Ben-Gurion University of the Negev

The Faculty of Natural Sciences

The Department of Computer Science

Online Planning in MDPs with Stochastic Durative Actions

Thesis submitted in partial fulfillment of the requirements

for the Master of Sciences degree

Tal Berman

Under the supervision of **Prof. Ronen Brafman, Dr. David Tolpin**

October 2024

Ben-Gurion University of the Negev

The Faculty of Natural Sciences

The Department of Computer Science

Online Planning in MDPs with Stochastic Durative Actions

Thesis submitted in partial fulfillment of the requirements

for the Master of Sciences degree

Tal Berman

Under the supervision of **Prof. Ronen Brafman, Dr. David Tolpin**

Signature of student: ____________	Date: ____________
Signature of supervisor: ____________	Date: ____________
Signature of chairperson of the committee for graduate studies: ____________	Date: ____________

October 2024

Online Planning in MDPs with Stochastic Durative Actions

Tal Berman

Master of Sciences Thesis

Ben-Gurion University of the Negev

2024

Abstract

Markov Decision Processes (MDPs) are a popular model for probabilistic planning. MDP actions are applied sequentially, and their effects are instantaneous. Yet, real-world scenarios often involve actions with duration and parallel action execution. We consider CoMDP+, a model that extends MDPs with durative, concurrent actions, and describe TP-MCTS, an online algorithm for solving CoMDP+ that combines Monte Carlo Tree Search (MCTS) with classical temporal planning techniques. TP-MCTS compiles durative actions to Start and End actions and enhances each tree node with a Simple Temporal Network to maintain temporal consistency and schedule the plan's action. Our empirical evaluation demonstrates the efficacy of the TP-MCTS algorithm in tackling CoMDP+.

Acknowledgements

Firstly, I would like to thank my supervisors Prof. Ronen Brafman and Dr. David Tolpin for their unwavering dedication and support. Their willingness to invest time and effort in helping me grow as a student and as a researcher. Secondly, I would like to express my gratitude to Dr. Erez Karpas for the collaboration and for sharing his insightful thoughts. I also want to thank my research lab members Elias Goldsztejn, Or Wertheim, Itamar Ben Atar, Dan Swissa, and Tal Shachar, for their camaraderie and assistance. Lastly, I want to thank my family for their love and for supporting my pursuit of a master's degree.

Contents

List of Figures

List of Tables

1

Introduction

In applications involving robots with multiple arms, cooperative multi-agent systems, and smart homes, the controlled system has multiple actuators that can perform diverse durative (i.e., non-instantaneous) actions concurrently. Often, such domains also feature temporal constraints such as deadlines (the meal should be ready by 5 PM) and time windows (solar charging is possible between 10 AM-6 PM), implying that both relative and absolute timing of actions in a plan are important for successful behavior. Classical temporal planning has developed diverse techniques and planners that deal with these issues, (e.g., [1–6]), but it assumes that action effects are deterministic. Yet, in many applications, many actions are stochastic, with various failure modes and potential side effects. Stochastic actions are often modeled using Markov decision processes (MDPs). However, MDPs assume that actions are applied sequentially, that their effects are instantaneous, and typically do not consider temporal constraints. Extensions of PO/MDPs to the multi-agent case allow for the use of (concurrent) joint-actions [7], but they are typically instantaneous and synchronized.

The combination of durative actions with stochastic effects forces us to consider the interaction between temporal constraints, success probability, and failure modes. Suppose we must select between a risky action with a short duration and a high success probability that leads to a dead-end state if it fails, and a longer, safe action with a low success probability that can be retried if it fails. An optimal policy without a deadline would typically be to reapply the safe action until it succeeds. However, if there is insufficient time to try the safe action enough times, it is better to try the risky one. Finding the optimal policy in this case requires reasoning both over temporal constraints (to understand what the deadline is), and over stochastic effects (to understand how likely the risky action is to lead to a dead-end). A more concrete example of this interaction is the *stuck car* domain, which we describe in our empirical evaluation.

Various variants of this problem of planning with stochastic, durative actions were considered by past work, but only in the *offline* setting. Most closely related are the Prottle planner [8], the gradient-based solver FPG [9], and Mausam and Weld's Concurrent MDP (CoMDP) formalism. These algorithms can insert actions only in *pivot* points – points in time in which some action's execution terminates. However, some domains with required concurrency and complex temporal constraints cannot be solved given these restrictions (e.g., see [10]). Moreover, offline algorithms are often limited in their ability to scale up, and require an explicit model.

In this paper, we consider CoMDP+, a model similar but slightly more general than CoMDPs [10], focusing on stochastic actions with deterministic durations that are identical for all outcomes of an action, although our technique is easily extendable to the case in which different outcomes have different (deterministic) durations. We propose TP-MCTS (Temporal Planning Monte Carlo Tree Search), an algorithm that combines the well-known Monte Carlo Tree Search (MCTS) algorithm [11, 12] with ideas from classical temporal planning. TP-MCTS is an *online* algorithm that can schedule actions in arbitrary, non-pivot, time points and, consequently, addresses the issues of scalability and temporal flexibility. It requires only information on action duration and the ability to sample them. To the best of our knowledge, it is the first planning algorithm with these properties.

TP-MCTS develops a search tree whose nodes contain both a state and a Simple

Temporal Network (STN) [13], which represents the various temporal constraints the plan must satisfy and come with efficient consistency checking and solution generation algorithms. To handle concurrency, the original durative actions are first transformed into instantaneous action pairs [14–16] consisting of a Start and End action. This transformation fits nicely with the STN framework, as it entails a simple temporal constraint between these two new actions. A classical MCTS algorithm is then employed to solve the transformed problem. The tree search algorithm is oblivious to the fact that the Start and End actions correspond to the same action or that there might be temporal constraints between actions. The STN component handles this part of the problem. It verifies that the plan generated by the MCTS is temporally consistent and, if not, prunes this branch. Thus, MCTS deals with action ordering, as in typical MDPs, while the STN coupled with the transformed domain model takes care of their actual timing. While MCTS estimates the value of leaf nodes using rollouts, TP-MCTS adapts the temporal relaxed planning-graph heuristic (TRPG) [17] to our stochastic setting and uses it to assign value to leaf nodes. It also employs a more sophisticated back-propagation step.

Our empirical evaluation of old and new domains demonstrates the clear advantage of our approach. All code and domains can be found at `https://github.com/taliBerman5/TP_MCTS`.

The primary contributions of this thesis are as follows:

- **Extension of the CoMDP Model:** We developed CoMDP+, an enhanced version of the CoMDP model that captures more complex and realistic domains. CoMDP+ incorporates start effects in addition to end effects, which are crucial for domains with required concurrency. Moreover, CoMDP+ supports the modeling of start, overall, and end conditions, and includes the ability to handle deadlines.

- **Introduction of TP-MCTS:** This thesis presents TP-MCTS, the first online planner designed to handle domains with stochastic, durative actions, and concurrent action execution. Unlike previous approaches that extend actions exponentially, TP-MCTS achieves this with only a linear factor of two. This improvement enables TP-MCTS to solve larger and more complex problems. Additionally, TP-MCTS is not limited to pivot point scheduling, as seen in earlier approaches, thus allowing it to address more intricate scenarios.

- **Adaptation of TRPG to the Probabilistic Case:** By adapting the Temporal Relaxed Planning Graph (TRPG) to the probabilistic case, we enable more accurate evaluation of states in domains with stochastic effects, enhancing the planner's ability to handle uncertainty in action outcomes.

The algorithm was presented in the PRL workshop, ICAPS 2024.

2

Background

2.1 Factored Markov Decision Process

In this thesis, we focus on *goal-oriented [18], factored* MDPs (fMDPs) [19] and use a variant of PPDDL [20] to specify them. Factored MDPs assume that states are assignments to variables. Goal-oriented MDPs, closely related to *stochastic-shortest path* problems, assume a set of terminal goal states. We model them as a four-tuple: $\langle \mathcal{P}, \mathcal{A}, \mathcal{G}, s_0 \rangle$ where

- $\mathcal{P}$ is a set of propositional variables that induce a state space $\mathcal{S}$ consisting of all possible truth assignments to $\mathcal{P}$.
- s_0 is the initial state.
- $\mathcal{G} \subseteq \mathcal{P}$ are the goal literals.
- $\mathcal{A}$ is a set of actions, where each action $a \in \mathcal{A}$ is a pair *(Pre, Eff)* such that:
  - *Pre* are a's precondition: a list of literals over $\mathcal{P}$.
  - *Eff* are a's effects, consisting of a set of triples (C, E, p), where C and E are a conjunction of literals representing a context (condition) and an effect, and $p \in (0, 1]$ is its probability. The set of contexts associated with an action's effects is mutually exclusive and exhaustive, and the sum of probabilities of effects with the same context is 1.

Action a =*(Pre, Eff)* is applicable in a non-terminal state s only if $s \models$*Pre.* Let $(C_1, E_1, p_1), \ldots (C_k, E_k, p_k)$ be all triples in *Eff* such that $s \models C_i$. If a is applied in state s then effect E_i will occur with probability p_i, and the resulting state s' will be identical to s on every proposition $p \in \mathcal{P}$ such that p does not appear (possibly negated) in E_i, and every other proposition will be assigned its value in E_i. This induces the Transition function (Tr) of the model. It is also possible to associate a cost with each action. States satisfying $\mathcal{G}$ are terminal and are considered goal states. Goals can be modeled as a reward function by associating a positive reward with every goal state and using a discount factor $\gamma < 1$, or by associating a strictly negative reward with actions.

2.2 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [11, 12, 21] is a class of online, sampling-based search algorithms for sequential decision problems that attempt to focus search effort on promising paths while still exploring other options. Each iteration in MCTS consists of four stages:

- Selection: The algorithm traverses from the root down the tree until it reaches a leaf node or a terminal state using the *tree policy.*
- Expansion: Once a leaf node is reached, a new child node is added to the tree.
- Simulation: From the newly expanded node, the algorithm follows a *default* policy until a terminal state is reached. This is called a *rollout.* The rewards obtained during this rollout provide an estimate of the node's value. We will replace this simulation phase with an alternative method for estimating the leaf node's value.
- Back-propagation: starting from the added node, values are propagated up the tree, updating nodes on the path from the root to the newly added node.

One commonly used *tree policy* in the Selection step uses the Upper Confidence Bounds (UCT) [12] algorithm. First, each action needs to be checked once and then according to the following formula:

$$a^* = arg\max_{a \in A} \left\{ Q(s,a) + C\sqrt{\frac{\ln[N(s)]}{N(s,a)}} \right\} \tag{2.1}$$

Where C is a parameter that controls the balance between exploration and exploitation. $N(s)$ is the number of times the state s has been visited or encountered during the search. $N(s,a)$ is the number of times action a has been taken in state s during the search.

MCTS is an anytime online planning algorithm: the number of iterations executed depends on the time allocated for decision-making, and at each point in time, a single decision is made: what should be the next action? This action is executed and the algorithm continues to select the next action.

2.3 Temporal Planning Concepts

2.3.1 Simple Temporal Networks

Simple Temporal Networks (STNs) [13] provide a convenient framework for analyzing temporal aspects in scheduling problems. Formally, an STN is a pair $S = (\mathcal{T}, \mathcal{C})$ where $\mathcal{T}$ is a set of temporal variables (events); and $\mathcal{C}$ is a finite set of binary constraints on $\mathcal{T}$, each of the form:

$$Y - X \leq \delta \tag{2.2}$$

where $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$.

A solution to a given problem instance S is referred to as a schedule. A schedule is a function $\sigma : \mathcal{T} \to \mathbb{R}$, assigning a real value to each event in $\mathcal{T}$ such that all constraints in $\mathcal{C}$ are satisfied. If such a schedule for an instance S exists then the STN is called *consistent*. Often an STN $S = (\mathcal{T}, \mathcal{C})$ is represented as a directed labeled graph $G_s = (\mathcal{T}_S, E_S, l_S)$ where the vertices $\mathcal{T}_S$ represent the events in $\mathcal{T}$ and for every constraint $Y - X \leq \delta$ in $\mathcal{C}$ there is a direct edge $(X, Y) \in E_S$ labeled by $l_S(X, Y) = \delta$.

2.3.2 Required Concurrency

Sometimes, concurrency is nice to have, as it shortens plan length (called *makespan*), but is not necessary. In others, concurrency is required. For example, a shorter makespan may be essential for meeting a deadline or a time-window. Another classic example is *match-cellar* [22]. A fuse must be replaced to fix the lights. Fuse repair requires light, too. The latter can be obtained by lighting a match. This example requires action with both start and end effects (light on when the match is lit and off when it ends) and requires the match to be lit before starting to fix the fuse, but not too long before, to cover the entire fix-fuse action duration.

2.3.3 Snap Actions

A Snap (instantaneous) action [23] is a pair *(Pre, Eff)* as described in fMDP. A durative action can be seen as a tuple $\langle a_{\vdash}, a_{\dashv}, Pre^{\leftrightarrow}, d \rangle$ where $a_{\vdash}$ and $a_{\dashv}$ are snap actions capturing the state change at the start and end of action execution, respectively. $Pre^{\leftrightarrow} \subseteq \mathcal{P}$ is the over-all condition and d represents the action duration.

2.3.4 Combining STN's with Planning

In some domains, it is possible to determine the order of actions first and then schedule them. However, domains with required concurrency necessitate combining planning with scheduling because the planning phase significantly impacts the scheduling problem, making it inseparable. Otherwise, this would result in an unscheduled plan. The CRIKEY [23] algorithm addresses this by integrating Simple Temporal Networks (STNs) during the planning phase when necessary. CRIKEY identifies situations where planning and scheduling problems interact and activates a mini-scheduler to handle relevant actions using an STN at those points.

2.3.5 Timed Initial Literals

Timed initial literals (TILs) [24] are a concept used in temporal planning to specify conditions that become true at specific times during the execution of a plan. These literals allow planners to incorporate dynamic changes in the environment that occur independently of the actions taken by the agent.

A TIL is a pair (l, t) with l a literal and $t \geq 0$ the time it becomes true. TILs can capture deadlines and time windows (e.g. workers work on shifts or agents can search for a rock only after 16PM because of the heat).

Stephen and Alexandra [25] present a method for encoding TILs using pure PDDL2.2. Their approach involves introducing dummy durative actions representing the sequence of the timed literals, as well as durative clip actions that connect these dummy actions, ensuring they are executed sequentially and without delay.

2.4 The Original CoMDPs

CoMDPs [10] extend fMDPs by associating a deterministic duration with each action and allowing for the concurrent execution of non-interacting actions. This is done by pre-processing the domain and generating new actions that are combinations of existing actions. A combination can contain any set of actions, no pair of which is mutually exclusive. Two actions are *mutex* if one of the following holds:

1. Their preconditions are inconsistent.
2. Their effects are contradictory.
3. One action's precondition contradicts a possible effect of another action.
4. One action's effect possibly modifies a proposition that influences another action's transition probabilities.

CoMDPs extend the state space to include the active actions and the remaining execution time for each active action. This compilation step can lead to an exponential blow-up in the set of actions and adds numerous real-valued variables to the state space. This greatly increases the search tree's branching factor of their search, but leads to shorter solutions.

CoMDPs restrict transitions to specific time points. In the Interwoven Epoch approach [10], transitions occur when an action within the current action combination terminates. Time is progressed to this time point, the state is updated with the action's effects, and the remaining actions' remaining-time variables are updated. This results in an MDP over an extended state-space, which they solve using the RTDP algorithm [26]. In comparison, TP-MCTS allows for flexible action scheduling using the STN, and its model increases the number of actions by a factor of two only.

Mausam and Welds' model [10] consider durative actions with a precondition that must hold when the action starts and a probabilistic effect that holds at the end. The preconditions and the effects of a parallel action combination are the union of the preconditions and effects (suitably timed) of its component actions. Their algorithm restricts action timing, and, consequently, cannot model scenarios such as the well-known *match cellar* problem [22] where you must fix a fuse using light provided by a match for which more flexible concurrent scheduling is required to solve the problem.

3

Related Work

3.1 Classical Temporal Planning

(Classical) temporal planning (TP) is a well-established research field that focuses on solving planning problems with deterministic, possibly concurrent actions having deterministic action durations or durations confined to some interval [27]. More recently, an extension to contingent domains with deterministic effects and partial, non-deterministic sensing was developed [28]. This planner selects a possible initial state and identifies the longest (in terms of actions count) deterministic plan from it. The plan contains the ordering and execution time of actions according to the temporal constraints. The planner constructs a tree by traversing the discovered plan, branching on the value of sensing actions encountered. TP-MCTS does not deal with partial observability, but models stochastic action effects rather than deterministic effects and uses a different search technique that maintains temporal information within each tree node.

3.2 Semi MDPs

Unlike classical methods, Markov decision processes (MDPs) [29] model stochastic outcomes in actions. Semi-Markov decision processes (SMDPs) extend them to model stochastic outcomes with stochastic durations that may depend on the original state and the action. However, action execution in SMDPs is sequential, while we seek to model concurrent execution and handle deadlines and other temporal constraints.

3.3 Constrained MDPs

Constrained MDPs (CMDPs) [30] and constrained SMDPs [31] extend MDPs and SMDPs with constraints on the system's behavior and the agent's actions. CMDPs seek to maximize expected cumulative reward while satisfying the constraints, typically in expectation only. The constraints are defined as cost functions $C_1, ..., C_m$ and limits $d_1, ..., d_m$. The function $C_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ where $\mathcal{S}$ are the system states, and $\mathcal{A}$ are the actions denotes the cost of the transition (s, a, s') where $s, s' \in S$, and $a \in A$. Feasible policies for a CMDP are policies that ensure the expected cumulative costs do not exceed the specified thresholds: $\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t C_i(s_t, a_t, s_{t+1})\right] \leq d_i, \quad \forall i$. Durative actions could be defined by setting transition costs to the transition durations. However, this model, too, would capture sequential execution only. Furthermore, temporal constraints have specific features that one can exploit using techniques such as STNs. TP-MCTS exploits these techiques as well as enables concurrent execution.

3.4 Porttle Planner

Few works tackle concurrent probabilistic durative domains. The Prottle planner [8] formalizes the search space as an AND/OR graph. AND nodes represent a chance associated with a probabilistic event, and OR nodes represent a choice associated with action selection. The tree is developed by either adding an action or an event concurrently with the current action or event, or moving forward in time to the next event or action. Its action description is more expressive than ours, allowing for effects at arbitrary time points and outcomes with different durations. However, it restricts the decision epochs to pivot points, which implies incompleteness in the general case [10]. In contrast, our algorithm extends beyond the confines of pivot points, providing a more versatile and comprehensive solution.

3.5 The Factored Policy-Gradient Planner

FPG Planner [9] uses a factored policy gradient approach to tackle concurrent probabilistic temporal planning problems. To handle concurrent execution, a command is defined as a decision to start zero or more actions concurrently. The planner policy needs to return a probability distribution over current legal commands. However, with N eligible actions, there are up to 2^N possible commands. This leads to exponential growth in the number of actions. To combat this command space explosion, they factored the overall policy into independent policies for each action. A gradient ascent is used for local policy search. They estimate gradients using Monte Carlo algorithms that allow arbitrary distributions. As in the Prottle planner, they restrict the decision epochs to pivot points which is not complete.

3.6 Duration Uncertainty

There are two works that use components like in our approach but they are tailored to uncertainty in the duration of the actions as opposed to probabilistic effect in our work. Foss and Onder [32] employ Simple Temporal Networks (STNs) to capture temporal constraints, as in our approach, but their uncertainty is not probabilistic and only over time. They represent uncertainty in action duration with the interval [min-d, max-d]. In their framework, they introduce two types of durative actions: (1) Assignable actions, where the user can choose a duration within [min-d, max-d].

(2) Uncertain actions, where the action duration will fall somewhere within [min-d, max-d] but is not user-controlled and becomes known only at execution time. Their algorithm adopts an optimistic approach by assuming actions will only require their minimum duration. Recognizing that this assumption might lead to an unsafe plan, they incorporate temporally contingent branches into the plan. This process involves generating an initial seed plan, identifying points where the plan is likely to fail, and inserting contingency branches at those critical points. Beaudry, Kabanza, and Michaud [33] construct a Bayesian Network, which is more general than an STN, but uncertainty is over durations, not effects. Their planning algorithm conducts a forward-chaining search in a space of states. In addition to the search graph, the planner builds a Bayesian network to track the dependency relationship between the random variables that determine event timings and action durations. Furthermore, their adaptation of the Relaxed Planning Graph (RPG) heuristic is geared to uncertainty over durations, whereas our adaptation is for uncertain effects.

3.7 Resource Uncertainty

Beaudry, Kabanza, and Michaud extended their work to include uncertainty over resources [34]. They use continuous random variables to represent resources and time rather than discrete numeric values. As in their previous work [33] they dynamically generate a Bayesian network. The Bayesian network models the dependency between time and numeric random variables. It is also queried in order to estimate the probability the resources remain in a valid state. Coles' work [35] adopts a pessimistic approach to resource consumption. During execution time, past resource consumption is known, so the pessimism can be applied only to future resource use. In their work, conditional branches are added to the plan for optional use during execution. These branches are generated by invoking the planner multiple times from states, assuming that uncertain resources will consume their expected (mean) value.

Both [34] and [35] consider extensions that deal with resource uncertainty while we focus on probabilistic effects.

3.8 Hybridized Planner

Mausam and Weld's Hybridized Planner [10], denoted MW, solves concurrent Markov decision processes (CoMDPs), addressing the setting closest to ours. CoMDPs extend MDPs by using durative actions and by allowing the execution of multiple non-interacting durative actions simultaneously. Since we use them as our baseline, we discuss them in more depth in the next section. Its main weaknesses are that it reduces CoMDPs to an MDP with potentially exponentially larger action space and much larger state space, does not support temporal constraints, and cannot model and solve problems with required concurrency and other domains that require scheduling actions to non-pivot points.

4

CoMDP+ Model and Its Solution Algorithm

4.1 The Models

Our decision model is a concurrent MDP similar to MW's model (page 19), except that we allow a richer class of durative actions as used in deterministic temporal planning: actions can have both *start* and *end* conditions and effects, as well as concurrency, or *overall* conditions – i.e., conditions that must hold throughout the action execution. This allows modeling more complex domains such as the classical *match cellar* in which the solution requires careful scheduling of concurrent actions. We also support deadlines. Since we use a more temporally expressive language, we will refer to it as CoMDP+.

Formally, a Goal-oriented CoMDP+ is a tuple $\langle \mathcal{P}, \mathcal{A}, \mathcal{G}, s_0, \mathcal{D} \rangle$ where:

- $\mathcal{P}$ is the set of propositions, defining a state space $\mathcal{S}$ consisting of all possible truth assignments to $\mathcal{P}$.
- $\mathcal{A}$ is a set of durative actions: $a = (P, E, d)$ where:
  - $P = (P_S, P_O, P_E)$ defines the conditions of a, consisting of three sets of propositions determining the applicability of action a, referred to as *start*

condition, *overall* condition, and *end* condition.

- $E = (E_S, E_E)$, where E_S and E_E are the *start* effects and *end* effects of a, respectively. E_S and E_E are defined as in factored MDPs, via sets of triples (c, e, p).
- d is the action duration.

- $\mathcal{G}$ is a set of literals denoting the goal condition.
- $\mathcal{D} \in \mathbb{R}^+$ is a deadline.
- s_0 is the initial state.

The transition function of CoMDP+ is defined as in the factored MDP (page 15). When a is applied at time t, two instantaneous changes of the system's state occur: immediately after time t (when a is applied) the state changes according to E_S. Immediately after $t + d$ (when a ends) the state changes according to E_E. The semantics of the changes are identical to that description for fMDPs.

These changes are well-defined only when a is applicable. To be applicable, the various conditions of a must hold at the appropriate time, and all concurrently executing actions must not conflict with a. We define this more formally, below.

Definition 1 *Mutex: Actions $a = (P, E, d)$ and $a' = (P', E', d')$ are mutex if E_S and P'_O are inconsistent or E'_S and P_O are inconsistent or $E_S \cup E_E$ and $E'_S \cup E'_E$ are inconsistent. That is, a has a start effect that contradicts one of the overall conditions of action a' or vice-versa. Or, some potential effect of a and some potential effect of a' at some state s are inconsistent.*[1]

[1] A weaker condition using state-dependent mutex can be defined. We enforce a strict *mutex* concept: a and a' are mutex if some effect of one is inconsistent with a condition of the other. From a decision-theoretic perspective, one could allow a and a' to occur concurrently if the probability that an inconsistency will arise is sufficiently low. This requires defining the outcomes of such executions, which is complex and unlikely to be realistic, or, in our goal-oriented setting, simply defining this to lead to a dead-end.

Definition 2 *End-Overall (EO) Mutex: Actions $a = (P, E, d)$ is considered EO mutex with action $a' = (P', E', d')$ if P_O is inconsistent with E'_E. That is, an end effect of a' violates the overall condition of a.*

Action $a = (P, E, d)$ is *applicable* at time t if the system's state is s, P_S is satisfied in s, P_O is satisfied in the interval $(t, t + d)$, P_E is satisfied at $t + d$, no action a' mutex with a is executed within the interval $[t, t + d]$, and every action a' that is EO-mutex with a and which overlaps a, ends before a. Hence, actions can be applied at any time point, including concurrently, as long as no two mutex actions are executing concurrently.

One can extend CoMDP+ with a reward function, but we focus on goal-oriented problems with the optimization criterion of maximizing goal achievement probability given the temporal constraints. CoMDP+ supports *timed initial literals* (TILs), too. A TIL is a pair (l, t) with l a literal and $t \geq 0$ the time it becomes true. TILs can capture deadlines and time windows, and they can be compiled into a CoMDP+ action executed at time 0 [25]. Our current code does not support this conversion, so we focus on deadlines only.

4.2 TP-MCTS

TP-MCTS works in two stages: Offline, use classical TP techniques to transform the CoMDP+ domain into an fMDP with instantaneous actions plus temporal constraints. Online, solve it by combining MCTS with temporal reasoning. TP-MCTS pseudo-code is shown in Algorithm 2. It is intentionally simplified for clarity by ignoring the fact that a search-depth parameter is used to constrain the depth of the search tree: Nodes deeper than this parameter are not considered.

4.2.1 Offline Preprocessing

In CoMDP+, time is continuous, but state changes are discrete events that occur at the start and end of an action. For this reason, it is possible and convenient to apply a well-known transformation of durative actions into instantaneous actions (e.g., [14–16] and others), obtaining what we refer to as the *transformed* model. A durative action a is split into two instantaneous actions a_{start} and a constraint that a_{end} is scheduled d time units after $a_{start}.a_{start}$ takes care of the start conditions and effects, a_{end} takes care of the end conditions and effects. An additional mechanism is used to ensure the overall conditions. Note that this can be extended to additional deterministic events that occur in specific time points during the execution of a, by splitting a into more instantaneous actions (not supported by our code). We can now represent the concurrent execution of actions a, a' by performing a_{start}, a'_{start}, followed by a'_{end}, a_{end}, for example.

To ensure that the overall conditions are satisfied, we add one new fluent, *InExecution(a)* for each action a, that is true while the action is executing. We make its negation a precondition of the start (respectively, end) of any action that is mutex (resp. EO-mutex) with this action. To ensure that a_{end} occurs d time units after a_{start}, we use an STN to keep track of such constraints.

We now formally define the transformation from a CoMDP+ to a regular fMDP with temporal constraints. The transformation is presented in Algorithm 1. Given a CoMDP+ $\langle \mathcal{P}, \mathcal{A}, \mathcal{G}, s_0, \mathcal{D} \rangle$, the transformed model is a fMDP $\langle \mathcal{P}', \mathcal{A}', \mathcal{G}, s_0 \rangle$ (plus some auxiliary data), such that:

- $\mathcal{P}' = \mathcal{P} \cup \{InExecution(a) | a \in \mathcal{A}\}$
- $\mathcal{A}' = \{a_{start} = (P_{a_{start}}, E_{a_{start}}) | a \in \mathcal{A}\} \cup \{a_{end} = (P_{a_{end}}, E_{a_{end}}) | a \in \mathcal{A}\}$ where, assuming $a = (P, E, d), P = (P_S, P_O, P_E)$ and $E = (E_S, E_E)$:
  - $P_{a_{start}} = P_S \cup (P_O \setminus E_S) \cup \neg InExecution(a)$

Algorithm 1 Generating the Transformed Model

```
1: procedure TRANSFORM(Problem $\langle \mathcal{P}, \mathcal{A}, \mathcal{G}, s_0, \mathcal{D} \rangle$)
2:     $\mathcal{P}' \leftarrow \mathcal{P} \cup \{InExecution(a) | a \in \mathcal{A}\}$
3:     $\mathcal{A}' \leftarrow \{\}$
4:     for $a \in \mathcal{A}$ do
5:         if $a$ is durative action then
6:             $start_a$, $end_a \leftarrow$ SPLIT($a$)
7:             $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{start_a, end_a\}$
8:         else
9:             $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{a\}$
10:        end if
11:    end for
12:    return $\langle \mathcal{P}', \mathcal{A}', \mathcal{G}, s_0 \rangle$
13: end procedure
14:
15: procedure SPLIT(Action $a$, ActionSet $\mathcal{A}$)
16:    $a = (P, E, d), P = (P_S, P_O, P_E)$ and $E = (E_S, E_E)$
17:    $P_{a_{start}} = P_S \cup (P_O \setminus E_S) \cup \neg InExecution(a)$
18:    $E_{a_{start}} = E_S \cup InExecution(a)$
19:    $P_{a_{end}} = P_E \cup InExecution(a)$
20:    $E_{a_{end}} = E_E \cup \neg InExecution(a)$
21:    $P_{a_{start}}, P_{a_{end}} \leftarrow$ CHECKMUTEX($a, \mathcal{A}, P_{a_{start}}, P_{a_{end}}$)
22:    $start_a \leftarrow (P_{a_{start}}, E_{a_{start}})$
23:    $end_a \leftarrow (P_{a_{end}}, E_{a_{end}})$
24:    return $start_a, end_a$
25: end procedure
26:
27: procedure CHECKMUTEX(Action $a$, ActionSet $\mathcal{A}$, Precondition $P_{a_{start}}$, Pre-
       condition $P_{a_{end}}$)
28:    for $a' \in \mathcal{A}$ do
29:        if Mutex($a', a$) then
30:            $P_{a_{start}} \leftarrow P_{a_{start}} \cup \neg inExecution(a')$
31:        end if
32:        if EOMutex($a', a$) then
33:            $P_{a_{end}} \leftarrow P_{a_{end}} \cup \neg inExecution(a')$
34:        end if
35:    end for
36:    return $P_{a_{start}}, P_{a_{end}}$
37: end procedure
```

- $E_{a_{start}} = E_S \cup InExecution(a)$
- $P_{a_{end}} = P_E \cup InExecution(a)$
- $E_{a_{end}} = E_E \cup \neg InExecution(a)$

Figure 4.1 illustrates the logic of splitting a durative action.

Additional preconditions and effects ensure that mutex and EO-mutex actions are not applied incorrectly:

- If a is mutex with a', add $\neg InExecution(a)$ to the precondition of a'_{start}.
- If a is EO-mutex with a', add $\neg InExecution(a)$ to the precondition of a'_{end}.

Above we assume no self-overlapping actions, i.e., two copies of the same ground action cannot overlap.[2] Notice that if the original effects are stochastic, then so are those of a_{start} and/or a_{end}, with the added effect on the $InExecution$ proposition. Because of the latter, the state at each point reflects the set of currently executing actions.

Beyond the fMDP, we also maintain the correspondence between every a_{start}, a_{end} pair corresponding to an original action a, a's duration, and the deadline D, so that the algorithm can add the relevant constraints to the STN.

4.2.2 Online Solution

Online, TP-MCTS works with the generated fMDP. We generate an initial root node containing state s_0 and an STN with *startPlan* and *endPlan* events and a constraint $endPlan - startPlan \leq D$. We now repeat the following steps: *Search* to find the next action; *Schedule* this action; *Step*: apply it; *Update* the STN. We describe them below.

The proposed algorithm makes three assumptions:

[2]If self-overlapping actions are allowed, planning complexity becomes undecidable already with deterministic action.

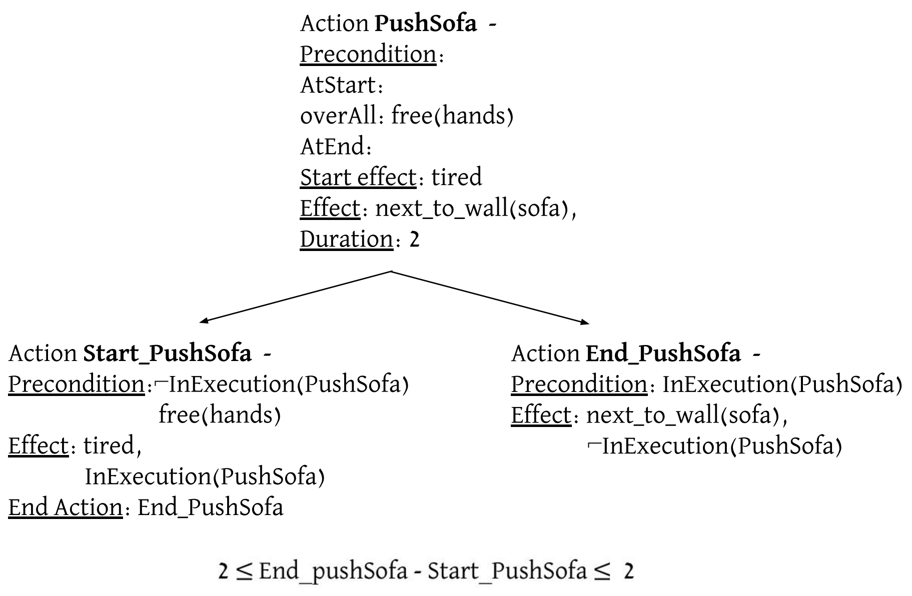
Action **PushSofa** -
Precondition:
AtStart:
overAll: free(hands)
AtEnd:
Start effect: tired
Effect: next_to_wall(sofa),
Duration: 2
Action **Start_PushSofa** -
Precondition: ¬InExecution(PushSofa) free(hands)
Effect: tired, InExecution(PushSofa)
End Action: End_PushSofa
Action **End_PushSofa** -
Precondition: InExecution(PushSofa)
Effect: next_to_wall(sofa), ¬InExecution(PushSofa)
2 ≤ End_pushSofa - Start_PushSofa ≤ 2

Figure 4.1: Spliting the durative PushSofa. To push the sofa against the wall, the robot's hands must be free. While it pushes the sofa, its hands are in use. After 2 sec. the robot finishes pushing, and the sofa is aligned with the wall.

Assumption 1 *The goal is considered achieved once all of the propositions $p \in \mathcal{G}$ (where $\mathcal{G}$ represents the set of goal propositions) are satisfied, even if it occurs in the middle of action execution.*

Assumption 2 *Once the execution of an action starts, it cannot be stopped.*

Assumption 3 *The algorithm does not take into account the passage of time during the search, i.e., the time is considered stopped during the search phase and resumes once an action is chosen.*

4.2.2.1 Search

Pseudo-code is Shown in Algorithm 3 and Algorithm 4. TP-MCTS uses the time allocated per decision online to construct a search tree with state and action nodes. State nodes contain the state reached and an STN representing the temporal constraint associated with the branch ending at this node. If the STN is consistent, a

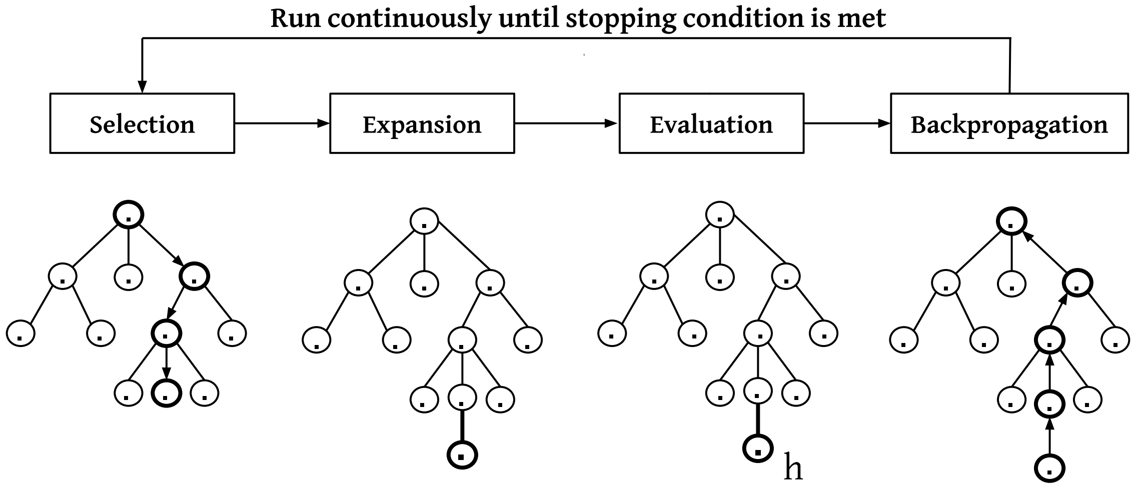

Figure 4.2: TP-MCTS. All steps except the evaluation step, are the same as in the MCTS algorithm. The dot in the nodes represents the STN that ensures the temporal constraints.

Algorithm 2 TP-MCTS

```
1: procedure TP-MCTS($CoMDP+\ \langle\mathcal{P},\mathcal{A},\mathcal{T}r,\mathcal{G},s_0\rangle$)
2:     $\langle\mathcal{P}',\mathcal{A}',\mathcal{G},s_0\rangle$ = Transform($\langle\mathcal{P},\mathcal{A},\mathcal{T}r,\mathcal{G},s_0\rangle$)
3:     $root = s_0$
4:     $STN_{root} \leftarrow initSTN()$
5:     repeat
6:         $n_{root} \leftarrow$ StateNode($root, STN_{root}$)
7:         $a, V_a \leftarrow$ Search($n_{root}$)
8:         $t \leftarrow$ Schedule($a, V_a$)
9:         $root \leftarrow$ Step($root, a$)
10:        $STN_{root} \leftarrow$ UpdateSTN($a, t, STN_{root}$)
11:    until Terminal($root$)
12: end procedure
```

legal schedule representing this branch exists. Action nodes contain an action. Both nodes maintain statistics.

Select. The tree is traversed starting from the root until reaching a new node. The UCB criterion [36] is used to select among actions (Algorithm 3, l.11). The next state is sampled based on the distribution described in its effects component (Algorithm 3, l.12). *Select*'s recursive invocation ends once we reach a new node – either by sampling a new state for an existing action or by sampling a new action and the following state. (UCB entails that a new action will be selected if we reach an existing node for which not all actions have been sampled.)

Expand. Add the new node reached to the tree. The node contains the state reached and an STN obtained by updating the parent's STN with events and constraints related to the action leading to it.

Evaluate. Typically, *evaluate* uses a simulation (rollout) step to assess the node's value. Instead, TP-MCTS uses a heuristic estimate of the probability of reaching the goal from this node in time. We use *PTRPG*, a new variant of the temporal relaxed planning graph heuristic (TRPG) [17], described later.

Back-propagation. Evaluate estimates and information about the consistent times for scheduling the root action in this path are propagated upwards. Details below.

4.2.2.2 Schedule

In standard MDPs the only decision is what action to execute next. However, in temporal domains, the probability of reaching the goal depends on when it is applied. If we could choose an action a and an execution time t for which the success probability is maximized, then we would have an asymptotically optimal algorithm for problems with deadlines. This follows because in such problems, the tree has bounded depth, and hence, success probabilities will be correct eventually (simply either 1 or 0) and because, in the limit, the frequency MCTS samples each stochastic effect of each action equals its probability.

Unfortunately, the best algorithm for checking whether a fixed policy over stochastic actions is *guaranteed* to succeed is polynomial in tree size, hence, exponential in its depth [37]. Here, we require much more: we must assess the maximal probability of success of each policy sub-tree *and* maximize over possible policies. Notice that the same action taking part in different policy sub-trees may best be scheduled at different times.

In principle, this can be done by developing the full And-Or search tree described next, and formulating the problem as a Mixed Integer Linear Program. A similar encoding could be used online for a partially developed tree with heuristic values at leaf nodes. Boolean indicator variables would be associated with tree edges, and continuous variables would be associated with each node, denoting their time. Boolean constraints would encode the structure of a legal policy, i.e., one edge indicator *true* for Or nodes, and all outcome edges indicators below a *true* action indicator are *true*. The needed temporal constraints between the time variables are all linear. The indicator variables turn on/off the associated temporal constraints using a standard technique [38]. Yet, as observed there, this method does not work well in practice. And it is unlikely to succeed in the online setting. This is not realistic computationally beyond very small trees. Hence, we resort to heuristics in which only some of the information in leaf-node STNs is propagated up the tree.

We considered two combinations of the *BackProp* and *Schedule* steps: *earliest* and *root-interval*. The latter is described in the pseudo-code. *earliest* follows UCT [12]. The estimated success probability provided by *evaluate* at a leaf node is added to each of its ancestors in the tree, and their visit count N is incremented by 1. The root action with the highest average value is chosen. It is scheduled at the earliest time the STN associated with the root, augmented with this action, is considered consistent.

earliest relies only on the root node's STN, thus, ignoring the root action's execution time on future actions. This information can be efficiently deduced from the STNs of leaf nodes, i.e., we can compute the time interval I such that, if the root action is scheduled in I, then a legal schedule exists for all actions within the branch leading to a leaf node.

In *Root-Interval*, each node n maintains a counter N and a function $V(t)$ from $t \in I_0 = [0, \text{deadline}]$ to $[0, 1]$, instead of a single value. $V_n(t)$ is an estimate of the probability the goal will be achieved if the root action leading to n is scheduled to t. Let n be a leaf node. Let c be the probability of plan success returned by *PTRPG* for n. Let I be the set of time points such that: if the root action leading to n is scheduled at $t \in I$, a legal schedule for *all* actions on the path to n exists. This is efficiently computable from n's STN. For all $t \in I$, set $V_n(t) = p$. For all $t \notin I$ set $V_n(t) = 0$. This information is propagated backward to n's ancestors. The visit count is incremented as usual. The value function can be thought of as being updated pointwise – i.e., $\forall t \in I_0$: add $V_n(t)$ to every ancestor's $V(t)$. In practice, we use an efficient algorithm that represents and updates V as a set of constant-valued intervals.

4.2.2.3 Step and Update

We apply the chosen action a at the selected time t in the real world (*step*) and generate a new root node for the next step (*update*). This node contains the state resulting from a's application and an STN that extends the current root node's STN. In the STN, each action is represented by a vertex. The constraints between different vertices influence the permissible execution intervals for each action. For each action a, let $\overline{a}$ denote the corresponding vertex in the STN.

The following constraints are then added to the new STN:

1. $startPlan - \overline{a} \leq 0$

2. $\overline{a} - endPlan \leq 0$

3. $\overline{pa(a)} - \overline{a} \leq 0$; $pa(a)$ is the action at at a's parent node.

4. If a is a start action: $d \leq \overline{end_a} - \overline{start_a} \leq d$; end_a is the end action of a.

4.2.3 The PTRPG Heuristic

TRPG [17], the relaxed planning graph heuristic for temporal domains, attempts to estimate the solution's execution time. Roughly speaking, TRPG maintains an over-estimate of the set of literals that *could* be satisfied at different time points. The literal set for node n contains all literals in s and is assigned a time t which is an under-estimate of n's execution time. All actions whose conditions are contained in the current literal set are applied at the closest time point possible – action interactions are ignored. Time progresses to the earliest applied action, and its effects are added to the literal set. The relationship between start and end actions is respected, i.e., a_{end} can be applied only if a_{start} has already been applied and a time corresponding a_{start}'s application time plus a's duration. Since no literal is ever removed from this set, the set of literals overestimates the literals achievable at each point. TRPG assumes deterministic actions. Once a deterministic action is applied, there is no need to reapply it later, since all its effects are already present. For stochastic actions, one could assume that all their possible effects hold after the action is applied, but this is too strong a relaxation. Instead, our probabilistic TRPG (PTRPG) variant samples effects according to their probability and reapplies the action once it terminates. Effect samples are independent. For example, if a has a probabilistic end-effect, its duration is 2, and it is first applicable at time 4, we will reapply it at time $6, 8$, etc., resampling its end-effects.

PTRPG returns an estimate of the time t the goal will be reached. We use two approaches to map this value to $[0, 1]$ and treat it as the probability of being able to reach the goal from the evaluated node. In both approaches if the deadline D is reached before the literal set contains all goal propositions, the problem is unsolvable, and $p = 0$. The greater $D - t$ is, the more time we have of compensating for overestimating what we can achieve by t. We use it in two ways to achieve p:

1. *Linear* Scaling Option: $p = 0.5 \cdot (1 + t/D)$ to this node. p is at least 0.5 if $t \leq D$

and increases the greater $D - t$ is. This means that if the PTRPG believes a problem is solvable, it is given at least a 50 percent chance of success.

2. *Logistic* Scaling Option: Following a common practice in Bayesian probabilistic modelling [39], we transform from the time domain to probability domain by chaining *logit* from the time range to the unconstrained space, an affine transformation on the unconstrained space, and logistic sigmoid (*expit*) from the transformed unconstrained space to [0,1]. By adjusting the parameters of the affine transformation, we can establish the relationship between the heuristic's outcome and the probability of meeting the deadline reflecting the properties of the particular heuristic. More specifically: given parameters a, b, c: Define $(1) D' = D + c$; $(2) z_1 = \log(\frac{t}{D'-t})$; $(3) z_2 = a * z_1 + b$; $(4) p = \frac{1}{1+\exp(-z2)}$. If $t > D$ we set $p = 0$. We set $a = 1, b = -0.5, c = 1$.

p can be viewed as a crude estimate of the probability that we can actually reach the goal from the current node, but essentially, it is just a heuristic value.

The PTRPG heuristic, as described in Algorithm 5, operates primarily through the *GetValue* function. This function begins by initializing the earliest time each end action can be executed using the *InitEarliest* function. For each action a currently in execution, *InitEarliest* sets $earliest(a) = a.duration + a_{\text{start}}$ *execution time*; otherwise, $earliest(a) = \infty$.

Following this initialization, *GetValue* processes the applicable actions at time t until either the goal is achieved or the estimated time t exceeds the deadline. For an action a to be considered applicable, its preconditions must be met, and if a is an end action, $earliest(a)$ must be less than or equal to t. When a start action a becomes applicable, $earliest(a)$ is updated to $\min(earliest(a), t + a.duration)$. The effects of these applicable actions are then incorporated into $s_{t+\epsilon}$.

If $s_{t+\epsilon} = s_t$, the time t advances to the next earliest time an action can be executed. Otherwise, the algorithm checks if additional applicable actions exist in the new state at time t. During each iteration, all previously applicable probabilistic actions are sampled if their $earliest(a) \leq t$. Finally, when the goal or the deadline is reached *pEvaluate* maps t to $[0, 1]$ in one of the two approaches described above.

Algorithm 3 TP-MCTS

```
1: procedure SEARCH($stateNode\ n$)
2:     while within computational budget do
3:         $n_{leaf} \leftarrow$ SELECT($n$)
4:         BACKPROP($n_{leaf}, n_{leaf}.V$)
5:     end while
6:     $a_{best} \leftarrow \arg\max_a val_{max}(na)$
7:     return $a_{best}$, $n.child(a_{best}).V$
8: end procedure
9: procedure SELECT($stateNode\ n$)
10:    $a = \arg\max_{a' \text{ a child of } n} val_{max}(na') + c\sqrt{\frac{\log N(n)}{N(na')}}$
11:    $s = \text{Step}(n.State, a)$
12:    if $n$ has a child $n'$ corresponding to $a$ and $s$ in the tree then
13:        if $n'$ is goal state then
14:            return $n'$
15:        end if
16:        return SELECT($n'$)
17:    end if
18:    return STATENODE($s, n.child(a).STN$)
19: end procedure
20: procedure EXPAND($action$ a, $STN\ S$)
21:    $STN_{new} = STN.copy().add(a)$
22:    $STN_{new} = STN_{new} \cup$constraints (1)-(4)
23:    return $node(a, STN_{new})$
24: end procedure
25: procedure EVALUATE($Snode\ snode$, $STN\ STN$)
26:    $c \leftarrow PTRPG(snode, STN)$
27:    $I \leftarrow STN.legalRoot$
28:    $V(t) \leftarrow 0$
29:    $\forall t \in I, V(t) \leftarrow c$
30:    return $V$
31: end procedure
```

Algorithm 4 TP-MCTS

```
 1: procedure StateNode($State\ s$, $STN\ STN$)
 2:     $snode \leftarrow node(s)$
 3:     for each $a$ in $s$.legalActions do
 4:         $anode \leftarrow$ Expand($a, STN$)
 5:         if $anode$.STN is consistent then
 6:             $snode$.AddChild($anode$)
 7:         end if
 8:     end for
 9:     $V \leftarrow$ Evaluate($snode, STN$)
10:     update($snode, V$)
11:     return $snode$
12: end procedure
13: procedure BackProp($stateNode\ n$, $func\ V_{leaf}$)
14:     while $n$ is not the $root$ do
15:         update($n, V_{leaf}$)
16:         $n \leftarrow n.Parent$
17:     end while
18: end procedure
19: procedure Update($Node\ n$, $func\ V$)
20:     $\forall t \in [0, deadline]$, $n.V(t) \leftarrow n.V(t) + V(t)$
21:     $n.N \leftarrow n.N + 1$
22:     $n.val_{max} \leftarrow \frac{\max_t n.V(t)}{n.N}$
23: end procedure
24: procedure Schedule($action\ a$, $func\ V$)
25:     $T_{\max} \leftarrow \arg\max_t V(t)$
26:     $t \leftarrow$ The earliest time the action can be executed in $T_{max}$
27:     return $t$
28: end procedure
```

Algorithm 5 PTRPG Heuristic

```
 1: procedure getValue($State\ s$, $Time\ CT$, $Func\ A_{time}$)
 2:     $s_0 \leftarrow s$
 3:     $t \leftarrow CT$
 4:     initEarliest($A_{time}$)
 5:     while $t < deadline$ and $Goal$ is not achieved do
 6:         $s_{t+\epsilon} \leftarrow s_t$
 7:         $a_t \leftarrow \{A_\dashv \mid pre(A_\dashv) \subseteq s_t \wedge earliest(A) \leq t\}$
 8:         for each new $A_\dashv \in a_t$ do
 9:             $s_{t+\epsilon} \leftarrow s_{t+\epsilon} \cup eff^+(A_\dashv) \cup eff^-(A_\dashv)$
10:         end for
11:         $a_t \leftarrow a_t \cup \{A_\vdash \mid pre(A_\vdash) \subseteq s_t\}$
12:         for each new $A_\vdash \in a_t$ do
13:             $s_{t+\epsilon} \leftarrow s_{t+\epsilon} \cup eff^+(A_\vdash) \cup eff^-(A_\vdash)$
14:             $earliest(A) = min[earliest(A), t + d(A)]$
15:         end for
16:         drawProbabilistic($a_t, s_{t+\epsilon}, t$)
17:         if $s_t \subset s_{t+\epsilon}$ then $t \leftarrow t + \epsilon$
18:         else
19:             $endpoints \leftarrow \{earliest(A) \mid pre(A_\dashv) \subseteq s_t \wedge earliest(A) > t\}$
20:             if $endpoints = \emptyset$ then $t \leftarrow \infty$
21:             else $t \leftarrow min[endpoints]$
22:             end if
23:         end if
24:     end while
25:     return pEvaluate($t, deadline$)
26: end procedure
27: procedure initEarliest($Func\ A_{time}$)
28:     for each $A_\dashv$ do
29:         if $inExecution(A) \in s_0$ then
30:             $earliest(A) \leftarrow A_{time}(A) + A.duration$
31:         else $earliest(A) \leftarrow \infty$
32:         end if
33:     end for
34: end procedure
35: procedure drawProbabilistic($a, s, t$)
36:     for each $probabilistic$ $A_\dashv \in a$ do
37:         if $earliest(A) <= t$ then
38:             $s \leftarrow s \cup sample(E_{A_\dashv})$
39:             $earliest(A) = t + d(A)$
40:         end if
41:     end for
42:     for each $probabilistic$ $A_\vdash \in a$ do
43:         $s \leftarrow s \cup sample(E_{A_\dashv})$
44:     end for
45: end procedure
```

5 Experimental Results

We compare the two TP-MCTS variants (earliest and root-interval) with MW, the closest relevant algorithm [10]. MW-RTDP is the original MW variant that uses RTDP [26], adapted to the online setting. For a fairer comparison, we also tested MW-MCTS which uses MCTS, which is often better suited for online planning. Like TP-MCTS, MW-MCTS uses PTRPG to estimate node values instead of a rollout. Assumptions 1 and 3 are maintained in all versions.

Our comparison covers more domains than previous work. We considered eight domains: five structured domains that model real-world problems: NASA Rover, Machine Shop adapted from MW's work, a new Stuck-Car domain that exhibits more interesting stochastic effects, and two new domains with required concurrency: Hosting and a stochastic variant of *match-cellar*. The other three domains are synthetic domains that we use to examine basic properties of the algorithms. We conducted experiments with two possible decision-time budgets: 1,10 seconds. In order to enable performance comparison between MW-RTDP, MW-MCTS, and TP-MCTS, most of the domains can be solved in MW's domain representation.

All algorithms were implemented in Python. The experiments were run on a computer with AMD EPYC 7702P 64-Core Processor. Each experiment was repeated 100 times. The results are the success rate and the average makespan of the realized trajectory over each successful run and its standard deviation. The TP-MCTS algorithm implementation supports domain representations written in the unified planning framework, a project of AIPlan4EU [40], making it easier for users to utilize and define new domains.

5.1 Domains

Stuck Car(C) C agents must get C cars out of the mud they are stuck in before a deadline is reached. Agents can push a car and/or its gas pedal, possibly simultaneously. Pushing the car has a higher probability of success than pushing the gas. Executing both actions simultaneously has a higher probability of success than performing each action independently, but takes longer, and the agent may become tired. The agents can also search for a rock and place it beneath the car to aid in its release. The rock's quality influences the probability of success, and it may be better to drop a rock of poor quality. Each car can be pushed by a single agent. Another agent could concurrently e.g., search for a rock for this car, or push a different car. To handle this constraint in the MW approach a *ready* predicate is added to the domain and an instantaneous *turnOn* action.

Hosting-v We must clean our house and prepare food before guests arrive. It is possible to clean and cook at the same time. Although cooking may result in a dirty floor, requiring cleaning completion after cooking concludes. There are two versions of the Hosting domain. In the more complex version the broom is missing and the agent needs to find it. There is a probability of the broom being found and the light needs to be turned on while searching. In this domain MW's domain representation is not sufficient to capture the setting, they do not model start effects.

Nasa Rover(R) A probabilistic variant of the well-known *NASA Rover* domain from the 2002 AIPS Planning Competition [41] with R different rovers. 'R' different rovers were sent to Mars to sample rocks, take photos, and communicate the findings back to Earth. Each Rover is equipped with a camera and is sent to Mars with instructions to sample specific rocks. To achieve these goals, the rovers need to perform various preparation actions, such as calibrating the camera, turning on the hand, etc. Some rover actions' execution may fail. Each rover has two arms, good and bad hands which can execute actions simultaneously. However, these hands take different times to execute the same action and have different failure probabilities. The good hand always succeeds in its actions, but takes longer, while the bad hand is instantaneous, but may fail.

Machine Shop(O) This is MW's probabilistic version of the classic job-shop planning domain. The domain captures a manufacturing environment comprising various subtasks, including shaping, painting, polishing, and more. Each subtask needs to be performed on different pieces using specific machines. Machines can perform in parallel, but not all are capable of every task. Pieces may need to be relocated to the appropriate machine capable of executing the required subtask. Execution of a subtask can end in failure. The goal of the domain is to successfully complete all subtasks and release all the machines. The 'O' parameter indicates the amount of different machines and pieces in the domain. Each machine can work on only one piece at a specific time. The machine can work simultaneously on different pieces.

Prob Match Cellar(O) In a probabilistic variant of the *match-cellar* domain [22], the problem involves fixing O broken fuses. Lighting is necessary to repair a fuse, and there exists a probability that the repair attempt may fail. There are O matches available, each capable of producing light while burning. Multiple matches can be ignited concurrently to facilitate simultaneous fuse repairs. However, each match can only be used to repair one fuse at a time.

Simple-x In this simple domain, there are x distinct actions with duration 4, each achieving a unique goal. There are no conflicts between the actions and the most efficient solution is to execute all actions simultaneously. As x grows, MW's representation grows exponentially. but the solution depth remains 1. With the transformed model, the action space and the solution depth grow linearly with x, as does the number of propositions (due to $InExecution$ propositions).

Conc This domain was designed to challenge the planners' abilities to handle problems requiring maximal concurrency to meet a deadline. There are four actions that take 1 time unit, two that take 2 time units, one that takes 4, and one that takes 9 time units. To meet the deadline of 9 seconds, we must always execute four actions concurrently, one from each class, except between times 4 and 5. Each of the four 1-unit actions requires the effects of the preceding one, similarly for the 2-unit actions. Then, an action that requires the effects of the last 1,2,4 actions can be applied. It adds a new effect and deletes all previous effects. The same 1,2,4 actions must now re-establish these effects to achieve the goal. The desired action execution sequence is illustrated in figure 5.1. The optimal solution execution time is nine seconds

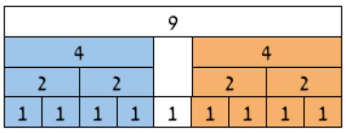

Figure 5.1: Conc domain

Prob Conc+G A probabilistic variant of Conc. with four actions: A deterministic action with duration 8 and three probabilistic actions with durations 4,2,1. All must succeed to achieve the goal. Failed execution does not change the state. Longer actions have a higher success probability. Denote the success probability for the i-unit action, $p_i, i \in 1, 2, 4$. With a deadline of eight seconds, an i-second action can be executed $8/i$ times, and the probability of failing to achieve the desired effect of each probabilistic action is $(1 - p_i)^{8/i}$.

To evaluate the planners' ability to handle a growing amount of actions and their ability to distinguish between relevant and irrelevant actions within a state effectively, we add G irrelevant actions to the action set. Each garbage action achieves a proposition that holds no relevance to the goal. Importantly, these garbage actions are not mutually exclusive (mutex) with other actions in the domain.

5.2 Results

First, we note that TP-MCTS offline compilation time is often less than 0.01 seconds, with the largest Machine Shop problem requiring 0.35 seconds, and hence negligible. MW's compilation time reaches 74 and 28 minutes for Rovers(2) and Machine Shop(3) respectively, and times out on larger versions of these problems.

Tables 5.1 and 5.2 present the experimental results for a decision-time budget of one and ten seconds, respectively for the PTRPG *logistic* heuristic. The PTRPG *linear* heuristic results are presented in four tables. Tables 5.3 and 5.4 display the experimental results for domains that MW can model. In contrast, Tables 5.5 and 5.6 show the experimental results for domains that MW cannot model, both for decision-time budgets of one and ten seconds, respectively. In the *linear* approach three deadlines are selected for each domain: a small deadline that allows reaching the goal without any action failures, a medium deadline that permits a minor degree of action failure, and a large deadline. The average makespan is computed *only* over successful runs. Hence, the primary statistic is the success rate.

A number of general results emerge:

1. MW MCTS almost always dominates MW RTDP. We introduced MW MCTS for a fairer comparison in the online setting. RTDP is better only in the smaller Simple domain, where, due to action non-interaction, it can update the value function quickly.

2. TP-MCTS dominates in almost all domains. There are two exceptions in each PTRPG evaluation formula approach: In *linear* Stuck Car(1), MW benefits from its shorter solution depth. This advantage no longer holds in the larger Stuck Car(2). In *logistic* we can see this in Nasa Rover(1) where this advantage disappears in the larger Nasa Rover(2). MW MCTS has a higher success rate in Prob-Conc+9 in *linear* and in *logistic* in Prob-Conc+8. Given the performance

of other Prob-Conc variants, this may be due to the variance in the success estimates. Overall, the average solution makespans are similar, except for the Simple-x domains. The impact of domain size on TP-MCTS (and the other algorithms) is clearly visible in the three structured domains.

3. When the deadline is extended, the success rate improves, but the average makespan also increases. Conversely, when deadlines are tightened further, the success rate decreases.

4. As expected, additional search time almost always leads to increased success rates for all algorithms. Occasionally, the average makespan improves, too, but only slightly. Yet, the general trends (1)-(3) hold in both 1 sec. and 10 sec. per search step tables.

5. Hosting-2 and Prob Match Cellar(i) are not solvable by MW's method due to the need for required concurrency.

6. Changing the PTRPG evaluation formula can have a significant impact on success rates. The *logistic* evaluation formula performs better on larger NASA Rover and Machine Shop domains. However, the *linear* formula performs better on the Conc domain in conjunction with the root version of TP-MCTS. Nevertheless, as above, the relative strength of the four variants remains similar. In particular, TP-MCTS remains much stronger than the MW variants. For future work, learning the parameters in the *logistic* evaluation approach may lead to better results.

The results in *Simple-x* highlight the difficulty MW's algorithm has scaling up as the number of non-mutex actions increases due to the exponential growth in the number of legal action combinations, which implies a very large branching factor. Although the solution depth is smaller, the algorithm does not have sufficient time to explore all actions and has a low chance of detecting the solution. Given more time, it is able to scale up slightly. TP-MCTS requires deeper solutions, but here, MCTS combined with PTRPG is apparently able to focus its exploration on more promising paths. This difficult is observed in other domains such as *Nasa Rover(2)*, *Machine Shop(2)*, and *Machine Shop(3)*.

Comparing the *earliest* and *root-interval* versions of TP-MCTS, we see that given less time, *earliest* can perform better because it can expand more nodes, provided the temporal constraints are not complex. For most problem instances, trying to start an action as soon as possible is a good heuristic. However, with sufficient time, *root-interval* succeeds more often, and in domains with more complex temporal dependencies, such as Hosting, it is the only version that can solve the problem reliably.

Table 5.1: *Logistic* Exp. Results: 1 sec. per search step. "**-**": Did not compile within 8h. "**N/A**": Cannot model the domain. **Bold**: Best.$|\mathcal{A}|, |\mathcal{P}|$ – # of actions and propositions.

		Success Rate (%)				Average Makespan (std)				Compilation Time	
Problem	$(\lvert\mathcal{A}\rvert, \lvert\mathcal{P}\rvert)$	TP-MCTS		MW RTDP	MW MCTS	TP-MCTS		MW RTDP	MW MCTS	TP-MCTS	MW
		earliest	root			earliest	root				
Stuck Car(1)	(7,9)	68	**91**	33	83	9.63 (0.55)	9.7 (0.37)	8.06 (0.73)	10.56 (0.38)	0.003s	0.004s
Stuck Car(2)	(24,18)	60	**81**	14	24	13.76 (0.51)	14.52 (0.45)	15.07 (0.96)	14.46 (0.85)	0.002s	0.039s
Hosting-1	(2,3)	0	**100**	0	0	-	10 (0)	-	-	0.002s	0.0014s
Nasa Rover(1)	(33,27)	91	67	79	**92**	22.8 (0.48)	29.27 (0.52)	24.85 (0.63)	24.96 (0.61)	0.037s	0.04s
Nasa Rover(2)	(66,80)	**83**	79	0	0	25.87 (0.48)	28.82 (0.46)	-	-	0.105s	74.65m
Nasa Rover(3)	(99,159)	**71**	70	-	-	27.51 (0.46)	29.34 (0.46)	-	-	0.194s	-
Machine Shop(2)	(30,26)	**96**	75	21	52	18.88 (0.22)	22.11 (0.34)	23.38 (0.87)	21.77 (0.53)	0.0224s	0.5s
Machine Shop(3)	(75,45)	**84**	50	0	0	20.78 (0.31)	24.22 (0.32)	-	-	0.0991s	28.9m
Machine Shop(4)	(148,68)	40	**48**	-	-	20.97 (0.46)	23.39 (0.32)	-	-	0.355s	-
Simple-10	(10,10)	**100**	**100**	**100**	31	4 (0)	4 (0)	4.04 (0.04)	10.58 (0.35)	0.003s	0.014s
Simple-11	(11,11)	**100**	**100**	59	16	4.44 (0.13)	4 (0)	11.39 (0.19)	11.5 (0.5)	0.003s	0.042s
Simple-12	(12,12)	100	**100**	0	29	5 (0.17)	**4 (0)**	-	10.21 (0.37)	0.003s	0.148s
Simple-13	(13,13)	100	**100**	0	30	5.08 (0.18)	**4 (0)**	-	10.53 (0.36)	0.003s	0.586s
Simple-15	(15,15)	100	**100**	0	23	5.28 (0.19)	**4 (0)**	-	11.48 (0.29)	0.004s	8.6s
Conc	(9,9)	**100**	0	0	0	11.25 (0.08)	-	-	-	0.003s	0.005s
Prob Conc+7	(11,5)	**94**	**94**	72	64	8.82 (0.15)	9.35 (0.18)	12.07 (0.22)	11.09 (0.28)	0.003s	0.046s
Prob Conc+8	(12,5)	94	**95**	37	76	8.68 (0.14)	9.46 (0.19)	12.81 (0.26)	11.05 (0.24)	0.003s	0.16s
Prob Conc+9	(13,5)	**96**	86	24	88	9.31 (0.18)	9.62 (0.21)	13.17 (0.3)	10.5 (0.25)	0.003s	0.61s
Prob Conc+10	(14,5)	**93**	92	37	92	9.58 (0.19)	9.67 (0.22)	13.38 (0.22)	9.69 (0.22)	0.003s	2.28s
Hosting-2	(4,4)	0	**87**	N/A	N/A	-	10 (0)	N/A	N/A	0.0015s	N/A
Prob Match Cellar(1)	(2,4)	**95**	90	N/A	N/A	2.42 (0.08)	2.42 (0.09)	N/A	N/A	0.0017s	N/A
Prob Match Cellar(2)	(6,8)	**85**	**85**	N/A	N/A	3.36 (0.15)	3.65 (0.13)	N/A	N/A	0.0032s	N/A
Prob Match Cellar(3)	(12,12)	**75**	72	N/A	N/A	3.8 (0.16)	3.5 (0.17)	N/A	N/A	0.0069s	N/A
Prob Match Cellar(4)	(20,16)	**70**	56	N/A	N/A	3.74 (0.11)	4.52 (0.24)	N/A	N/A	0.0123s	N/A
Prob Match Cellar(5)	(30,20)	**46**	46	N/A	N/A	**3.91 (0.06)**	6.76 (0.13)	N/A	N/A	0.0227s	N/A

Table 5.2: *Logistic* Exp. Results for 10 sec. per step. Names abbreviated.

	Success Rate (%)				Average Makespan (std)			
Problem	TP		MW R.	MW M.	TP		MW R.	MW M.
	earliest	root			earliest	root		
S. Car(1)	91	**94**	31	91	8.62 (0.37)	9.5 (0.37)	9.64 (0.6)	8.95 (0.33)
S. Car(2)	**69**	62	13	37	13.43 (0.57)	16.84 (0.31)	14.92 (1.22)	14.27 (0.84)
Hosting-1	0	**100**	0	0	-	10 (0)	-	-
Rover(1)	**91**	90	65	**91**	23.17 (0.5)	28.14 (0.43)	24 (0.68)	23.13 (0.6)
Rover(2)	**86**	50	0	0	27.45 (0.47)	30.24 (0.42)	-	-
Rover(3)	**78**	65	-	-	27.45 (0.44)	30.29 (0.45)	-	-
Shop(2)	**98**	88	64	82	18.46 (0.25)	22.49 (0.27)	20.67 (0.43)	21.12 (0.35)
Shop(3)	**74**	42	0	0	22 (0.34)	24.57 (0.29)	-	-
Shop(4)	**58**	23	-	-	22.84 (0.37)	25 (0.43)	-	-
Sim-10	**100**	**100**	**100**	22	4 (0)	4 (0)	4 (0)	10.54 (0.42)
Sim-11	**100**	**100**	**100**	27	4.24 (0.09)	4 (0)	4 (0)	10.81 (0.36)
Sim-12	100	**100**	**100**	7	8 (0)	**4 (0)**	**4 (0)**	9.14 (1.14)
Sim-13	**100**	**100**	**100**	11	5.04 (0.18)	4 (0)	4.24 (0.09)	11.27 (0.49)
Sim-15	**100**	**100**	0	18	4.84 (0.16)	4 (0)	-	10.44 (0.47)
Conc	**100**	100	87	12	**12 (0)**	15 (0)	13.67 (0.09)	10.17 (0.17)
P.Co+7	90	**95**	84	48	9.02 (0.18)	9.31 (0.16)	8.7 (0.16)	10.96 (0.34)
P.Co+8	91	91	**94**	53	8.57 (0.14)	9.87 (0.21)	9.03 (0.16)	11.92 (0.3)
P.Co+9	92	**96**	91	55	8.97 (0.17)	9.57 (0.19)	10.48 (0.16)	11.38 (0.29)
P.Co+10	**97**	91	68	62	9.06 (0.17)	9.12 (0.18)	11.95 (0.23)	11.92 (0.25)
Hosting-2	0	**93**	N/A	N/A	-	10 (0)	N/A	N/A
P.MC(1)	**93**	89	N/A	N/A	2.34 (0.08)	2.52 (0.09)	N/A	N/A
P.MC(2)	**83**	81	N/A	N/A	2.75 (0.11)	3.62 (0.1)	N/A	N/A
P.MC(3)	**80**	56	N/A	N/A	3.82 (0.17)	3.84 (0.16)	N/A	N/A
P.MC(4)	**79**	47	N/A	N/A	3.89 (0.15)	3.98 (0.1)	N/A	N/A
P.MC(5)	**69**	61	N/A	N/A	4.11 (0.14)	5.61 (0.19)	N/A	N/A

Table 5.3: *Linear* Exp. Results: 1 sec. per search step. ”**-**”: Did not compile within 8h. **Bold**: Best.$|\mathcal{A}|, |\mathcal{P}|$ – # of actions and propositions.

		Deadline	Success Rate (%)				Average Makespan (std)				Compilation Time	
Problem	$(\|\mathcal{A}\|, \|\mathcal{P}\|)$		TP-MCTS		MW RTDP	MW MCTS	TP-MCTS		MW RTDP	MW MCTS	TP-MCTS	MW
			earliest	root			earliest	root				
Stuck Car(1)	(7,9)	10	48	**85**	24	44	7.04 (0.4)	8.34 (0.17)	4.75 (0.5)	7.59 (0.36)	0.003s	0.004s
		15	87	92	33	**95**	10.1 (0.38)	9.78 (0.36)	8.9 (0.82)	10 (0.29)	”	”
		30	94	89	56	**99**	12.5 (0.74)	15.2 (0.94)	13.17 (1.1)	12.7 (0.65)	”	”
Stuck Car(2)	(24,18)	10	30	**54**	5	8	8 (0.32)	8.22 (0.23)	5.8 (1.32)	7 (0.073)	0.002s	0.039s
		20	**78**	72	14	31	13.8 (0.46)	13.02 (0.5)	13.1 (1.01)	11.7 (0.96)	”	”
		30	88	**91**	25	45	17.76 (0.67)	17.77 (0.6)	17.44 (1.3)	19.93 (1.1)	”	”
Hosting-1	(2,3)	10	0	**100**	0	0	-	10 (0)	-	-	0.002s	0.0014s
		15	0	**100**	100	100	-	**10 (0)**	15 (0)	15 (0)	”	”
		30	0	**100**	100	100	-	**10 (0)**	15 (0)	15 (0)	”	”
Nasa Rover(1)	(33,27)	25	**71**	61	48	65	19.4 (0.45)	22.9 (0.26)	20.66 (0.48)	20.35 (0.4)	0.037s	0.04s
		35	**97**	74	77	88	21.9 (0.59)	28.5 (0.59)	23.3 (0.57)	23.9 (0.6)	”	”
		50	**99**	86	94	**99**	26.88 (0.67)	37 (0.866)	27.22 (0.8)	27.2 (0.77)	”	”
Nasa Rover(2)	(66,80)	25	**47**	33	0	0	21.25 (0.4)	23.2 (0.32)	-	-	0.105s	74.65m
		35	**77**	76	0	0	27.2 (0.62)	27.3 (0.54)	-	-	”	”
		50	**95**	94	0	0	27.4 (0.73)	32.6 (0.72)	-	-	”	”
Nasa Rover(3)	(99,159)	25	**28**	14	-	-	22.6 (0.41)	23.9 (0.39)	-	-	0.194s	-
		35	**59**	59	-	-	**28.9 (0.5)**	29.7 (0.44)	-	-	”	”
		50	90	**95**	-	-	29.84 (0.6)	34.2 (0.72)	-	-	”	”
Machine Shop(2)	(30,26)	17	**60**	13	6	10	17 (0)	17 (0)	17 (0)	17 (0)	0.0224s	0.5s
		27	**94**	74	21	52	19.2 (0.33)	22.8 (0.29)	21.7 (0.85)	21.5 (0.55)	”	”
		40	**99**	98	38	85	21.23 (0.45)	27.6 (0.54)	29.15 (1.1)	26.4 (0.75)	”	”
Machine Shop(3)	(75,45)	17	**21**	2	0	0	17(0)	17(0)	-	-	0.0991s	28.9m
		27	**79**	38	0	0	21.8 (0.39)	23.8 (0.39)	-	-	”	”
		40	**97**	81	0	0	24.65 (0.56)	31.7 (0.64)	-	-	”	”
Machine Shop(4)	(148,68)	17	**5**	1	-	-	17(0)	17(0)	-	-	0.355s	-
		27	**18**	18	-	-	**23.3 (0.65)**	25.7 (0.37)	-	-	”	”
		40	64	**73**	-	-	27.41 (0.95)	29.41 (0.7)			”	”
Simple-10	(10,10)	10	**100**	**100**	84	12	4 (0)	4 (0)	8 (0)	8 (0)	0.003s	0.014s
		15	**100**	**100**	**100**	45	**4 (0)**	**4 (0)**	**4.04 (0.04)**	14.1 (0.39)	”	”
		30	**100**	**100**	**100**	84	**4 (0)**	**4 (0)**	**4.08 (0.08)**	18.33 (0.6)	”	”
Simple-11	(11,11)	10	**100**	**100**	0	12	4.36 (0.11)	4 (0)	-	7.66 (0.33)	0.003s	0.042s
		15	**100**	**100**	51	39	4 (0)	4 (0)	13.8 (0.34)	13.6 (0.48)	”	”
		30	100	**100**	100	83	8 (0)	**4 (0)**	14.76 (0.3)	18.75 (0.7)	”	”
Simple-12	(12,12)	10	**100**	**100**	0	9	4.76 (0.15)	4 (0)	-	7.55 (0.44)	0.003s	0.148s
		15	**100**	**100**	0	29	4 (0)	4 (0)	-	10 (0.43)	”	”
		30	100	**100**	98	90	5.84 (0.2)	**4 (0)**	22.5 (0.34)	18.4 (0.58)	”	”
Simple-13	(13,13)	10	100	**100**	0	7	5 (0.17)	**4 (0)**	-	8 (0)	0.003s	0.586s
		15	**100**	**100**	0	22	4 (0)	4 (0)	-	11.3 (0.34)	”	”
		30	100	**100**	82	92	6.36 (0.19)	**4 (0)**	26.1 (0.27)	17.13 (0.6)	”	”
Simple-15	(15,15)	10	100	**100**	0	4	5.36 (0.19)	**4 (0)**	-	8 (0)	0.004s	8.6s
		15	100	**100**	0	28	5.4 (0.19)	**4 (0)**	-	11.4 (0.27)	”	”
		30	100	**100**	2	98	5.52 (0.19)	**4 (0)**	28 (0)	18.2 (0.49)	”	”
Conc	(9,9)	10	**100**	0	0	0	10 (0)	-	-	-	0.003s	0.005s
		15	**100**	100	0	0	**10.7 (0.05)**	15 (0)	-	-	”	”
		30	**100**	100	24	0	**11 (0)**	28.57 (0.2)	21.4 (0.96)	-	”	”
Prob Conc+7	(11,5)	10	**87**	72	32	23	8.15 (0.05)	8.53 (0.08)	9.91 (0.05)	9.22 (0.16)	0.003s	0.046s
		15	**96**	88	82	93	9.3 (0.9)	9.9 (0.22)	11.7 (0.19)	10 (0.24)	”	”
		30	**100**	99	100	96	**9.66 (0.6)**	13.27 (0.4)	13.05 (0.36)	15.26 (0.6)	”	”
Prob Conc+8	(12,5)	10	**84**	73	21	36	8.24 (0.06)	8.74 (0.09)	10 (0)	9.17 (0.12)	0.003s	0.16s
		15	**93**	90	47	92	9.25 (0.19)	9.5 (0.2)	13.6 (0.28)	10 (0.21)	”	”
		30	**100**	100	96	98	**10.6 (0.35)**	12.7 (0.45)	16.1 (0.44)	13.4 (0.46)	”	”
Prob Conc+9	(13,5)	10	**80**	**80**	0	47	8.16 (0.06)	8.6 (0.08)	-	8.72 (0.12)	0.003s	0.61s
		15	89	91	32	**95**	9.5 (0.22)	9.8 (0.2)	13.5 (0.28)	9.6 (0.22)	”	”
		30	**100**	**100**	96	**100**	10.04 (0.3)	12.5 (0.47)	16.9 (0.46)	11.49 (0.3)	”	”
Prob Conc+10	(14,5)	10	**80**	**80**	0	59	8.22 (0.06)	8.59 (0.07)	-	8.61 (0.11)	0.003s	2.28s
		15	**90**	87	28	90	**9.3 (0.22)**	9.8 (0.19)	13.1 (0.35)	9.5 (0.22)	”	”
		30	**100**	99	95	**100**	11.4 (0.37)	12.3 (0.46)	20.24 (0.4)	10.9 (0.29)	”	”

Table 5.4: *Linear* Exp. Results for 10 sec. per step. Names abbreviated.

	deadline	Success Rate (%)				Average Makespan (std)			
Problem		TP		MW R.	MW M.	TP		MW R.	MW M.
		earliest	root			earliest	root		
S. Car(1)	10	64	**79**	26	59	6.48 (0.3)	7.8 (0.23)	5.73 (0.45)	7.95 (0.24)
	15	**98**	92	40	96	9 (0.29)	9.65 (0.36)	7.9 (0.73)	8.3 (0.26)
	30	**100**	91	53	**99**	10.42 (0.46)	14.9 (0.97)	14.23 (1.12)	11.6 (0.52)
S. Car(2)	10	29	**46**	1	7	7.55 (0.4)	7.24 (0.34)	8 (0)	8.43 (0.65)
	20	76	**77**	18	26	11.9 (0.43)	14.1 (0.52)	12.5 (1.23)	13.5 (0.66)
	30	86	72	24	55	14.81 (0.65)	24.64 (0.53)	20.46 (1.16)	19.64 (0.84)
Hosting-1	10	0	**100**	0	0	-	10 (0)	-	-
	15	0	**100**	100	100	-	**10 (0)**	15 (0)	15 (0)
	30	0	**100**	100	100	-	**10 (0)**	15 (0)	15 (0)
Rover(1)	25	**65**	43	33	61	20.2 (0.45)	22.67 (0.37)	20.36 (0.56)	19.65 (0.44)
	35	**98**	78	75	93	21.5 (0.53)	28.02 (0.5)	23.4 (0.69)	21.1 (0.48)
	50	**100**	96	90	**100**	25.7 (0.67)	37.96 (0.49)	28.4 (0.76)	24.64 (0.73)
Rover(2)	25	**46**	27	1	0	21.43 (0.41)	23.26 (0.32)	25 (0)	-
	35	**84**	57	0	0	24.7 (0.45)	31.59 (0.4)	-	-
	50	**97**	72	0	1	31.56 (0.66)	44.74 (0.54)	-	42 (0)
Rover(3)	25	**22**	**22**	-	-	23.09 (0.47)	24 (0.26)	-	-
	35	75	**76**	-	-	27.5 (0.49)	30.8 (0.43)	-	-
	50	**93**	84	-	-	31.13 (0.75)	38.53 (0.71)		
Shop(2)	17	**57**	**57**	27	54	17 (0)	17 (0)	17 (0)	17 (0)
	27	**97**	85	88	95	19.3 (0.34)	22.3 (0.29)	21.5 (0.37)	21.2 (0.28)
	40	**100**	99	90	95	19.76 (0.4)	26.8 (0.5)	26.03 (0.6)	25.46 (0.67)
Shop(3)	17	**16**	4	0	0	17 (0)	17 (0)	-	-
	27	**92**	71	0	0	21.2 (0.28)	24.4 (0.27)	-	-
	40	**91**	78	0	0	26.4 (0.6)	34.78 (0.35)	-	-
Shop(4)	17	1	**2**	-	-	17 (0)	17 (0)		
	27	**56**	12	-	-	21.5 (0.38)	24.08 (0.7)	-	-
	40	**79**	34	-	-	27.64 (0.61)	34.7 (0.69)	-	-
Sim-10	10	**100**	**100**	**100**	13	4 (0)	4 (0)	4 (0)	8 (0)
	15	**100**	**100**	**100**	56	4 (0)	4 (0)	4 (0)	13.4 (0.45)
	30	**100**	**100**	**100**	91	4 (0)	4 (0)	4 (0)	18.02 (0.6)
Sim-11	10	**100**	**100**	**100**	3	4.56 (0.14)	4 (0)	4 (0)	8 (0)
	15	**100**	**100**	**100**	36	4 (0)	4 (0)	4 (0)	12.8 (0.52)
	30	**100**	**100**	**100**	82	4.4 (0.12)	4 (0)	4 (0)	19.75 (0.65)
Sim-12	10	100	**100**	**100**	5	7.96 (0.04)	**4 (0)**	**4 (0)**	8 (0)
	15	**100**	**100**	**100**	10	4 (0)	4 (0)	4 (0)	10.4 (0.65)
	30	100	**100**	**100**	71	8 (0)	**4 (0)**	**4 (0)**	20.34 (0.64)
Sim-13	10	**100**	**100**	80	3	4.16 (0.08)	4 (0)	8 (0)	8 (0)
	15	**100**	**100**	60	26	4 (0)	4 (0)	13.9 (0.26)	10.2 (0.4)
	30	100	**100**	100	66	6 (0.2)	**4 (0)**	6.32 (0.21)	19.82 (0.78)
Sim-15	10	**100**	**100**	0	1	4.68 (0.15)	4 (0)	-	8 (0)
	15	**100**	**100**	0	18	4 (0)	4 (0)	-	10.9 (0.43)
	30	100	**100**	93	81	5.48 (0.2)	**4 (0)**	23.65 (0.33)	18.56 (0.59)
Conc	10	**100**	0	1	0	10 (0)	-	10 (0)	-
	15	**100**	100	89	0	**11 (0)**	15 (0)	13.8 (0.09)	-
	30	**100**	100	100	4	**12 (0)**	27.15 (0.01)	15.02 (0.22)	21.5 (1.26)
P.Co+7	10	80	**83**	75	11	8.24 (0.07)	8.48 (0.08)	8.17 (0.05)	9 (0.23)
	15	92	**94**	85	88	9.2 (0.18)	9.8 (0.22)	8.7 (0.18)	10 (0.21)
	30	**100**	99	96	82	10.11 (0.32)	14.34 (0.48)	10.26 (0.41)	18.08 (0.71)
P.Co+8	10	80	**83**	75	11	8.24 (0.07)	8.48 (0.08)	8.17 (0.05)	9 (0.23)
	15	95	94	**98**	91	9.2 (0.19)	8.9 (0.16)	9.2 (0.19)	10.1 (0.24)
	30	**100**	98	**100**	88	10.72 (0.42)	12.96 (0.48)	9.46 (0.33)	15.9 (0.62)
P.Co+9	10	**89**	74	60	17	8.18 (0.05)	8.31 (0.08)	9.58 (0.06)	8.94 (0.18)
	15	91	95	**96**	87	9 (0.17)	9.57 (0.2)	10.4 (0.17)	10.2 (0.22)
	30	**100**	100	**100**	86	**9.78 (0.28)**	12 (0.42)	**10.14 (0.23)**	15.27 (0.52)
P.Co+10	10	**85**	74	31	17	8.26 (0.07)	8.35 (0.07)	9.84 (0.07)	9.12 (0.22)
	15	**92**	90	66	**92**	9.3 (0.2)	9.5 (0.2)	11.9 (0.22)	9.5 (0.17)
	30	**100**	99	**100**	91	9.89 (0.27)	12.25 (0.47)	11.45 (0.3)	15.01 (0.56)

Table 5.5: *Linear* Exp. Results: 1 sec. per search step. Domains MW unable to model

Problem	$(\vert\mathcal{A}\vert, \vert\mathcal{P}\vert)$	deadline	Success Rate (%)		Average Makespan (std)		Compilation Time
			earliest	root	earliest	root	
Hosting-2	(4,4)	10	7	**100**	11.5 (0.57)	10.1 (0.06)	0.0015s
		15	8	**100**	11.5 (0.33)	10.21 (0.08)	”
		30	16	**100**	15.87 (0.89)	11.15 (0.06)	”
Prob Match Cellar(1)	(2,4)	3	**71**	67	2 (0)	2 (0)	”
		5	**94**	87	2.47 (0.09)	2.46 (0.09)	0.0017s
		10	87	**94**	2.34 (0.08)	2.47 (0.9)	”
Prob Match Cellar(2)	(6,8)	3	**51**	45	2 (0)	2 (0)	”
		5	79	**84**	2.78 (0.11)	3.56 (0.1)	0.0032s
		10	82	**90**	3.06 (0.15)	3.55 (0.14)	”
Prob Match Cellar(3)	(12,12)	3	36	**37**	2 (0)	2 (0)	”
		5	68	**72**	3.15 (0.12)	3.29 (0.13)	0.0069s
		10	71	**79**	3.32 (0.19)	3.57 (0.15)	”
Prob Match Cellar(4)	(20,16)	3	18	26	2 (0)	2 (0)	”
		5	65	**75**	3.51 (0.11)	3.33 (0.14)	0.0123s
		10	**65**	58	3.53 (0.13)	4.34 (0.24)	”
Prob Match Cellar(5)	(30,20)	3	9	**11**	2 (0)	2 (0)	”
		5	57	**65**	3.61 (0.11)	3.57 (0.12)	0.0227s
		10	53	**54**	3.85 (0.07)	6.11 (0.22)	”

Table 5.6: *Linear* Exp. Results: 10 sec. per search step. Domains MW unable to model

Problem	deadline	Success Rate (%)		Average Makespan (std)	
		earliest	root	earliest	root
Hosting-2	10	13	**100**	12.07 (0.3)	10.1 (0.04)
	15	11	**99**	11.72 (0.41)	10.21 (0.08)
	30	72	**100**	17.72 (0.17)	18.45 (0.15)
Prob Match Cellar(1)	3	71	**73**	2 (0)	2 (0)
	5	**90**	89	2.47 (0.09)	2.43 (0.09)
	10	93	**95**	2.52 (0.09)	2.52 (0.09)
Prob Match Cellar(2)	3	**49**	47	2 (0)	2 (0)
	5	79	**83**	2.66 (0.11)	3.7 (0.11)
	10	**82**	81	2.7 (0.1)	3.86 (0.11)
Prob Match Cellar(3)	3	**40**	39	2 (0)	2 (0)
	5	70	**75**	3.2 (0.12)	3.12 (0.14)
	10	**79**	62	3.31 (0.11)	3.53 (0.15)
Prob Match Cellar(4)	3	24	**28**	2 (0)	2 (0)
	5	60	**72**	3.3 (0.12)	3.33 (0.13)
	10	**71**	67	3.56 (0.12)	3.37 (0.13)
Prob Match Cellar(5)	3	14	**21**	2 (0)	2 (0)
	5	59	**71**	3.49 (0.11)	3.58 (0.11)
	10	**67**	58	3.7 (0.14)	3.31 (0.14)

6

Discussion and Conclusions

We presented the TP-MCTS algorithm for planning in MDPs with durative, concurrent actions. TP-MCTS combines the idea of Monte-Carlo tree search with techniques for temporal planning: Start and End actions, STNs, and the TRPG heuristic. Using these ideas, we are able to search as if actions are instantaneous, employing classical MDP search techniques and focusing on their order only, while the temporal aspects are handled by the STN. The probabilistic elements are handled by sampling, so an explicit model is not required. We also introduced a more elaborate backpropagation step that backs up values related to the temporal constraints over the next action. This results in a richer yet more economical representation and a more scalable planning algorithm.

Bibliography

[1] Long, Derek and Fox, Maria. Exploiting a graphplan framework in temporal planning. In *(ICAPS 2003)*, pages 52–61. AAAI, 2003. URL `http://www.aaai.org/Library/ICAPS/2003/icaps03-006.php`.

[2] Schoenauer, Marc, Savéant, Pierre, and Vidal, Vincent. Divide-and-evolve: A new memetic scheme for domain-independent temporal planning. In *Evolutionary Computation in Combinatorial Optimization, 6th European Conference, EvoCOP 2006, Budapest, Hungary, April 10-12, 2006, Proceedings*, volume 3906 of *Lecture Notes in Computer Science*, pages 247–260. Springer, 2006. URL `https://doi.org/10.1007/11730095_21`.

[3] Vidal, Vincent and Geffner, Hector. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artif. Intell.*, 170(3): 298–335, 2006. URL `https://doi.org/10.1016/j.artint.2005.08.004`.

[4] Coles, Amanda Jane, Coles, Andrew, Fox, Maria, and Long, Derek. Forward-chaining partial-order planning. In *ICAPS 2010*, pages 42–49. AAAI, 2010. URL `http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/view/1421`.

[5] Bit-Monnot, Arthur, Ghallab, Malik, Ingrand, Félix, and Smith, David E. FAPE: a constraint-based planner for generative and hierarchical temporal planning. *CoRR*, abs/2010.13121, 2020. URL `https://arxiv.org/abs/2010.13121`.

[6] Panjkovic, Stefan and Micheli, Andrea. Expressive optimal temporal planning via optimization modulo theory. In *AAAI 2023*, pages 12095–12102. AAAI Press, 2023. URL `https://doi.org/10.1609/aaai.v37i10.26426`.

[7] Bernstein, Daniel S., Givan, Robert, Immerman, Neil, and Zilberstein, Shlomo. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, 27(4):819–840, 2002.

[8] Little, Iain, Aberdeen, Douglas, Thiébaux, Sylvie, et al. Prottle: A probabilistic temporal planner. 2005.

[9] Buffet, Olivier and Aberdeen, Douglas. The factored policy-gradient planner. *Artificial Intelligence*, 173(5-6):722–747, 2009.

[10] Mausam and Weld, Daniel S. Planning with durative actions in stochastic domains. *Journal of Artificial Intelligence Research*, 31:33–82, 2008.

[11] Coulom, Rémi. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[12] Kocsis, Levente and Szepesvári, Csaba. Bandit based monte-carlo planning. In *ECML*, pages 282–293, 2006.

[13] Dechter, Rina, Meiri, Itay, and Pearl, Judea. Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95, 1991.

[14] Coles, A. J., Coles, A. I., Fox, M., and Long, D. Forward-chaining partial-order planning. In *ICAPS*, 2010.

[15] Benton, J., Coles, Amanda Jane, and Coles, Andrew. Temporal planning with preferences and time-dependent continuous costs. In McCluskey, Lee, Williams, Brian Charles, Silva, José Reinaldo, and Bonet, Blai, editors, *ICAPS*. AAAI, 2012.

[16] Coles, Andrew, Fox, Maria, Long, Derek, and Smith, Amanda. Temporal planning with required concurrency using classical planning. volume 25, pages 129–137, 2015.

[17] Coles, Andrew, Fox, Maria, Long, Derek, and Smith, Amanda. Planning with problems requiring temporal coordination. In *AAAI*, volume 2, pages 892–897, 2008.

[18] Puterman, Martin L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[19] Boutilier, Craig, Dearden, Richard, and Goldszmidt, Moisés. Stochastic dynamic programming with factored representations. *Artificial intelligence*, 121 (1-2):49–107, 2000.

[20] Younes, Håkan LS and Littman, Michael L. Ppddl1. 0: The language for the probabilistic part of ipc-4. In *Proc. International Planning Competition*, 2004.

[21] Keller, Thomas and Helmert, Malte. Trial-based heuristic tree search for finite horizon mdps. In *ICAPS*, 2013.

[22] Coles, Andrew, Fox, Maria, Halsey, Keith, Long, Derek, and Smith, Amanda. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, 173(1):1–44, 2009.

[23] Halsey, Keith, Long, Derek, and Fox, Maria. Crikey-a temporal planner looking at the integration of scheduling and planning. In *Workshop on Integrating Planning into Scheduling, ICAPS*, pages 46–52. Citeseer, 2004.

[24] Edelkamp, Stefan and Hoffmann, Jörg. Pddl2. 2: The language for the classical part of the 4th international planning competition. Technical report, Technical Report 195, University of Freiburg, 2004.

[25] Cresswell, Stephen and Coddington, Alexandra. Planning with timed literals and deadlines. In *Proceedings of 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, pages 23–35, 2003.

[26] Barto, A.G., Bradtke, S.J., and Singh, S.P. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 672(1-2):81–138, 1995.

[27] Fox, Maria and Long, Derek. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124, 2003.

[28] Carreno, Yaniel, Petillot, Yvan, and Petrick, Ronald. Temporal planning with incomplete knowledge and perceptual information. *arXiv preprint arXiv:2207.09709*, 2022.

[29] Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, 2005.

[30] Altman, Eitan. *Constrained Markov decision processes*, volume 7. CRC press, 1999.

[31] Beutler, Frederick J. and Ross, Keith W. Time-average optimal constrained semi-markov decision processes. *Advances in Applied Probability*, 18(2):341–359, 1986.

[32] Foss, Janae N and Onder, Nilufer. Generating temporally contingent plans. *Planning and Learning in A Priori Unknown or Dynamic Domains*, page 62, 2005.

[33] Beaudry, Eric, Kabanza, Froduald, and Michaud, Francois. Planning for concurrent action executions under action duration uncertainty using dynamically generated bayesian networks. In *ICAPS*, volume 20, pages 10–17, 2010.

[34] Beaudry, Eric, Kabanza, Froduald, and Michaud, François. Planning with

concurrency under resources and time uncertainty. In *ECAI*, volume 2010, page 217, 2010.

[35] Coles, Amanda Jane. Opportunistic branched plans to maximise utility in the presence of resource uncertainty. In *ECAI*, volume 2012, page 252, 2012.

[36] Auer, Peter, Cesa-Bianchi, Nicolò, and Fischer, Paul. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002. doi: 10.1023/A:1013689704352. URL `https://doi.org/10.1023/A:1013689704352`.

[37] Hunsberger, Luke and Posenato, Roberto. A new approach to checking the dynamic consistency of conditional simple temporal networks. In Rueher, Michel, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 268–286. Springer, 2016. doi: 10.1007/978-3-319-44953-1_18. URL `https://doi.org/10.1007/978-3-319-44953-1_18`.

[38] On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65(3):545–566, 2016. doi: 10.1007/s10589-016-9847-8. URL `https://doi.org/10.1007/s10589-016-9847-8`.

[39] Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., and Rubin, D.B. *Bayesian Data Analysis.* Chapman & Hall/CRC Texts in Statistical Science. CRC Press, 2013.

[40] Kapellos, K, Micheli, A, and Valentini, A. Aiplan4eu: Planning and scheduling for space applications.

[41] Long, Derek and Fox, Maria. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

תוכן עניינים

תכנון אונליין של MDPs עם פעולות מתמשכות בעלות אפקטים הסתברותיים

טל ברמן

עבודת גמר לתואר מוסמך למדעי הטבע

אוניברסיטת בן־גוריון בנגב

2024

תקציר

תהליך החלטה מרקובי (MDP) הוא מודל פופולרי לתכנון הסתברותי. פעולות MDP מיושמות אחת לאחר השנייה, והשפעותיהן הן מיידיות. עם זאת, תרחישים בעולם האמיתי כוללים לעתים קרובות פעולות עם השפעות לא מיידיות וביצוע פעולות במקביל. בעבודה זאת, אנו מגדירים את מודל ה־CoMDP+, מודל המרחיב את ה־MDP עם פעולות מתמשכות, אפשרות להפעלת פעולות במקביל, ומתארים את TP־MCTS, אלגוריתם אונליין לפתרון CoMDP+ המשלב חיפוש עצי מונטה קרלו (MCTS) עם טכניקות תכנון זמנים קלאסיות. TP־MCTS יוצר מפעולות עם אפקטים לא מיידים, פעולות התחלה וסיום בעלות אפקטים מיידים. בנוסף, מכיל בכל צומת בעץ Network Simple Temporal כדי לשמור על עקביות זמנים ולתזמן את פעולות התוכנית. ההערכה האמפירית שלנו מדגימה את היעילות של אלגוריתם TP־MCTS בהתמודדות עם CoMDP+.

אוניברסיטת בן־גוריון בנגב
הפקולטה למדעי הטבע
המחלקה למדעי המחשב

תכנון אונליין של MDPs עם פעולות מתמשכות בעלות אפקטים הסתברותיים

חיבור זה מהווה חלק מהדרישות לקבלת התואר מוסמך למדעי הטבע (M.Sc)

טל ברמן

בהנחיית פרופ' רונן ברפמן, ד"ר דוד טולפין

אוקטובר 2024