delay, the greater the gain, thus the smaller the loss, and the lower the urgency. Thus, this term is to be minimized.

Since the time t_u allocated at each round is not equal to e_i; and since with more than two processes the delay a process suffers is unknown, the tradeoff between these terms is not necessarily equal. We use an empirically determined constant multiplier γ to balance between exploiting the current process reward from allocating time to process i now and the loss in reward due to delay. Thus, the delay-damage aware (DDA) greedy scheme is to assign, at each processing allocation round, t_u time to the process i that maximizes:

$$Q'(i) = \frac{\gamma \cdot LPF_i(e_i(t_u), t_u)}{e_i(t_u)} - \frac{LPF_i(e_i, 0)}{e_i} \qquad (7)$$

3.3 Evaluation on S(AE)^2

We performed an empirical evaluation in order to assess the effectiveness of the different metareasoning schemes on the abstract problem, and to decide which scheme to integrate into an actual planner. Following Shperberg et al. (2019), we generated problems with performance profiles and deadline distributions based on a variety of 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; and Planner (P), distributions collected from search trees of the Robocup Logistics League (Niemueller, Lakemeyer, and Ferrein 2015) domain generated by the OPTIC planner. To acquire the planner distributions, 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 set of nodes V was selected randomly from the trees, each node standing for a process. For each such $v \in V$, the list of numbers of expansions 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). Experiments were run with unknown deadlines, and with a known deadline randomly drawn from the corresponding distribution before execution.

The algorithms we evaluated are: the optimal MDP solution (whenever possible), computed using the Bellman equation in value-determination; the basic greedy scheme using $\alpha \in \{0, 0.2, 0.5, 1, 20\}$; the DDA scheme using $t_u = 1$ and $\gamma \in \{0, 0.2, 0.5, 0.75, 1, 2, 10, 100\}$; and the dynamic programming (DP) scheme, treating the expected deadline of each process as its true deadline when the deadlines are unknown. The reported results are only the best parameter values: $\alpha = 0$ for basic greedy and $\gamma = 1$ for DDA.

Dist	# pr	MDP		Basic		DDA		DP	
		Q	T	Q	T	Q	T	Q	T
B	2	***0.72***	***0.1***	0.63	0.00	0.66	0.00	**0.72**	0.00
	5			0.71	0.00	0.78	0.01	**0.83**	0.01
	10			0.63	0.01	0.75	0.08	**0.88**	0.09
	100			0.82	0.07	0.99	0.19	**1.00**	0.21
N	2	***0.61***	***7.7***	0.55	0.00	0.57	0.00	**0.61**	0.00
	5			**0.84**	0.00	**0.84**	0.01	**0.84**	0.01
	10			0.92	0.01	**0.93**	0.06	**0.93**	0.08
	100			**1.00**	0.02	**1.00**	0.23	**1.00**	0.28
U	2	***0.66***	***1.5***	0.62	0.00	0.64	0.01	**0.66**	0.02
	5			0.85	0.02	0.85	0.09	**0.91**	0.10
	10			0.97	0.03	**0.98**	0.22	**0.98**	0.31
	100			**1.00**	0.08	**1.00**	0.78	**1.00**	0.71
P	2	***0.82***	***11.7***	0.71	0.00	0.77	0.00	**0.82**	0.00
	5			0.77	0.00	0.82	0.02	**0.83**	0.03
	10			0.98	0.01	**1.00**	0.09	**1.00**	0.1
	100			**1.00**	0.05	**1.00**	0.27	**1.00**	0.24
Known, Avg.				0.81	0.02	0.85	0.13	**0.87**	0.14
B	2	***0.69***	***188.1***	0.62	0.01	0.65	0.03	0.50	0.03
	5			0.65	0.02	**0.77**	0.12	0.57	0.14
	10			0.71	0.06	**0.75**	0.48	0.72	0.55
	100			0.70	0.21	**0.82**	1.19	0.68	1.43
N	2	***0.69***	***25.6***	0.61	0.01	**0.69**	0.04	0.47	0.04
	5			0.70	0.02	**0.87**	0.10	0.66	0.13
	10			0.64	0.05	**0.72**	0.44	0.55	0.49
	100			0.76	0.19	**0.84**	2.03	0.73	1.95
U	2	***0.73***	***112.8***	0.67	0.04	**0.73**	0.25	**0.73**	0.26
	5			0.69	0.19	**0.78**	1.26	0.45	1.12
	10			0.79	0.22	**0.91**	2.25	0.84	2.31
	100			0.85	0.88	**0.89**	7.83	0.74	7.50
P	2	***0.81***	***20.6***	0.62	0.00	0.77	0.01	0.63	0.01
	5			0.89	0.00	**0.93**	0.03	0.81	0.06
	10			**0.9**	0.05	**0.9**	0.38	0.88	0.39
	100			0.86	0.21	**0.95**	2.21	0.75	2.23
Unknown, Avg.				0.73	0.14	**0.81**	1.17	0.67	1.17
Total, Avg				0.77	0.08	**0.83**	0.65	0.77	0.65

Table 1: Solution quality and runtime for different settings

Evaluating the quality of a solution (policy) is not trivial, especially for adaptive policies. 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. 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 (± 0.02), therefore, the introduced noise does not affect the trends reported below. The results are shown in Table 1, in which the top and bottom halves of the table contains results of the known and unknown deadlines cases respectively. Q indicates the solution quality (success probability achieved by the policy created by the algorithm); T is the metareasoning runtime in seconds. 'Avg' rows give average solution quality and geometric mean of the runtimes. DP is optimal (and therefore the best) when the deadlines are known; however, it performed poorly for unknown deadlines. For the unknown deadlines case, and on average across

both cases, DDA achieved the best solution quality, outperforming the basic greedy scheme. However, DDA had the worst metareasoning runtime, except for the MDP, which we must consider when integrating it with a planner.

4 Integrating DDA into a Planner

DDA was the best performing algorithm in Section 3.3, while the basic greedy scheme (with $\alpha = 0$) had the best runtime. Since DDA with $\gamma = 0$ is equivalent to basic greedy with $\alpha = 0$, it suffices to implement DDA in the planner. In order to do so, several issues must be addressed. First, the non-trivial task of obtaining the distributions; second, how to use the DDA scheme as the basis for search.

4.1 Estimating the Distributions

DDA needs the D_i and M_i distributions as input. To estimate these distributions, we leverage estimates easily obtained in the situated temporal planner on which we build (Cashmore et al. 2018). The planner uses the temporal relaxed planning graph (TRPG) (Coles et al. 2010) to estimate $E[D_i]$, and the distance to go (also from the TRPG) to estimate remaining search time (Dionne, Thayer, and Ruml 2011), which we treat as an estimate of $E[M_i]$.

However, to use the DDA greedy rule we must also estimate the whole distribution, rather than just the expected values. For $D_i(t)$, we simply use a step function that goes from probability zero to one when t is *equal* to the deadline of the relaxed plan (our estimation of $E[D_i]$). In order to estimate M_i, we use an online temporal difference learning technique (Thayer, Dionne, and Ruml 2011). Note that the true distance-to-go from some state s, denoted $d^*(s)$ is equal to $d^*(bc(s)) + 1$, where $bc(s)$ is the *best child* of state s, that is, the successor that is on the best path to the goal. If d is a heuristic estimate of d^*, then the one-step error of d at s is defined as $\epsilon_d(s) = d(bc(s)) + 1 - d(s)$. The average error of d is then estimated by the average one-step error, either throughout the entire search space or along a specific path.

We extend these ideas to estimate M_i. First, as we have the one-step errors for all states observed so far, we can estimate the distribution of one-step errors of distance to go, denoted O_d. Since these errors accumulate along the path to the goal, to estimate the distribution of the distance to go from state s, we convolve O_d with itself $d(s)$ times. We then multiply this distribution by the average expansion delay divided by the expansion rate (Cashmore et al. 2018), which measures how much time passes, on average, between generating a state and expanding it.

One important implementation detail is that, when the expansion delay is small, the remaining-search-time estimate may unrealistically have zero probability of failure. Thus we add to the M_i distribution an infinite remaining search time outcome, with probability $pf_{\min} = 0.0001$.

4.2 Searching with DDA

For search, each state i on the open list can be treated as a process. Hence, the $Q'(i)$ value (Equation 7) is maintained for each state i, and computation time is allocated to a state with highest $Q'(i)$. Recalling that the DDA scheme is based on allocating t_u units of computation time, thus we perform t_u expansions in the *subtree* rooted at i; after t_u expansions, the non-expanded (frontier) nodes in this subtree are added to the open list, and another state is chosen according to Q'.

One important aspect of searching based on Q' values is that these values are not static: during search, the distribution O_d changes, and time passes, all of which change the $Q'(i)$ values. Hence, we recompute them every t_u expansions, i.e. whenever the next subtree to allocate time to is to be chosen. To reduce the metareasoning overhead (for instance, to keep it under some desired proportion of planner runtime), one could consider changing the frequency of this during search, but as we have not found the overheads to be excessive in our experiments, we leave this for future work.

Since DDA needs statistics, such as one-step error estimates, that are unavailable early in the search, we actually start off by using the baseline weighted A* search (Cashmore et al. 2018). The DDA ordering kicks in only after nexp expansions (an algorithm parameter).

In some domains the distribution information is unhelpful, e.g. if the deadline is very far, in which case probability of success for many nodes is close to 1, both before and after delay. Alternately, the deadline may be very close, in which case the probability of success is close to 0, regardless of delay for many nodes. In such cases, many leading Q' values are identical, and our scheme may become erratic. In such cases we break ties or near-ties by preferring nodes with lower f value. This tie-breaking was implemented by actually expanding nodes with the highest $Q'(i) + \beta f(i)$, where a very small $\beta = -0.000001$ was used throughout.

One could also imagine a more sophisticated scheme in which the algorithm monitors the distribution of Q' values to assess whether they contain useful information or suffer from substantial estimation error. This interesting issue remains for future work.

5 Empirical Evaluation

To evaluate the DDA metareasoning scheme, we implemented it on top of the situated temporal planner of Cashmore et al. (2018), which itself is implemented on top of OPTIC (Benton, Coles, and Coles 2012). As a baseline, we use the heuristic search scheme of Cashmore et al. (2018). We used the same set of benchmarks as this prior work, which includes all IPC domains with Timed Initial Literals (TILs), representing constraints on absolute time; as well as 200 instances from the Robocup Logistics League (RCLL) (Niemueller et al. 2016), a simulated robotic manufacturing setting – divided into instances with 1 or 2 robots, and a Turtlebot office delivery domain with tight deadlines from Cashmore et al. (2019). All experiments were run on a server with 72 Intel Xeon E5-2695 CPUs, using up to 64 processes in parallel (Tange 2011) and a 3GB memory limit.

5.1 Comparing DDA to the Baseline

We begin by comparing DDA, with the default parameter values ($t_u = 100, \gamma = 1$, and nexp $= 1000$), to the baseline. These values were the result of an initial guess, which we verified empirically. In situated temporal planning, the

Domain	baseline		DDA		DDA(nexp = 1)		DDA($\gamma = 0$)		DDA(hs)		DDA (dom tuned)	
airport	19.0	(19–19)	20.0	(20–20)	20.0	(20–20)	20.0	(20–20)	19.1	(19–20)	20.5	(19–21)
pw-nt	4.0	(3–4)	4.0	(3–5)	4.5	(3–5)	4.0	(3–4)	4.0	(4–4)	3.9	(3–5)
rcll 1	37.7	(37–40)	73.7	(53–92)	73.0	(65–91)	76.4	(69–81)	34.6	(15–75)	83.9	(59–99)
rcll 2	1.0	(1–1)	4.0	(2–23)	1.1	(0–14)	2.0	(2–2)	1.8	(1–7)	2.7	(0–13)
sat cmplx	5.0	(5–5)	5.0	(5–5)	4.8	(4–5)	5.0	(5–5)	5.0	(5–5)	3.8	(2–5)
sat tw	5.0	(5–5)	5.0	(5–5)	5.1	(5–6)	5.0	(5–5)	5.0	(5–5)	3.6	(3–5)
trucks	6.0	(6–6)	6.9	(6–9)	6.3	(6–7)	7.5	(7–8)	6.5	(6–7)	5.7	(5–8)
turtlebot	14.0	(14–14)	12.5	(10–13)	13.0	(13–13)	8.0	(8–8)	14.0	(14–14)	13.0	(13–13)
umts-flaw	4.1	(4–5)	5.1	(5–6)	0.0	(0–0)	0.1	(0–1)	0.4	(0–4)	5.0	(5–5)
umts	48.0	(48–48)	45.5	(42–49)	44.2	(41–48)	43.8	(41–48)	41.5	(41–42)	45.7	(44–49)
TOTAL	143.8	(142–147)	181.7	(151–227)	172.0	(157–209)	171.6	(160–182)	131.7	(110–183)	187.7	(153–223)

Table 2: Number of problems solved by each planner, shown as: average (solved by all 20 runs - solved by at least one run).

time elapsed during planning affects search decisions. This introduces noise due to multiple processes sharing the same CPU, as well as features such as Intel Turbo Boost, which vary the speed of the CPU according to load. Thus, in this experiment we ran the planner 20 times on each planning problem. Each run was limited to 200 seconds of CPU time.

Table 2 shows the (average) number of problems solved in each domain for each planner. Each entry shows the average number of problems solved in that domain by that planner (averaging over the 20 runs). Furthermore, in parentheses we give the number of problems solved by *all* 20 runs (of the given planner in the given domain) and the number of problems solved by *at least one* of the 20 runs (of the given planner in the given domain). These numbers serve as a confidence interval of sorts, as they indicate ease of replication, thus we denote them by low bar and high bar, respectively.

Looking only at the DDA and baseline columns for now, observe that DDA solves 38 more problems than the baseline. Interestingly, the low bar of 151 for DDA is higher than the high bar for the baseline – that is, there were 151 instances that were solved by all 20 runs of DDA, compared to 147 that were solved by at lease one run of the baseline.

Also note that the "noise" (the difference between the high bar and the low bar) for DDA is much higher than the baseline. This is because the baseline only uses elapsed search time to prune search nodes, but otherwise keeps the same ordering between nodes based on f-values. On the other hand, DDA uses elapsed time to compute Q', thus changing the ordering between nodes.

However, looking at the total coverage could be misleading, as the numbers of problems in each domain are different — and the number of solvable problem even more so (varying from about 5 to almost 100). Thus, we also count in how many domains DDA outperformed the baseline. DDA beats the baseline in 4 domains: airport, RCLL (with 1 and 2 robots), trucks, and UMTS flaw, and loses only in 2 domains: turtlebot and UMTS.

To further analyze this result, we exploited the fact that we have 20 different runs for each planner on each problem. Thus, we can perform a statistical significance test on the number of times each planner solved each problem, and check whether DDA is statistically significantly better than the baseline on each problem, whether the baseline is statistically significantly better than DDA, or whether there is no statistically significantly difference. Specifically we used the Mann-Whitney U-test (Mann and Whitney 1947), a non-parametric test that checks if one variable is more likely to have a higher value than another. Table 3 reports the number of problems in each domain in which the baseline or DDA were statistically significantly better than the other. The results here correspond well with the domains listed above, and show that in total DDA is statistically significantly better than the baseline on 3.5 times more problems.

5.2 DDA Ablation Studies

Having seen that DDA outperforms the previous work in situated temporal planning, we performed some ablation studies to test which of the features of DDA contribute the most to its success. We considered the following planner variants:

DDA(nexp $= 1$) immediately starts with DDA, instead of waiting 1000 node expansions for the estimates to stabilize.

DDA($\gamma = 0$) uses $\gamma = 0$ to compute $Q'(i)$, thus ignoring the slope later component of Q'.

DDA(hs) uses standard heuristic search based on Q', instead of doing t_u expansions in the chosen subtree.

In all of these variants, the other parameters were kept the same as the default configuration. Table 2 also shows the results for these variants. The results show that, indeed, every one of these features contributes somewhat to the success of DDA, with the focused search leading to many more solved problems. Interestingly, in the turtlebot domain, where DDA loses to the baseline, the pure heuristic search variant solves all the problems — the same as the baseline. This is likely because there are dead ends in the domain (due to deadlines expiring), and heuristic search does better at avoiding these.

5.3 Automated Parameter Tuning for DDA

The results we presented above were based on hand-chosen default parameter setting for DDA. On the one hand, this is only one point in a large space of possible parameter settings, and thus DDA has the potential for even better performance. On the other hand, there could be a concern that these parameter settings were chosen based on their performance on the problem domains being evaluated, and thus there is overfitting in the process.

Therefore, we also show that it is possible to automatically find good settings for each domain automatically, using SMAC – Sequential Model-Based Algorithm Configuration (Hutter, Hoos, and Leyton-Brown 2011; Lindauer et al.

Domain	baseline	DDA
airport	0	1
pw-nt	0	0
rcll 1	7	48
rcll 2	0	4
sat cmplx	0	0
sat tw	0	0
trucks	0	2
turtlebot	2	0
umts-flaw	0	1
umts	7	1
TOTAL	16	57

Table 3: Statistically significant wins

	baseline				DDA			
Domain	0.25	0.5	2	4	0.25	0.5	2	4
airport	19.0	19.0	19.0	19.0	20.0	20.0	20.0	20.0
pw-nt	0.0	0.0	9.0	12.1	0.0	0.0	10.0	12.2
rcll 1	4.2	9.0	39.4	46.3	10.1	35.8	70.7	72.6
rcll 2	0.0	0.0	1.0	2.0	0.6	0.2	5.4	8.4
sat cmplx	0.0	0.0	5.0	5.0	0.0	0.0	5.0	5.0
sat tw	0.0	0.0	6.0	7.0	0.0	0.0	6.0	7.4
trucks	0.0	0.0	7.0	13.0	0.0	0.0	9.1	13.0
turtlebot	0.0	4.1	14.0	14.0	0.0	4.0	14.0	14.0
umts-flaw	0.0	4.8	4.3	2.2	3.5	5.5	5.0	5.0
umts	16.0	33.7	42.0	42.0	16.0	35.4	42.9	42.0
TOTAL	39.2	70.6	146.7	162.6	50.2	100.9	188.1	199.6

Table 4: Coverage with different TIL multipliers

2017). To avoid overfitting, we divided each domain into 2 folds – the evenly numbered problems and the odd numbered problems (this was done to try to maintain the same distribution of problem sizes in both folds). We then used SMAC — specifically, Bayesian Optimization and HyperBand (Falkner, Klein, and Hutter 2018; Li et al. 2017) — to find the best configuration for each fold. This configuration was then used to evaluate the other fold. The results reported here use the configuration trained on the even problems to measure the performance of DDA on the odd problems, and vice versa — thus avoiding overfitting.

We gave SMAC 72 hours to find the best configuration. We fixed the values of $\min_{pf}$ and β, as well as the decision to use the subtree-focused search. Thus we searched for values for the remaining parameters: γ was limited to values between -10 and 10, t_u to values between 10 and 1000 (on a logarithmic scale), and nexp to values between 1 and 10,000 (also on a logarithmic scale). The score for each run was the total search time in seconds (if the problem was solved), or 2^{31} if the problem was not solved. To overcome noise, each problem was run 3 times, and the average score was used.

Table 2 also reports the results of DDA tuned for each domain (these are the combined results from both configurations in each domain). Overall, the domain-tuned version of DDA outperforms DDA in coverage. However, in a few domains, performance is actually worse. Notably, these are the domains where DDA solves very few problems to begin with (up to 7), and thus the signal for training is rather weak. On the other hand, in domains where DDA already solved 10 or more problems, the domain-specific parameter tuning helped improve performance. We believe this problem could be alleviated by using a random problem generator, but this is beyond the scope of this paper.

5.4 Evaluating the Impact of Tight Deadlines

We conclude the empirical evaluation by examining more closely when DDA outperforms the baseline. First, note that when the deadlines are very loose, there should be no difference between DDA and the baseline, as they will both have time to explore the search space. Second, when the deadlines are very tight, both approaches are likely to fail to find a solution before the deadline — in fact, this is the case in pipesworld-no-tankage (pw-nt). Thus, we expect DDA to outperform the basline in the "Goldilocks" zone, where deadlines are tight, but not too tight.

To evaluate this claim empirically, we ran the baseline and DDA on the same problem instances as before, modified by multiplying the TILs in the problem by a factor. We tried factors of 0.25 and 0.5 (for tighter deadlines) and 2 and 4 (for looser deadlines). The number of problems solved by each planner in each domain is shown in Table 4, where the number in each cell is the average of 10 runs.

First, observe that DDA outperforms the baseline overall for all multipliers. Second, looking at the pipesworld domain (pw-nt), we can see the phenomenon we described above. For multiplier of 0.25 and 0.5, both approaches solve 0 (and from Table 2, for a multiplier of 1, both solve 4). When the multiplier is 2, DDA solves 1 more problem than the baseline, but when the multiplier is 4 (and deadlines are now very loose), the difference becomes only 0.1. A similar phenomenon occurs in rcll 2 and in trucks.

6 Discussion

We advanced a formal metareasoning approach for situated temporal planning. We note that optimizing the time allocation in an algorithm portfolio with known probabilistic performance profiles is a special case where there is a common deadline by which all processes must deliver the result, namely the (usually) known time limit per instance. Despite using rather crude estimates, the enhanced planner empirically outperforms previous work. This is an important step in making situated temporal planning practical. It also demonstrates the enduring power of metareasoning as a productive perspective on rational resource-bounded problem-solving.

We assumed here that the plan must be completed before execution. However, with very tight deadlines, it may be necessary to start execution before a complete plan is found. We intend to expand the metareasoning scheme to a (more complicated) model which supports execution of actions while still searching for a plan. Finally, we mean to enrich OPTIC with other metareasoning schemes (Shperberg et al. 2020) tailored for the problem of minimizing expected cost (rather than trying to maximize the probability of finding a solution).

Acknowledgements

This research was supported by Grant No. 2019730 from the United States-Israel Binational Science Foundation (BSF) and by Grant No. 2008594 from the United States National Science Foundation (NSF). Project also funded by the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No. 821988 (ADE), by a Grant from the GIF, the German-Israeli Foundation for Scientific Research and Development, by ISF grant #844/17, and by the Frankel center for CS at BGU.

References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *ICAPS*, 210. AAAI Press.

Cashmore, M.; Coles, A.; Cserna, B.; Karpas, E.; Magazzeni, D.; and Ruml, W. 2018. Temporal Planning While the Clock Ticks. In *ICAPS*, 39–46. AAAI Press.

Cashmore, M.; Coles, A.; Cserna, B.; Karpas, E.; Magazzeni, D.; and Ruml, W. 2019. Replanning for Situated Robots. In *ICAPS*, 665–673. AAAI Press.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *ICAPS*, 42–49. AAAI Press.

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.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artif. Intell.* 49(1-3): 61–95.

Dionne, A. J.; Thayer, J. T.; and Ruml, W. 2011. Deadline-Aware Search Using On-Line Measures of Behavior. In *SOCS*, 39–46.

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.

Falkner, S.; Klein, A.; and Hutter, F. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, 1436–1445. PMLR.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research (JAIR)* 20: 61–124.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION*, volume 6683 of *Lecture Notes in Computer Science*, 507–523. Springer.

Korf, R. E. 1990. Real-Time Heuristic Search. *Artif. Intell.* 42(2-3): 189–211.

Li, L.; Jamieson, K. G.; DeSalvo, G.; Rostamizadeh, A.; and Talwalkar, A. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *J. Mach. Learn. Res.* 18: 185:1–185:52.

Lindauer, M.; Eggensperger, K.; Feurer, M.; Falkner, S.; Biedenkapp, A.; and Hutter, F. 2017. SMAC v3: Algorithm Configuration in Python. https://github.com/automl/SMAC3.

Mann, H. B.; and Whitney, D. R. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Annals of Mathematical Statistics* 18(1): 50–60.

Niemueller, T.; Karpas, E.; Vaquero, T.; and Timmons, E. 2016. Planning Competition for Logistics Robots in Simulation. In *ICAPS Workshop on Planning and Robotics (PlanRob)*.

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

Russell, S. J.; and Wefald, E. 1991. Principles of Metareasoning. *Artif. Intell.* 49(1-3): 361–395.

Shperberg, S. S.; Coles, A.; Cserna, B.; Karpas, E.; Ruml, W.; and Shimony, S. E. 2019. Allocating Planning Effort When Actions Expire. In *AAAI*, 2371–2378. AAAI Press.

Shperberg, S. S.; Felner, A.; Sturtevant, N.; Shimony, S. E.; and Hayoun, A. 2020. Bidirectional Heuristic Search: Expanding Nodes by a Lower Bound. In *IJCAI*, 4775–4779. ijcai.org.

Tange, O. 2011. GNU Parallel - The Command-Line Power Tool. *;login: The USENIX Magazine* 36(1): 42–47. URL http://www.gnu.org/s/parallel.

Thayer, J. T.; Dionne, A. J.; and Ruml, W. 2011. Learning Inadmissible Heuristics During Search. In *ICAPS*, 250–257.

Zilberstein, S.; and Russell, S. J. 1996. Optimal Composition of Real-Time Systems. *Artif. Intell.* 82(1-2): 181–213.