Situated Planning for Execution Under Temporal Constraints

Michael Cashmore, Andrew Coles
King's College London

Bence Cserna
University of New Hampshire

Erez Karpas
Technion — Israel Institute of Technology

Daniele Magazzeni
King's College London

Wheeler Ruml
University of New Hampshire

Abstract

One of the original motivations for domain-independent planning was to generate plans that would then be executed in the environment. However, most existing planners ignore the passage of time during planning. While this can work well when absolute time does not play a role, this approach can lead to plans failing when there are external timing constraints, such as deadlines. In this paper, we describe a new approach for time-sensitive temporal planning. Our planner is aware of the fact that plan execution will start only once planning finishes, and incorporates this information into its decision making, in order to focus the search on branches that are more likely to lead to plans that will be feasible when the planner finishes.

Introduction

One of the original motivations for domain-independent planning was for controlling robots performing complex tasks (Fikes and Nilsson 1971). The typical approach to controlling robots using a planner is to call the planner to generate a plan which solves the problem, and then execute that plan in the environment. This approach works well if the plan remains applicable regardless of when it is executed. However, if there are external timing constraints, such as deadlines which must be met, things become more complex. This is because we must take into account the *planning time*.

For example, in the Robocup Logistics League (RCLL) challenge (Niemueller, Lakemeyer, and Ferrein 2015), a team of robots must move workpieces between different machines that perform some operations on them, and fulfill some order with a deadline. This calls for using temporal planning, because we would like all robots to work in parallel, and actions have different durations. The typical approach would have the planner come up with a plan which would work had it been executed at time 0, and then execute this plan when the planner completes. Obviously, this might lead to missing the deadline, and thus, plan failure.

One simple approach to handling this problem is to use some estimate on how long planning will take, and adapt all the deadlines assuming plan execution would start when the planner finishes. While using an upper bound on planning time will eliminate the problem of plans failing, it might lead to the planner not finding a feasible plan to begin with. On the other hand, using too low an estimate could still lead to plans failing, as discussed above.

In this paper, we describe a new approach for situated temporal planning. Our planner is aware of the fact that plan execution will start once planning finishes, and incorporates this information into the internal data structure for temporal reasoning used by the planner, together with estimates of remaining planning time. This helps our planner prune partial plans which are likely to lead to the planner finishing planning too late for the plans to be of use, and focus on more promising branches of the search.

Our empirical evaluation demonstrates that this planner can handle temporal planning problems with absolute deadlines much better than a naive baseline approach, in realistic settings where planning time counts, and the plan can only start executing once it is completed. To the best of our knowledge, this is the first temporal planner to explicitly consider planning time, within the context of planning and execution. Thus, our planner is especially applicable to online planning for robotics, where a robot must find a plan to execute, but the world does not stop while the robot is planning.

Preliminaries

We consider propositional temporal planning problems with Timed Initial Literals (TIL) (Cresswell and Coddington 2003; Edelkamp and Hoffmann 2004). Such a planning problem Π is specified by a tuple $\Pi = \langle F, A, I, T, G \rangle$, where:

- F is a set of Boolean propositions, which describe the state of the world.
- A is a set of durative actions. Each action $a \in A$ is described by:
  - Minimum duration $dur_{\min}(a)$ and maximum duration $dur_{\max}(a)$, both in $\mathbb{R}^{0+}$ with $dur_{\min}(a) \leq dur_{\max}(a)$,
  - 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 $eff_{\vdash}(a)$ and end effect $eff_{\dashv}(a)$, both of which specify which propositions in F become true (add effects), and which become false (delete effects).
- $I \subseteq F$ is the initial state, specifying exactly which propositions are true at time 0.

- T is a set of timed initial literals (TIL). Each TIL $l \in T$ consists of a time $time(l)$ and a literal $lit(l)$, which specifies which proposition in F becomes true (or false) at time $time(l)$.
- $G \subseteq F$ specifies the goal, that is, which propositions we want to be true at the end of plan execution.

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

Related Work

Temporal planners have of course been used in on-line applications before. For example, researchers at PARC built a special-purpose temporal planner for on-line manufacturing (Ruml et al. 2011). As in many temporal planners, each search node contains a Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991) to represent the time points of events in the plan and constraints on when they can occur. To reflect the fact that actions cannot occur until planning has completed, the PARC planner includes a hard-coded estimate of the required planning time, and every time point in the STN is constrained to occur at least that far after the time that planning started (Ruml et al. 2011, Figure 11). While this is a reasonable solution in a domain where the expected planning problems are all of similar difficulty, this approach can perform poorly in domains that include a wide variety of problems, as we will see below.

There has also been work on time-aware planning in the search community. Dionne, Thayer and Ruml (2011) present a so-called 'contract algorithm' called Deadline-Aware Search (DAS) that, given a deadline, attempts to return the cheapest complete plan that it can find within that deadline. The main part of the algorithm works by estimating the time that will be required to find a solution beneath each node in the open list, and pruning those for which this estimate exceeds the remaining search time. The estimate is the product of three quantities that are determined on-line: the time required to expand a node, expressed in seconds, an estimate on the number of search nodes remaining on the path to a goal beneath the given node, notated $d(n)$, and the average number of expansions required before a generated node is selected for expansion, called the *expansion delay*. Although DAS was shown to surpass anytime algorithms on combinatorial benchmarks, its ideas have never been implemented in a domain-independent planner.

Bugsy (Burns, Ruml, and Do 2013) is a search algorithm that attempts to minimize the user's utility, which is represented as a linear combination of planning time and plan cost. If plan cost is makespan, then the utility measures the 'goal achievement time', or the time from when the goal is presented to the planner, and planning starts, to when the plan finishes executing, and the goal is achieved by the agent. Bugsy is a best-first search algorithm, and relies on an estimate of remaining planning time similar to that of DAS in order to estimate the utility of each node it expands. While Bugsy is sensitive to its own planning time, it is not cognizant of external timed events such as deadlines, and does not prune nodes based on temporal information.

Related concepts in the search community include real-time search and anytime search. In the real-time search setting, the planner must return within a prespecified time bound the next action for the agent to take. This differs from our setting, in which the planner must return a complete plan and the temporal constraints are fine-grained and can relate individual domain propositions to absolute times. In anytime search, a planner quickly finds a complete plan, and then uses additional computation time to improve it until either it is terminated by an external signal or an optimal solution is found. In our setting, the planner may not run indefinitely, but rather is expected to minimize the agent's goal achievement time. And while doing so, we demand that the planner recognize that time is passing and that it be responsive to timed events in the external world.

Encoding Planning and Execution Time

Many temporal planners (e.g., (Coles et al. 2009; 2012; 2010; Benton, Coles, and Coles 2012; Fernández-González, Karpas, and Williams 2015; 2017)) rely on an internal Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991) (or possibly a linear program or a convex optimization problem — but we will abuse terminology and call all of these the STN) to represent the temporal constraints between the set of the *time points* where actions start or end. Specifically, planners that support required concurrency (Cushing et al. 2007) tend to use this representation to support concurrent execution of actions.

When planning is done offline, the STN contains some time point t_{ES}, which is the start of plan execution, and is assigned the value of 0. For convenience, we split each occurrence of action a in the plan into two snap-actions: $a_{\vdash}$ and $a_{\dashv}$, corresponding to the start and end of the action, respectively. For each of these we have a corresponding time point in the STN: $t(a_{\vdash})$ when a starts, and $t(a_{\dashv})$ when a ends. Actions which have started but not yet finished will only have the start time point, since this is a partial plan (as noted earlier, all starts eventually need to be paired with an end, but this is not a requirement of plans that are still under construction). Temporal constraints between the time points are either action *duration* constraints (between the time points of the same action occurrence), or *sequencing* constraints due to causal relations between actions. For example, if the end of action a achieves the start conditions of action b, then

we would have $t(a_\dashv) - t(b_\vdash) \geq \epsilon$, where ϵ is the minimum separation between two events that depend on each other (Fox and Long 2003). Or, if the start of c threatens the preconditions of d, then $t(c_\vdash) - t(d_\dashv) \geq \epsilon$. Additionally, timed initial literals (TIL) (Edelkamp and Hoffmann 2004) are encoded into the STN by adding a time point $t(f)$ for the occurrence of TIL f, with the temporal constraint $t(f) - t_{ES} = time(f)$, where $time(f)$ is the time at which f occurs, as specified in the problem definition. These are then ordered with respect to the other steps in the plan by, again, adding sequencing constraints due to the causal relations between $lit(f)$ and the other steps in the plan.

In this paper, we focus on *online* planning. We want to account for the fact that time passes *during* the planning process, and that, in fact, planning time and execution time are both the same. In order to do so, we modify the STN described above by adding two additional time points: t_{PS} which is the time when planning started, and t_{now} which is the current time. We add the temporal constraint that $t_{now} - t_{PS}$ equals the currently elapsed time in planning. The expression $t_{ES} - t_{now}$ corresponds to the remaining planning time, which is, of course, unknown. We will discuss this expression, and how to treat it, in the next section. Now, $t_{PS} = 0$, while t_{ES} is unknown. Finally, because TILs describe absolute time, we must modify the temporal constraints corresponding to TILs to use t_{PS} instead of t_{ES}, i.e., the temporal constraint for TIL l would be $t(l) - t_{PS} = time(l)$, where $time(l)$ is the time at which l must occur.

Time-Aware Planning

We have described a technique for encoding an STN which captures the fact that execution only starts after planning ends, and planning takes time. We now describe the impact this has on search within a temporal planner.

Forward Planning Search Space

We take as our basis the forward-search approach of the planner OPTIC (Benton, Coles, and Coles 2012). Here, each search state comprises the plan π (of snap actions) that reaches that state; the propositions $p \subseteq F$ that hold after π was executed from the initial state; and the Simple Temporal Network $STN(\pi)$ encoding the temporal constraints over π.

When expanding a state in OPTIC, successors were generated in one of three ways:

- By applying a *start* snap-action that is logically applicable: any $a_\vdash$ where $p \vDash cond_\vdash(a)$; $\mathit{eff}_\vdash(a)$ would not break the invariant condition of an action that has started in π but not yet ended; and $cond_\leftrightarrow$ would be satisfied once $a_\vdash$ has been applied. In this case, in the successor state, $\pi' = \pi + [a_\vdash]$, p is updated according to $\mathit{eff}_\vdash(a)$ to yield p', and a variable $t(a_\vdash)$ added to $STN(\pi')$. Sequence constraints are imposed on this such that it follows any step in π that met one of $cond_\vdash(a)$; or whose effects refer to the same propositions as $\mathit{eff}_\vdash(a)$; or whose preconditions (including invariant conditions) would be threatened by $\mathit{eff}_\vdash(a)$[1].
- By applying an *end* snap-action that is logically applicable – any $a_\dashv$ where a has started in π but not yet ended; $p \vDash cond_\dashv(a)$; and whose effects $\mathit{eff}_\vdash(a)$ would not break the invariant of *any other* action that has started in π but not yet ended. In this case, the successor state is updated in a way analogous to starting an action, with the additional STN constraint $dur_{\min}(a) \leq t(a_\dashv) - t_{a_\vdash} \leq dur_{\max}(a)$.
- By applying a *Timed Initial Literal* l that has not already occurred in π. In this case, $\pi' = \pi + [l]$, p is updated according to $lit(l)$ to yield p', and a variable $t(l)$ is added to $STN(\pi')$. For the purposes of sequence constraints, this can be thought of as being a snap-action with no preconditions – it suffices to order it after steps in π whose preconditions or effects refer to $lit(l)$. To fix the time at which l occurs, an additional STN constraint is added: $t(l) - t_{PS} = time(l)$ – while snap-actions are ordered only relative to other points in the plan, TILs must also occur a specific amount of time after time zero.

State expansion in this way generates candidate successors that are logically feasible; to ensure they are also temporally feasible, only those whose STNs are consistent are kept. Using an all-pairs shortest path algorithm in the STN will both check consistency (with negative cycles corresponding to an inconsistent STN), and give us the earliest and latest possible time at which each snap-action could be applied. We denote these $t_{\min}(x)$ and $t_{\max}(x)$ for each STN variable $t(x)$. Typically, only the former of these is used – to map π to a schedule σ, each start–end snap-action pair $a_\vdash$, $a_\dashv$ gives a triple $\langle a, t_{\min}(a_\vdash), (t_{\min}(a_\dashv) - t_{\min}(a_\vdash))\rangle$. In other words, apply each action as soon as possible, with the shortest possible duration, thereby minimizing execution time.

Extending this approach to planning while aware of planning and execution time requires a number of modifications, which we now step through.

No action can start before plan execution starts – because execution cannot start until a plan has been produced. That is, for each $a_\vdash$ in the plan π, we add a constraint $t_{ES} \leq t(a_\vdash)$ to the STN, where t_{ES} is the time at which execution will start. We do not know this *a priori*, but can at least say $t_{now} \leq t_{ES}$ is the time since the planner started executing. An STN for a plan produced during successor generation will then be consistent *iff* it is not already too late to start executing the plan.

These additional constraints can be thought of as pushing the earliest actions in the plan to start after now; the effects of which are then propagated through the STN to appropriately delay the later actions, according to the sequence and duration constraints. If an otherwise-consistent STN is made inconsistent by these, then necessarily there must be a snap-action x where $t_{\max}(x) < t_{now}$ – i.e. we are past the latest point at which x could have been applied.

[1] As search progresses in a strictly forward direction, all threats are dealt with by *demotion* – ordering the new step after existing steps.

Planning time particularly matters in the presence of TILs – in the absence of these, we can start executing a plan whenever we like by simply delaying the start of the first action. If TILs are present, though, these anchor the plan to having to fit around absolute time: with reference to state expansion, when a TIL is added to the plan, this fixes it to come after any earlier steps with which it would interfere, thereby constraining their maximum time.

Automatically applying past TILs – if we are now past the time at which a TIL has occurred, it is added to π before expanding the state.

More formally, immediately before expanding a state $S = \langle \pi, p, STN(\pi) \rangle$, the following TILs are applied:

$$\{l \in T \mid t(now) \geq time(l) \wedge l \notin \pi\}$$

If there are several such TILs, they are applied in ascending order of $time(l)$. The mechanism for applying these TILs is identical to that in OPTIC: each is applied, to yield a successor state S'; and then S' replaces S. By doing this before expanding the state, we account for time having passed since S having been placed on the open list, and it being expanded – if in this time a TIL will have happened, S is updated accordingly, before expansion.

If this modification was not made, search would be free to branch over what step should next be added to π. In the case where a TIL l represents a deadline – by deleting a precondition on actions that must occur by a given time – search would be free to apply these actions, even though in reality it is too late. By forcing the application of past TILs, we avoid this behavior: all such actions would then become inapplicable.

Pruning states where it is too late to start their plan From the STN for a plan π, we can note the latest point at which that plan can start executing; and prune any states for which this time has already passed.

As noted earlier, to check if the STN for a state is consistent, we use an all-pairs shortest path algorithm. This incidentally yields the minimum and maximum time-stamps for each snap-action. For snap-actions that are ordered before a TIL – which are fixed in time – these maximum time-stamps are finite. Moreover, because the plan is expanded in a strictly forward direction, the maximum timestamps are monotonically decreasing: it is not possible to somehow order a new action before a plan step, in a way that reduces its maximum time-stamp. Thus, for each state $S = \langle \pi, p, STN(\pi) \rangle$ we identify the start snap-action in π that has the earliest possible maximum time-stamp – this is the latest time at which π could feasibly be executed:

$$latest\_start(\pi) = \min\{t_{\max}(a_{\vdash}) \mid a_{\vdash} \in \pi\}$$

Then, when S is about to be expanded – after it was generated, placed on the open list, and then removed – it is pruned if $t_{now} > latest\_start(\pi)$.

Experiments

To gain a concrete sense of the practical import of our technique, we experimentally compared it to the baseline method of prespecified planning times. We performed experiments in two types of domains: a realistic AUV simulation, and a set of IPC domains.

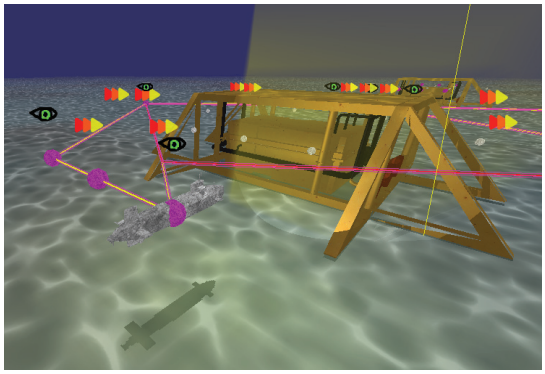

Figure 1: Screenshot of the underwater simulator, in which the AUV is inspecting the structure.

As a baseline against which to compare our time-aware planner, we used OPTIC in optimization mode, searching for the best plan within a varying fixed planning time of T seconds. Time windows were considered to be T seconds earlier, to adjust the initial state to the start of execution time. Therefore, a TIL l occurring at time $time(l)$ seconds, using a planning time of T seconds, will occur at time $(time(l) - T)$ (at least 0) in the plan.

AUVs

We demonstrate the approach in simulation with autonomous underwater vehicles (AUVs). We embed OPTIC and our planner into ROS, using ROSPlan (Cashmore et al. 2015), to control the AUV. The AUV is equipped with a manipulator and placed in an underwater structure, with the task to inspect certain areas and to ensure that valves are turned to correct angles. The valves can only be turned within certain time windows, outside of which the valve is blocked. If the valve cannot be turned to the correct angle within an early time window, then a later window can be used. We generated 41 missions with varying time windows. A screenshot of the simulation is shown in Figure 1

These missions normally form part of a larger, strategic mission, spread out over a number of seabed manifolds. The AUV moves between these manifolds in order to complete the missions. Due to the uncertainty in the environment, it is not known beforehand precisely what time the AUV will arrive at the manifold. Before beginning the task, the AUV must construct a new plan. Plans with shorter durations are considered to be of higher quality, as this eases the time constraints on the remainder of the missions. We use this scenario to show that our approach allows the AUV to make use of earlier time windows, generating plans of higher quality.

The results are summarized in Table 1. The table shows the number of problems solved for each planner, out of a possible 41. Using our approach every problem was solved. Using a fixed planning time, some problems were unsolvable due to a planning time that was too short. The table

	Time Aware	OPTIC$_{50}$	OPTIC$_{100}$	OPTIC$_{200}$
best quality	34	13	20	19
IPC quality	40.19	25.55	26.19	26.47
problems solved	41	26	34	40

Table 1: Table comparing the number of problems solved, the number of plans of highest quality, and the IPC quality for each approach.

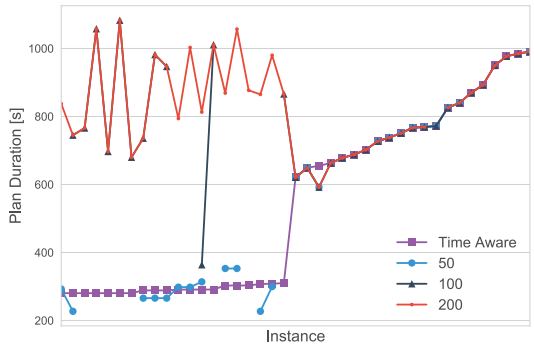

Figure 2: Plan durations per problem for each approach. The time-aware approach solves many problems using an earlier time window. OPTIC using a long planning time solves almost every problem, but only using the later time windows. Other planning time bounds are less reliable.

also shows the number of best plans for each approach. This is the number of problems for which that approach produced the plan of highest quality between the four approaches (possibly jointly). There it can be seen that although increasing the planning time allows for all problems to be solved, the quality is much poorer. The higher absolute number of best plans for the 200 second planning time is due to the greater number of problems solved. Finally, the table shows the IPC quality, calculated for all problems. These results demonstrate the choice between acting quickly, utilizing early time-windows, or producing plans reliably. Using the time-aware approach does both.

This can be seen more clearly in Figure 2. This figure compares the plan duration from each approach per problem. Using OPTIC$_{200}$ almost every problem is solved, at the longest possible plan duration – assuming planning takes 200 seconds forces the planner to have to use the later time windows. Other approaches may generate shorter plan durations, but fail to solve many of the problems.

IPC Domains

In our IPC experiments, we tested all IPC-4 and IPC-5 domains that contain TILs: airport, pipesworld, satellite, truck, and UMTS. The UMTS domains and half of the airport instances were omitted as none of the planners completed these. The planners were given a maximum of $200s$ of CPU time and 4GB of memory.

	Time Aware	OPTIC$_{0.1}$	OPTIC$_{1}$	OPTIC$_{10}$
best quality	38	1	0	1
IPC quality	38	9.99	29.74	19.89
problems solved	38	10	30	21

Table 2: Table comparing the number of problems solved, the number of plans of highest quality, and the IPC quality for each approach

Table 2 presents results on the modified IPC domains. The fixed planning time planners were outperformed by the time-aware methods in every domain. Several instances were unsolvable by the former due to the fixed planning time constraints. Table 3 shows the planners detailed performance in each relevant domain tested.

In addition to the fixed planning times that are showed in Table 2 and Table 3 we have tested $50s$, $100s$, and $200s$. The performance of the baseline approach with these planning times were lower than the time-aware method and the best presented baseline, thus these results were omitted.

Conclusions and Future Work

We have presented a domain-independent temporal planner that takes the interaction between the time spent on planning and execution time into consideration. We have demonstrated empirically that this planner achieves much better results in domains with absolute deadlines than our baseline approach. However, our work is merely the first step in addressing this important topic. There remain many exciting avenues for future work.

For example, our planner only looks at the current partial plan, and uses a heuristic to "look" into the future. This heuristic is used to estimate the remaining search depth, but not to obtain more information about future actions and their effects on deadlines. In order to get a more informed view of future actions, and their effect on deadlines, we will explore using temporal landmarks (Karpas et al. 2015). These landmarks could be encoded into the same STN of the partial plan, and thus we believe we will be able to achieve even better pruning of branches of the search tree which will not lead to a solution in time.

More broadly, the problem we are addressing here could benefit from more explicit metareasoning (Russell and Wefald 1991). For example, suppose we had a planning problem with two possible solutions, each of which must be explored on a separate branch of the search tree. Further suppose that each of these solutions has a deadline which leaves just enough time to explore one of the branches, but not both of them. Clearly, a planner with perfect knowledge would choose one of these branches and explore it. On the other hand, the approach we present here will explore both branches until it realizes there is not enough time left, and will then prune both branches — without solving the problem. In future work, we will explore ways of addressing this type of problem by incorporating explicit metareasoning on

group	planner	solved	time	GAT
airport-1	**Time Aware**	**14**	6.62	193.54
	OPTIC$_{0.1}$	2	0.06	89.61
	OPTIC$_{1}$	10	0.24	167.72
	OPTIC$_{10}$	10	0.20	176.72
pipesworld	Time Aware	3	0.72	16.06
	OPTIC$_{0.1}$	1	0.05	12.11
	OPTIC$_{1}$	**4**	0.51	15.51
	OPTIC$_{10}$	0		
satellite-1	**Time Aware**	1	0.03	190.23
	OPTIC$_{0.1}$	1	0.04	190.31
	OPTIC$_{10}$	1	0.02	200.21
	OPTIC$_{1}$	1	0.02	191.21
satellite-2	**Time Aware**	**5**	0.71	181.89
	OPTIC$_{0.1}$	1	0.03	190.31
	OPTIC$_{1}$	4	0.39	182.87
	OPTIC$_{10}$	1	0.56	129.16
satellite-3	**Time Aware**	**5**	0.80	181.88
	OPTIC$_{0.1}$	1	0.03	190.31
	OPTIC$_{1}$	4	0.36	182.87
	OPTIC$_{10}$	1	0.56	129.16
satellite-4	**Time Aware**	**4**	2.20	165.20
	OPTIC$_{0.1}$	0		
	OPTIC$_{1}$	2	0.15	155.00
	OPTIC$_{10}$	2	1.38	147.38
truck	**Time Aware**	**6**	0.21	1840.98
	OPTIC$_{0.1}$	4	0.05	1673.95
	OPTIC$_{1}$	5	0.06	1674.20
	OPTIC$_{10}$	**6**	0.20	1855.97

Table 3: Table comparing the number of problems solved, the planning time, and the goal achievement time (GAT) grouped by IPC instance type. The planning time, and the GAT is the mean of all instances in the group solved by the planner.

planning time allocation into the search strategy.

One possible approach for this would be to treat the expression $t_{ES} - t_{now}$ as a variable, which we will denote by *slack*. We can then treat the STN as a mathematical optimization problem, and maximize the slack. The slack for node n can serve as a proxy for the probability of finding a solution in time in the subtree rooted at n. Our metareasoning algorithm could then choose the next node to expand based on both heuristic estimates and the slack.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No. 730086 (ERGO).

References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Burns, E.; Ruml, W.; and Do, M. B. 2013. Heuristic search when time matters. *Journal of Artificial Intelligence Research* 47:697–740.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 333–341.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence* 173(1):1–44.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: planning with continuous linear numeric change. *Journal of Artificial Intelligence Research (JAIR)* 44:1–96.

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.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 1852–1859.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 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 *Proceedings of the Symposium on Combinatorial Search (SoCS-11)*.

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.

Fernández-González, E.; Karpas, E.; and Williams, B. C. 2015. Mixed discrete-continuous heuristic generative planning based on flow tubes. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 1565–1572.

Fernández-González, E.; Karpas, E.; and Williams, B. C. 2017. Mixed discrete-continuous planning with convex optimization. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 4574–4580.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI)*, 608–620.

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.

Karpas, E.; Wang, D.; Williams, B. C.; and Haslum, P. 2015. Temporal landmarks: What must happen, and when. In *Pro-*

ceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS), 138–146.

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

Ruml, W.; Do, M. B.; Zhou, R.; and Fromherz, M. P. J. 2011. On-line planning and scheduling: An application to controlling modular printers. *Journal of Artificial Intelligence Research* 40:415–468.

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