Temporal Concurrent Planning with Stressed Actions

Fabian Peller, Mirko Wächter, Markus Grotz, Peter Kaiser and Tamim Asfour

***Abstract*— Temporal stress is something that humans have to face nearly every day. Humans have to handle situations, where there is not much time left for a specific task. Most robotic systems, on the other hand, are not able to act in such temporally unstructured environments. For that reason, we present the novel *Temporal Stressing Fast Downward* (TSFD) planning system based on *Temporal Fast Downward* (TFD), which solves temporal problems using a modified heuristic forward search. With this planner, we introduce the novel concept of *stressed actions* for temporally bounded problems. Stressed actions enable a robot to accelerate or decelerate actions under consideration of an action-specific temporal cost function and the available time for plan execution. We further introduce an improved decision epoch search that allows complete planning with temporal gaps. Our evaluation in benchmark domains and on the real humanoid robot ARMAR-III shows that TSFD has the ability to produce plans of better makespan than TFD and is able to solve problems that could not be handled before. Furthermore, TSFD performs better in typical service robotics tasks than baseline approaches. Finally, we show that stressed actions greatly increase the possibility of finding feasible solutions in temporally bounded tasks.**

I. Introduction

In situations where humans have to solve complex tasks in short amounts of time, they feel under pressure and get stressed. In [1], Lazarus and Folkman describe human stress as an "*internal state which can be caused by physical demands of body or by environmental and social situations, which are evaluated as potentially harmful, uncontrollable, or exceeding our resources for coping*". Human beings are able to perform well under temporal stress, because they can estimate the duration of sequences of activities and consider these estimates during planning. When exposed to temporal stress, humans commonly accelerate their planned actions in the intend of achieving the goal within the required deadline. However, accelerated plan execution often comes with increased risks of failure. This motivates our assumption that the questions *if* and *how much* planned actions need to be accelerated is a tradeoff between the proximity of the temporal deadline and the tolerated risk of failure. The ability of planning under temporal stress would allow robots to perform tasks for which cost or risk requirements may be bended through temporal acceleration of actions in order to meet a defined deadline. Moreover, the possibility to define temporal deadlines would potentially ease the communication between humans and robots.

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the FET Proactive grant agreement No 641100 (TIMESTORM).

The authors are with the High Performance Humanoid Technologies Lab, Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology (KIT), Germany, {fabian.peller, asfour}@kit.edu

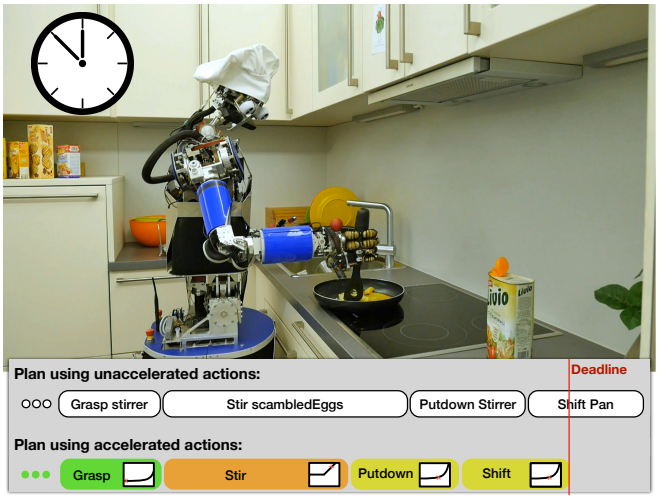


Fig. 1: The humanoid robot ARMAR-III [2] cooking scrambled eggs with a hard temporal deadline. The upper row depicts an insufficient plan without accelerated actions. The lower row shows a plan with accelerated actions that meets the deadline. Accelerating actions has the disadvantage of causing increased risk or cost. These acceleration effects are pictured in each action's cost function and color (green means low cost, red means high cost). The actually planned acceleration is marked with a red cross in each cost function.

An artificial planning system, which not only performs classical planning (*which actions should be executed?*) but also considers temporal information (*when should actions be executed?*) is called a temporal planner [3]. Temporal planning has been a long-standing part of the International Planning Competition (IPC) [4]. However, in [5], Cushing et al. showed that most of these planners share the ability to produce high-quality but still suboptimal plans to temporal expressive problems with required concurrency. Cushing et al. attribute this to the incomplete *Decision Epoch* (DE) planning approach. DE-planners are only able to schedule new actions at the start of the problem or when another action has finished, which minimizes branching to a small set of decision nodes instead of allowing actions to be scheduled at any point in time. Plans that must include *temporal gaps* [5] cannot be found with this approach. Nevertheless, DE-planners performed well on most tracks of the IPC. They benefit from the fact that discretizing time into decision epochs greatly decreases the complexity of the search space and makes the application of heuristics and techniques from classical planning feasible.

In this work, we propose an extension of the DE-planning

approach, which allows newly scheduled actions to not only start synchronously, but also end synchronously to other scheduled actions. This approach still has the advantage of limiting scheduling to a small number of decision epochs, while enabling complete planning. Our proposed approach is further capable of planning under temporal stress, which implies that the planner is aware of action durations, action accelerations and the potential effects of action acceleration.

Fig. 1 visualizes the process of action planning under temporal stress in an experiment using the humanoid robot ARMAR-III: The robot is given the task to cook scrambled eggs within ten minutes. Classical planners would fail to find a solution if none exists using unaccelerated actions. The proposed planner TSFD is capable of accelerating actions under consideration of measures of cost or risk, and would therefore be able to find suitable solutions that meet the defined deadline. To achieve the goal within the desired ten minutes, the robot could e. g. highly accelerate the `Stir` action and slightly accelerate the `Putdown` and `Shift` actions.

The remainder of this paper is structured as follows: We first introduce the formalisms of temporal concurrent planning in section II. Afterwards, we explain the similarities and differences to related approaches in section III. We present our complete DE planning approach and describe stressed action planning in section IV. Section V covers the evaluation and discussion of the proposed planner. Finally we conclude the paper and provide an outlook on future work in section VI.

II. An Introduction to Temporal Planning

Temporal planning can be seen as a generalization of classical planning. In addition to the defined state fact list $state(s)$, a *time-stamped state* s contains a specific time-stamp and a list of currently scheduled actions. *Scheduled actions* are actions that have been started but are not yet finished. A *durative action* a not only needs a list of *at-start* preconditions which must be fulfilled before an action starts and *at-end* effects which come into force after the action, but also requires *overall* preconditions which have to be satisfied during the entire operation and *at-end* preconditions which must be fulfilled when the action ends. Moreover, a durative action could have a list of *at-start* effects which are applied after the action starts and a specific duration.

Because TSFD bases on TFD [3], it does not use PDDL formulations of the planning instances directly. It preprocesses the PDDL definitions into a formalism called temporal numeric SAS+ (for *Simplified Action Structures*) [3][6]. SAS+ uses multi-valued state variables and handles logical implications and arithmetic subterms via axioms. Axioms are useful to minimize complex operations such as quantifiers or derived predicates [7]. A temporal planning task can be seen as a tuple $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{A}, \mathcal{O} \rangle$, where:

- $\mathcal{V}$ is a set of time-stamped states,
- s_0 is the initial state assignment of the problem,
- s_* is a partial assignment defining the goal state,
- $\mathcal{A}$ is a set of axioms,
- $\mathcal{O}$ is a set of durative actions.

TFD accepts all features of PDDL 2.1 except user-defined metrics and duration inequalities.

A solution to a temporal planning task is a collection of *plan steps*, where each plan step $\mathcal{S} = \langle a, t_s, t_e \rangle$ contains a durative action $a \in \mathcal{O}$ to execute, a timestamp t_s which indicates when to execute that action and a timestamp t_e which indicates when this action ends. The planner guarantees that each action of the solution plan is applicable at its starting point. The final state after applying all plan steps must satisfy the goal assignment s_*.

A plan is called *concurrent* if at least two actions are planned to be executed concurrently. Solvable problems require concurrency if all possible solutions are concurrent. A problem is called *temporally expressive* if any solution to the problem contains *temporal gaps* [5]. A planner is called *complete* with respect to an action language $\mathcal{L}$ if for all problems expressible in $\mathcal{L}$, the planner guarantees to find a solution if such a solution exists.

In order to prune away bad decisions at an early stage and to guide the search, modern planners usually employ heuristics. So do temporal planners, for which the possibility of action scheduling over time potentially increases the branching factor to infinity. A heuristic $h : \mathcal{V} \to \mathbb{N}_0$ ideally approximates the optimal heuristic h^*, which returns the real distance from a state $s \in \mathcal{V}$ to the goal (Note that temporal planners usually estimate the remaining duration of the plan, called *remaining makespan*). If there is no path to the goal, h^* returns ∞.

III. Related Work

Since 2002, temporal planning systems participated next to classical planning systems at the IPC. Temporal planning capabilities were introduced with PDDL 2.1 [8], an extension to PDDL [9] for classical planning systems. The first successful temporal planners of the temporal IPC tracks ignored all temporal aspects and used classical planners like Metric-FF [10] to find sequential paths to the goal (e. g. SGPlan [11], MIPS [12] or LPG-td [13]). The scheduling of actions was performed in a decoupled step independently of the planning. These planners are not optimal for problems which require concurrency. Planners that integrate scheduling into the planning phase often produce higher-quality plans regarding makespan, because they also consider plans that cannot be rescheduled into sequential solutions. SAPA [14], TP4 [15], TALPlan [16], TLPlan [17], SGPlan [11], TFD [3], CRIKEY [18] or OPTIC [19] follow this approach. The planner CRIKEY [18] actually uses a mixed approach by employing fast classical search algorithms to find sequential arrangements of actions and switching to a temporal approach if no sequential solutions could be found [5].

Since more and more complex temporal problems become solvable, it becomes apparent that plan duration is not the optimal metric for plan quality. For instance, TFD uses a special representation of its closed list to find paths back to the initial state allowing to sum up duration and cost of actions in order to calculate heuristic values for the weighted

```
(:action transport-fruits
   :parameters(?from ?to - location)
   :condition (and (truck-at ?from))
   :effect (and
      (truck-at ?to)
      (when (> (current-time) (final-deadline))
         (increase (total-cost) (full-penalty)))
      (when (> (current-time) (deadline-one))
         (increase (total-cost) (part-penalty)))
   )
)
```

Fig. 2: A time-dependent action definition in PDDL 3 as used in OPTIC [19]

A^* algorithm. However, due to DE-planning, TFD is not able to return optimal solutions for temporally expressive action languages. DE-planners distinguish between scheduling epochs (also known as *fattening epochs*) and execution epochs (also known as *advancing time epochs*). Actions can only be scheduled during scheduling epochs and scheduled actions only get executed in execution epochs.

Cushing et al. [5] describe another approach called *Temporally Lifted Progression Planning* where they delay the decisions about when to execute until all decisions about what to execute have been made. Instead of scheduling an action s to the current timestamp their algorithm `TEMPO` uses a list of temporal constraints for each lifted state $\mathcal{N}$ of the form $\tau_{begin}(s) \geq \tau(\mathcal{N})$ where τ_{begin} denotes the start time of the action and τ the current time of $\mathcal{N}$. When advancing time, a new constraint $\tau_{end}(s) \geq \tau(\mathcal{N})$ gets added to the lifted state before simulating the execution of the next action. Just before terminating, `TEMPO` must actually pick some particular assignment of times satisfying $constraints(\mathcal{N})$ for a goal assignment $\mathcal{N}$. This technique allows `TEMPO` to use a higher representation of time than the explicit one used by other DE-planners.

OPTIC [19] uses mixed integer programming methods instead of linear scheduling to arrange actions during planning. To the best of our knowledge, OPTIC is the only modern planner which is able to make use of *time-dependent* costs during planning using soft deadlines. To do so it employs a special state variable `current-time` accessible in PDDL 3 and the PDDL conditional `when` to define functions between two deadlines in specific cost evaluation action as shown in Fig. 2. The state variable `current-time` is updated continuously once per time unit using a process with no conditions and the effect `(increase (current-time) (* #t 1))`. The PDDL instruction `sometime after f g` guarantees that the final plan contains the action `f` before its cost evaluation action `g`. Unfortunately, this approach cannot be used to adequately represent acceleration and deceleration, because `sometime after` provides no boundaries for a minimum or a maximum duration.

IV. Approach

With TFSD, we implemented a novel, complete planner that is able to handle continuous duration-cost relations during planning to find solutions to problems with deadlines. In the following, we will first introduce the changes in PDDL which allow us to specify the required properties for stressed actions. After that, we will describe how to discretize stressable actions into stressed actions and compare our improved DE-planning procedure to conventional DE-planning. Stressable actions are high-level continuous actions with known acceleration functions, fixed acceleration bounds and one discretization value per stressable action. The stressable action gets discretized into stressed actions with a fixed acceleration and, therefore, fixed cost. Finally, we will explain how TSFD works inside the robot software environment ArmarX [20].

```
(:durative-action grasp
     :parameters(?x - robotHand
          ?y - location
          ?z - obj)
     :costfunction (= (-2 * a^2) +8)
     :minacceleration (= 0.2)
     :maxacceleration (= 1.5)
     :discretizations (= 5)
     :duration (= ?duration 2)
     :condition (and (over all (handEmpty ?x))
          (over all (objAt ?z ?y)))
     :effect (and (at end (inHand ?x ?z))
          (at end (not (handEmpty ?x)))
          (at end (not (objAt ?z ?x))))
)
```

Fig. 3: Our modified PDDL 2.1 action definition. New tags are highlighted in green.

A. Planning with stressed actions

We introduce an extension of the PDDL action definition that allows specifying a manually defined cost function in relation to the acceleration of the action. This cost function can be used by the planning algorithm to find solutions with accelerated execution but higher costs or risk. These criteria are usually anti-proportional and the search algorithm has to find the cheapest solution while meeting the hard deadline. The definition of the cost function is context-dependent and could e. g. express the estimated energy consumption or risk of an action depending on the acceleration factor. Given several measurements at certain accelerations, one can define a continuous acceleration function, which interpolates at unknown values.

1) Modified PDDL syntax: We augmented the PDDL syntax to also accept continuous function strings of the form $y = f(A_a) : A \in \mathcal{O}$ within a specific `:costfunction` tag, two single float values within a `:minacceleration` and a `:maxacceleration` tag and an integer value within a `:discretizations` tag. The function definitions depend on a factor $A_a \in (0, \infty)$, called the *acceleration factor* for an action $A \in \mathcal{O}$. To the best of our knowledge, there is no possibility to define continuous functions in plain PDDL 2.1 without duration inequalities. While the actual duration of each stressed action gets divided by A_a to get the accelerated duration, y denotes the used cost for that action and that acceleration. Fig. 3 shows our changes compared to an original PDDL 2.1 action definition. New tags are highlighted in

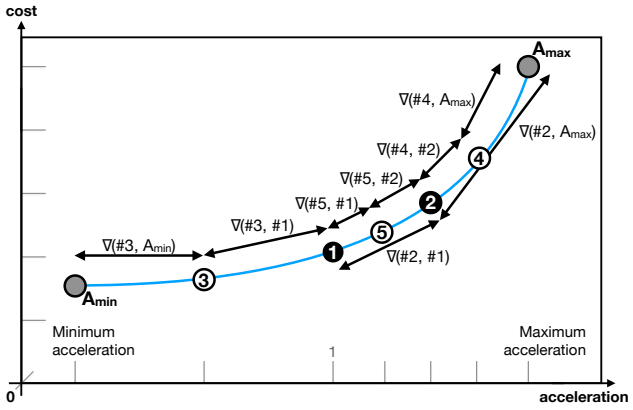


Fig. 4: The way how TSFD discretizes stressable actions. Black filled circles depict already accepted accelerations and white filled circles show open candidates. Gray filled circles picture the minimum and maximum acceleration. The arrows above each candidate show the gradient to the next accepted acceleration or boundary. The blue graph depicts the cost function from its minimum acceleration up to its maximum acceleration.

green. The usage of a mathematical expression parser [21] during planning allows us to use mathematical operators, functions, loops or variable definitions. Control structures such as `if-then-else` allow different functions on different intervals. Unlike OPTIC, TSFD uses two boundaries to limit the acceleration. These values assure that our planner does not use accelerations faster than `maxacceleration` or slower than `minacceleration`.

2) Discretization into stressed actions: Since the cost functions are continuous, there is an infinite number of possible action accelerations. However, the employed search algorithm works on discrete actions. Therefore, TSFD discretizes each stressable action into up to n actions with fixed acceleration factors. The parameter n is specified within the `:discretizations` tag.

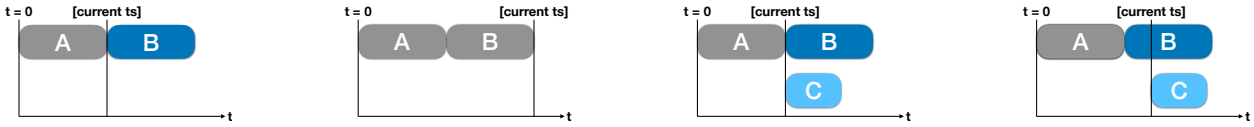
Although PDDL allows duration-changes of actions during planning by using state-dependent durations, we assume that each stressable action A has a fixed duration (most problems of the IPC do so, too). TSFD discretizes A's cost function only once at the initialization of the algorithm, creating several stressed actions with fixed acceleration factors A_a and, therefore, fixed durations $duration(A)/A_a$ and fixed costs $f(A_a)$. These discretized actions are stored within a list of accepted operators which TSFD later uses for planning. Because we only want to discretize actions in interesting areas, the planner uses a binary approach similar to [22]. Fig. 4 depicts the process of discretization: For each action A it uses the default acceleration $A_a = 1$ (node with ID 1 in Fig. 4) as accepted root for the binary tree and adds its both neighbors $A_a^{faster} = (A_a + A_{max})/2$ (node with ID 2) and $A_a^{slower} = (A_a + A_{min})/2$ (node with ID 3) to a list of discretization candidates. A_{min} and A_{max} describe the minimum accelerated action and the maximum accelerated action, respectively. Afterwards, TSFD searches for the candidate C with the highest information gain about the duration-cost graph (maximum of distances of $f(C_a)$ to the next slower and the next faster already accepted action or boundary) which gets added to the list of accepted operators. In the example in Fig. 4, node 2 is already classified to have a higher information gain than node 3.

After accepting this action, its both neighbors C_a^{faster} (node with ID 4) and C_a^{slower} (node with ID 5) are added to the list of candidates. We repeat this procedure until all candidates have an information gain $< \epsilon$ or until we have n discretized operators. The list of candidates ensures, that old neighbors, which have not been accepted in the first place are still involved during the search for the best candidate and may be accepted later.

3) Search with stressed actions: TSFD offers several different modes of search. The first one is the default for temporal problems: *minimizing makespan*. Therefore it only uses heuristics which estimate remaining makespan. The second one is *minimizing a cost sum*. This mode is useful for stressed action planning with cost such as energy consumption, where the robot has a hard deadline but tries to find a solution which is as cheap as possible within this deadline. Finally, the last mode is *minimizing a cost product*. Like minimizing a cost sum, this mode is useful for stressed action planning, but computes the product of all costs instead of the sum. This is useful for multiplicative cost values, such as probabilities.

B. Improved decision epoch search

The second contribution of this paper describes a modified DE search method to even find solutions to any kind of temporally expressive problems. Other than usual DE-approaches, where a new action can only be scheduled in the initial state or when another action ends, we added further decision epochs to our planning procedure so that a new action can additionally be scheduled for a starting time in the future (called *future scheduling*) or in the past (called *past scheduling*) so that it ends synchronously to other scheduled actions. In order to minimize branching, TSFD only allows actions to be scheduled to end simultaneously with the next coming decision epoch, which is always the earliest end of all scheduled actions. In the following, we will explain, how TSFD uses future- and past scheduling.Therefore, we assume that TSFD starts expansion on a specific state s with an already scheduled action B and an executed action A. Δ_{DE} denotes the difference of timestamps of s to the next decision epoch. Figure 5 (c), (d) and (e) shows all possible derived states during the search of our planning procedure after scheduling an action C with a shorter duration than Δ_{DE} to s. Subfigures (f), (g) and (h) depict all possible derived states after scheduling an action D with $duration(D) > \Delta_{DE}$ to s. All states with a different timestamp than s are either future- or past scheduled states (except (b)). It is impossible to schedule actions to start earlier than 0, which is why we assume the already executed action A. Note that past- and future scheduling only makes sense, when there is another

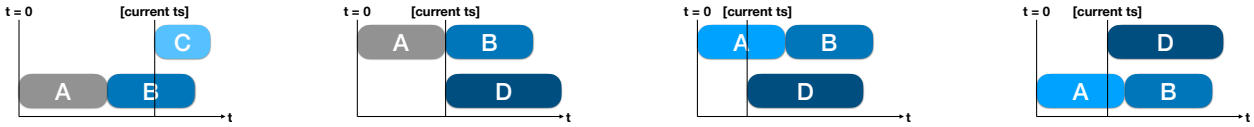


(a) The assumed state before scheduling any action. This state contains a scheduled action B. Executing action A in the initial state leads to this state.

(b) The first state derived after executing the scheduled action B.

(c) A new state derived after scheduling action C to begin shortly after the scheduled action B.

(d) A new state derived after future scheduling C so that it ends shortly before B ends.



(e) A new state derived after future scheduling C so that it ends shortly after B ends.

(f) A new state derived after scheduling action D to begin shortly after the scheduled action B.

(g) A new state derived after past scheduling D so that it ends shortly before B ends.

(h) A new state derived after past scheduling D so that it ends shortly after B ends.

Fig. 5: All derived new states after scheduling a new action C and a new action D to a state with an already scheduled action B and executed action A (expanding a state with two different actions). Here $duration(C) < \Delta_{DE}$ and $duration(D) > \Delta_{DE}$. *Current ts* denotes the timestamp of the generated state. Actions with gray colors depict already executed actions.

scheduled action. While scheduling a new action at the current timestamp s does not change the timestamp of the derived state, scheduling an action to some point in the future always increases the timestamp of the derived state by a value > 0 and scheduling an action to the past will decrease the timestamp by a value > 0. Subfigures (b), (c), and (f) depict normal DE expansion, where either the already scheduled action B gets executed or where action C respectively D gets scheduled to start immediately after A (synchronously to B).

Because we modify the timestamp of s when scheduling action C to the future or action D to the past, we always need the knowledge of how the derived state will look like or looked like at that specific time. The first question is very easy to answer because we only allow actions to be scheduled to a timestamp between the current and the next decision epoch. The definition of decision epochs disallows state changes between two of those, so the new state will be equal s, except for the increased timestamp and additionally scheduled action and its at-start effects. To solve the latter, we use the knowledge of predecessor states. Whenever we schedule action D to a timestamp t_{past} in the past, we backtrack to the first state s_{pre} with $timestamp(s_{pre}) < t_{past}$ and future schedule D to s_{pre}. We do not care if this means that other already scheduled actions gets deleted during backtracking (if they were scheduled after t_{past}), because they may get scheduled again when expanding the new state.

Unfortunately, this method disables TFDs heuristic H^{CEA}, which is temporally incomplete. It classifies states to be dead-ends if the state has scheduled concurrency and at least two scheduled actions have mutually dependencies. In the following, we will call this kind of states *temporally expressive states*. In order to still keep temporally expressive states while searching for solutions with minimal makespan, TSFD uses a temporal version of H_{add} based on the relaxed temporal planning graph [14] next to H^{CEA}. The new heuristic keeps the information about action durations while H^{CEA} is used to guide the search for minimizing makespan. For solutions with minimal cost, TSFD uses cost estimations instead of makespan ones. We modified h^{CEA} so that it also returns cost estimations if required.

C. Integration into the robot software environment ArmarX

We completely integrated TSFD into ArmarX [20]. This robot-agnostic software framework was developed to ease the realization of higher-level capabilities needed by complex robotic systems such as humanoid robots. To use a symbolic planner with a mobile robot in a dynamic environment, it is necessary to perceive the state of the robot and environment and maintain a consistent state representation. The memory architecture of ArmarX, called *MemoryX*, provides these capabilities and is used to generate a domain and problem description for TSFD.

Each symbolic action of TSFD is produced by *Object Action Complexes* (OACs) [23] stored in the long-term memory of the robot. An OAC is a triple (E, T, M) where:

- E is an identifier for an execution specification,
- $T : S \rightarrow S$ is a prediction function defined on an attribute space S encoding a model of how the world (and the agent) will change if the execution specification is executed, and
- M is a statistical measure representing the success of the OAC in a window over the past, e.g. failure rates.

We augmented M to additionally store values for the lowest and the highest acceleration and the cost function. Both, T and M (even with our extensions), can be automatically extended and updated using machine learning methods and

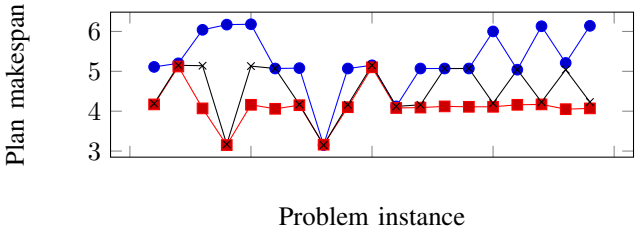


Fig. 6: Results of TSFD (red squares) in comparison to TFD (blue circles) and the best solution so far (black crosses) on the parking challenge.

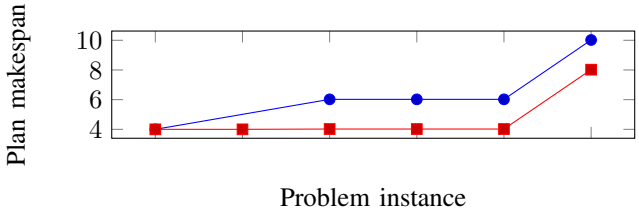


Fig. 7: Results of TSFD (red squares) in comparison to TFD (blue circles) on temporally expressive domains. A missing node indicates, that the planner was not able to find a solution.

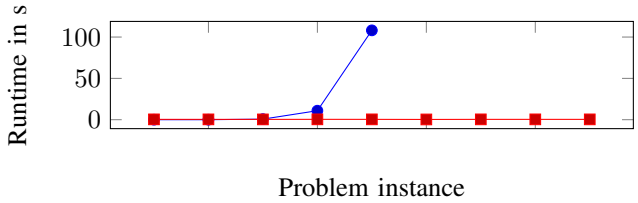


Fig. 8: Benchmark of TSFD (red squares) in comparison to the PKS planning system (blue circles) on the *wiping* and the *graspAndPutdown* tasks.

plan execution monitoring. The stored measurements in M can be used to automatically extract acceleration functions.

Based on OACs and the perceived state of the environment, ArmarX produces PDDL code using its sub-framework *Spoac* for symbolic planning with OACs [24] [25]. Once a solution is found, ArmarX executes the plan and monitors the action executions for failed actions. In case of failures, Spoac triggers a search based on the currently perceived world state.

V. Evaluation

We carried out four different evaluation experiments. First, we evaluated TSFD against TFD, showing that TSFD is able to produce plans of higher quality. Second, we used real-world benchmarks from ArmarX and measured the runtime of our planner to find a solution in comparison to PKS [26], the previous planning system of ArmarX. Third, we evaluated planning with stressed actions in typical service robotics domains. Finally, we evaluated the entire system using the humanoid robot ARMAR-III in the exemplary task of the preparation of scrambled eggs using TSFD and stressed actions. For all benchmarks except the real robot experiment, we used an Intel Core i5 CPU with 8GB RAM storage running Ubuntu 14.04. We counted the best solution which the planners returned within $300s$. To make sure that all returned PDDL plans of IPC domains are valid we used the plan validation tool VAL [27].

A. Evaluation of TSFD against TFD

For our first evaluation, we use an official challenge of the IPC 2014, where TFD failed to find the best solution, namely the *Parking* challenge. During the IPC 2014 TFD constantly returned suboptimal results on this specific domain. Unfortunately, all challenges of the IPC are temporally simple [5], which is why we also use a second benchmark containing only temporally expressive domains first designed by [5]. The PDDL descriptions of the temporally expressive domains can be found at https://goo.gl/RbVf5V. In [3], Eyerich et al. already measured the runtime of TFD on IPC benchmarks, showing that it is able to compete with other state-of-the-art temporal planning systems.

Fig. 6 shows the results of our planning system against TFD on the parking domain. TSFDs modified planning procedure was able to produce plans of higher quality than TFD. On several domains, TSFD returned even better plans than the best participant of the IPC. Fig. 7 shows the results of TSFD and TFD on temporally expressive domains. While TSFD always found the best solution (the one with temporal gap and required concurrency), TFD failed on all domains, but the result also shows, that up to some degree TFD is able to find plans with required concurrency. Both engines explored the whole state space during the five minutes of our benchmark.

B. Evaluation of TSFD against PKS

For our evaluation of TSFD against the PKS planning system, we use custom domains as they are used in ArmarX. We distinguish between a *wiping* task and a *graspAndPutdown* task, both containing several problem definitions. Both describe typical tasks for service robots. For the wiping domain, our simulated robot ARMAR-III first has to request a sponge from a nearby human. Afterwards, it has to move to several locations and clean them using a `Wipe` action. For the graspAndPutdown problems, ARMAR-III has to grasp objects and put them to specific locations. The results are shown in Fig. 8. Both tasks returned almost similar results, hence we used only one graph to depict the results. In all instances TSFD significantly outperformed the PKS planning system, which completely failed for plans with length > 8.

C. Evaluation of planning with stressed actions

To show that TSFD is able to find plans with accelerated actions we use the same wiping task as in section V-B. For each problem and action, we use up to $n = 10$ discretized operators. In [28] [29], Maniadakis et al. measured the minimum and maximum execution time as well as the robustness of each relevant action of the humanoid robot

Name	Duration [s]	acc_{min}	acc_{max}	$f(a)$
`Move`	13.5	0.4	1.6	$f(a) = a$
`Request`	28.5	0.8	1.2	$f(a) = 5a$
`Wipe`	19	0.7	1.3	$f(a) = 3a$
`Grasp`	25.5	0.4	1.6	$f(a) = 3a$
`Putdown`	29	0.4	1.6	$f(a) = 3a$

TABLE I: Relevant actions for the service robot challenges with corresponding acceleration and cost values [28] [29].

Problem	Complexity	Deadline [s]	makespan [s]	cost	acc_{avg}
1	1 goal	no	84.8736	6.8875	0.704
1	1 goal	61.5	60.9252	8.825	1.058
1	1 goal	50	49.0959	11	1.28
2	2 goals	no	135.202	9.65	0.68
2	2 goals	94	93.517	12.65	1.07
2	2 goals	75	73.9364	16.125	1.305
3	3 goals	no	185.53	12.4125	0.669
3	3 goals	139.5	139.248	15	1.007
3	3 goals	110	108.752	20.5	1.207

TABLE II: Problem instances of the wiping challenge and TSFD's results.

ARMAR-III in simulation. For our test we use the same durations for highest or lowest accelerated actions as shown in Table I. All other actions are pruned away during SAS+ preprocessing. Note that all those values are domain-specific and do not correspond to accelerations, but we will use them in an experimental manner here. The authors imposed them while executing a breakfast preparation task.

Some of our results are shown in Table II. The problem complexity simply corresponds to the number of surfaces the robot has to clean. The first instance of each problem uses no deadline, the second instance uses a deadline equal to the minimum makespan when using unaccelerated actions with $A_a = 1$ and the third instance uses a deadline close to the fastest solution possible. The last column of Table II corresponds to the average used acceleration of the returned plan. One can see that TSFD decelerates its actions to A_{min} when there is no deadline specified. On the other hand, it accelerates its actions when we use deadlines.

D. Real Robot Experiment

In another experiment, we used the real humanoid robot ARMAR-III for demonstrating planning with stressed actions in the context of a breakfast preparation task. The experiment demonstrates the integration of the TSFD planner into the robot software environment ArmarX and its capability to solve real-world problems with temporal constraints. The experiment can be seen in the video attachment of this paper, or under http://y2u.be/fspDIUZ4Ynw.

The robot first uses its episodic memory [30] to automatically recognize that a human aims to prepare scrambled eggs. In a dialogue, the robot offers its help and clarifies the temporal deadline for the preparation (10 minutes). This information triggers a temporally bounded planning task using the available actions of the robot stored in MemoryX. ArmarX creates a PDDL domain file containing the actions as well as the known acceleration bounds and acceleration functions, and a problem file containing the objects of the prior memory as well as the goal `(and (cooked eggs))`. The robot can only execute one action at once, thus the planner returns a sequential plan. Nevertheless, the robot tells the human to concurrently turn on the stove and concurrently prepare the eggs while he is executing some other action. All other action are performed by the robot.

To be able to execute the given task in only ten minutes the planner uses stressed actions. Every plan without stressed action exceeds the limit of ten minutes, hence planning is only possible if the PDDL domain file contains acceleration information. For this specific task we estimated the acceleration bounds and acceleration functions (returning risk values) based on the measured data within the OACs. Note that for the real robot, these values differ for from those shown in Table I.

E. Discussion

Our evaluation shows that plans returned by TSFD may have better quality than plans returned by TFD. Nevertheless, all returned solutions for the parking challenge are temporally simple, which means that TSFDs improved search procedure is not responsible for better results. We assume that in this special case the heuristic H_{add} proved better pruning techniques than H^{CEA}. The results of the temporally expressive domains show, that TFD fails to find shortest plans on most temporally expressive domains even with a completely explored state space. The reason is that first H^{CAE} is not able to classify temporally expressive states correctly and second TFDs search procedure does not produce temporally expressive states.

The evaluation of TSFD against the PKS planning system shows that, due to a heuristically improved search, TSFD returns plans much faster, especially on more complex tracks. Although PKS is a highly optimized planning system, it only uses a breadth-first search. Due to the optimizations of PKS TSFD has a bit worse performance on easy tracks ($0.1s$ slower). Another reason for this is that we also measured the time to translate the problem into temporal SAS+. PKS uses its own problem description language without preprocessing such as useless variable or action pruning. Hence, the usage of TSFD provides better performance on more complex tasks and additionally enables features such as temporal planning and numeric planning.

The third evaluation experiment showed that TSFD accelerates actions when there is less time and decelerates them when there is enough time. During all our test with deadlines, TSFD decides to decelerate the action `RequestFromHuman` up to some degree. That is little wonder since this action has the highest impact on the total cost ($f(a) = 5a$). But it is interesting up to which value TSFD accelerates this action. For the medium deadlines it decelerates the `RequestFromHuman` action to $a = 0.95$, $a = 0.9$ and $a = 0.825$ and accelerates the `move` action. The gradient of this deceleration shows that TSFD only accelerates actions to the minimum or maximum value if there is enough time left. During the first problem, there were simply too few other actions than `RequestFromHuman` to

decelerate it further. The more actions the optimal solution contains, the more can TSFD make use of acceleration and deceleration to reduce the cost of the solution.

The real robot experiment using ARMAR-III showed that stressed actions can be used to accelerate the task execution, if there are hard deadlines. Our robot successfully cooked scrambled eggs in 10 minutes, because it accelerates some of its actions. Moreover, this video shows that TSFD is applicable in complex, real-world scenarios within the robot software framework ArmarX.

VI. Conclusion and future work

We successfully augmented the TFD planning system with stressed actions and improved its DE planning procedure to also find solutions to temporally expressive problems. Unfortunately, the heuristic H^{CEA} is not able to classify states with required concurrency and mutually dependent actions correctly, which is why we implemented our own version of H_{add} based on the relaxed temporal planning graph. Although this heuristic has worse qualities in pruning and evaluating heuristic values, it keeps temporal expressive states. The usage of multiple heuristics at the same time assures the planner to benefit from the strengths of both heuristics. To the best of our knowledge, this is the first complete approach based on classical decision epochs. Moreover, we argued that stressed action planning is useful for temporally bounded tasks and showed, that our planning system is able to find such solutions.

During the real robot experiment, we estimated values for minimum and maximum accelerations of the robot's actions. We plan to use them as well as resulting failure rates and changed energy consumption statistics in order to define improved risk functions for real robot experiments. Second, real-world applications are usually imprecise and noisy. The usage of a probabilistic planning approach could improve resulting plans and plan execution, particularly because our used sensor model in ArmarX already supports unsharp world states. Finally, TSFD does not include planning time to makespan. For that reason, the goal-achievement-time of our planner, the sum of planning time and plan makespan, usually exceeds the defined deadline. Inspired by recent contributions, e. g. [31], we aim to integrate planning search time to makespan and to reason about expected makespan for specific branches during search space exploration.

References

[1] R. S. Lazarus and S. Folkman, "Stress, appraisal, and coping." Springer New York, 1984.

[2] T. Asfour, K. Regenstein, P. Azad, J. Schröder, N. Vahrenkamp, and R. Dillmann, "Armar-iii: An integrated humanoid platform for sensory-motor control," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2006, pp. 169–175.

[3] P. Eyerich, R. Mattmüller, and G. Röger, "Using the context-enhanced additive heuristic for temporal and numeric planning," 2009.

[4] "Icaps international planning competition," http://icaps-conference.org/index.php/Main/, accessed: 2018-06-09.

[5] W. Cushing, S. Kambhampati, and D. S. Weld, "When is temporal planning really temporal," in *In IJCAI*, 2007.

[6] C. Bäckström and B. Nebel, "Complexity results for sas+ planning," *Computational Intelligence*, vol. 11, pp. 625–655, 1993.

[7] S. Thiebaux, J. Hoffmann, and B. Nebel, "In defense of pddl axioms," vol. 168, pp. 38–69, 10 2005.

[8] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," 2003.

[9] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld, "Pddl - the planning domain definition language," 1998.

[10] J. Hoffmann, "The metric-ff planning system: Translating "ignoring delete lists" to numerical state variables," *Journal of artificial intelligence research. Special issue on the 3rd international planning competition*, vol. 20, 2003.

[11] Y. Chen, B. W. Wah, and C.-W. Hsu, "Temporal planning using subgoal partitioning and resolution in sgplan," *J. Artif. Int. Res.*, vol. 26, pp. 323–369, 2006.

[12] S. Edelkamp, "Taming numbers and durations in the model checking integrated planning system," vol. 20, 2002.

[13] A. Gerevini and I. Serina, "Lpg: A planner based on local search for planning graphs," in *In Proc. of 6th Int. Conf. on AI Planning Systems (AIPS 2002*. AAAI Press, 2002.

[14] M. B. Do and S. Kambhampati, "Sapa: A domain-independent heuristic metric temporal planner," in *In proceedings of ECP-01*, 2001, pp. 109–120.

[15] P. Haslum and H. Geffner, "Heuristic planning with time and resources," 2001.

[16] J. Kvarnström, P. Doherty, and P. Haslum, "Extending talplanner with concurrency and resources," 2000.

[17] F. Bacchus and M. Ady, "Planning with resources and concurrency a forward chaining approach," in *IJCAI International Joint Conference on Artificial Intelligence*, 01 2001, pp. 417–424.

[18] K. Halsey, D. Long, and M. Fox, "Crikey: A temporal planner looking at the integration of scheduling and planning."

[19] J. Benton, A. Coles, and A. Coles, "Temporal planning with preferences and time-dependent continuous costs," 01 2012.

[20] N. Vahrenkamp, M. Wächter, M. Kröhnert, K. Welke, and T. Asfour, "The robot software framework armarx," vol. 57, 2015.

[21] "C++ mathematical expression library," http://www.partow.net/programming/exprtk/index.html, accessed: 2018-06-09.

[22] A. Weinstein and M. Littman, "Bandit-based planning and learning in continuous-action markov decision processes," pp. 306–314, 01 2012.

[23] N. Krüger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omrčen, A. Agostini, and R. Dillmann, "Object-action complexes: Grounded abstractions of sensorimotor processes," *Robotics and Autonomous Systems*, vol. 59, pp. 740–757, 2011.

[24] M. Wächter, E. Ovchinnikova, V. Wittenbeck, P. Kaiser, S. Szedmak, W. Mustafa, D. Kraft, N. Krüger, J. Piater, and T. Asfour, "Integrating multi-purpose natural language understanding, robot's memory, and symbolic planning for task execution in humanoid robots," *Robotics and Autonomous Systems*, vol. 99, pp. 148–165, 2018.

[25] V. Wittenbeck, *Generating symbolic representations from sensorimotor experience for planning and plan execution*, Karlsruhe, 2016.

[26] R. P. A. Petrick and F. Bacchus, "Pks: Knowledge-based planning with incomplete information and sensing," 07 2008.

[27] R. Howey, D. Long, and M. Fox, "Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl."

[28] M. Maniadakis, E. E. Aksoy, T. Asfour, and P. Trahanias, "Collaboration of heterogeneous agents in time constrained tasks," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 448–453.

[29] M. Maniadakis, E. Hourdakis, and P. Trahanias, "Time-informed task planning in multi-agent collaboration," *Cognitive Systems Research*, vol. 43, pp. 291 – 300, 2017.

[30] J. Rothfuss, F. Ferreira, E. E. Aksoy, Y. Zhou, and T. Asfour, "Deep episodic memory: Encoding, recalling, and predicting episodic experiences for robot action execution," *CoRR*, vol. abs/1801.04134, 2018.

[31] M. Cashmore, A. Coles, B. Cserna, E. Karpas, D. Magazzeni, and W. Ruml, "Temporal planning while the clock ticks," in *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018.*, 2018, pp. 39–46.