Online Planning in MDPs with Stochastic Durative Actions

Tal Berman[1], Ronen I. Brafman[1], Erez Karpas[2]

[1]Ben Gurion University of the Negev, Israel
[2]Technion — Israel Institute of Technology
bermant@post.bgu.ac.il, brafman@bgu.ac.il, karpase@technion.ac.il

Abstract

Markov Decision Processes (MDPs) are a popular model for probabilistic planning. Actions in MDPs are applied sequentially, and their effects are instantaneous. Yet, real-world scenarios often involve actions with duration and parallel action execution. This paper considers CoMDPs, a model that extends MDPs with durative, concurrent actions, and describes TP-MCTS, an online algorithm for solving CoMDPs that combines Monte Carlo Tree Search (MCTS) with techniques used in classical temporal planning. TP-MCTS uses a compilation of 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 CoMDPs.

Introduction

In many applications, the controlled system has multiple actuators that can perform diverse durative (i.e., non-instantaneous) actions concurrently. Examples include humanoid robots and other robots with multiple actuators, cooperative multi-agent systems, and smart homes. Moreover, such domains often feature temporal constraints such as deadlines (the meal should be ready by 5PM) and time windows (solar charging is possible between 10AM-6PM), 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, including (Long and Fox 2003b; Schoenauer, Savéant, and Vidal 2006; Vidal and Geffner 2006; Coles et al. 2010a; Bit-Monnot et al. 2020; Panjkovic and Micheli 2023). However, classical temporal planning 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 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 concurrency and consider joint-actions (Bernstein et al. 2002). However, joint-actions, too, are typically instantaneous and synchronized.

In this paper, we seek to model and solve domains that have durative actions with stochastic effects that can be applied concurrently in the presence of deadlines. Various variants of this problem were considered by diverse authors but only in the *offline* setting. Of these, the most closely related works include the Prottle planner (Little, Aberdeen, and Thiébaux 2005), Buffet and Aberdeen's gradient-based solver (Buffet and Aberdeen 2009), and Mausam and Weld's Concurrent MDP (CoMDP) formalism. In these models, actions can only be inserted 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 (Mausam and Weld 2008)).

In this paper, we consider a model similar to CoMDPs (Mausam and Weld 2008). We focus 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 seek an *online* algorithm that can schedule actions in arbitrary, non-pivot, time points, and propose the TP-MCTS (Temporal Planning Monte Carlo Tree Search) algorithm, which combines the well-known Monte Carlo Tree Search (MCTS) algorithm (Coulom 2006) with ideas from classical temporal planning. Specifically, the nodes of the tree developed by TP-MCTS contain both a state and a Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991), which represents the various temporal constraints the plan must satisfy and come with efficient consistency checking and solution generation algorithms. To the best of our knowledge, this is the first online algorithm to tackle this domain setting, enabling us to solve more complex problems.

To handle concurrency, the original durative actions are first transformed into instantaneous action pairs (Coles et al. 2010b; Benton, Coles, and Coles 2012; Jiménez, Jonsson, and Palacios 2015) 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 and ensures that the plan generated by the MCTS remains 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) (Coles et al. 2008) to our stochastic setting and uses it to assign value to leaf nodes.

We conducted an extensive empirical evaluation of our algorithm, comparing it with an online version of Mausam's concurrent MDP algorithm (Mausam and Weld 2008) on the two domains used by Mausam on a novel Stuck-Car domain and three additional synthetic domains, showing the clear advantage of our approach. Code and domains can be found at https://github.com/taliBerman5/TP_MCTS.

Related Work

(Classical) temporal planning is a well-established research field that focuses on solving classical planning problems involving durative actions and concurrent execution (Fox and Long 2003). Work in this area focuses on deterministic actions with action durations that are deterministic or confined to some interval duration.

More recently, (Carreno, Petillot, and Petrick 2022) introduced TraCE, a temporal planner for contingent domains with concurrent actions and partial observability. TraCE assumes state-changing actions are deterministic and only sensing actions are non-deterministic. It selects a possible initial state and identifies the longest (in terms of actions count) deterministic plan from it. This 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.

Unlike classical methods, Markov decision processes (MDPs) (Puterman 2005) model stochastic actions. Semi-Markov decision processes (SMDPs) extend them to model stochastic actions with stochastic durations. Action duration is modeled as a continuous random variable. This variable's distribution may depend on the original state and the action. However, action execution in SMPDs is sequential, while we seek to model concurrent execution and handle deadlines and other temporal constraints.

Constrained MDPs(CMDPs) (Altman 1999) and SMDPs (Beutler and Ross 1986) extend MDPs and SMDPs with constraints on the system's behavior and the agent's actions. The objective in CMDPs is maximizing the expected cumulative reward while satisfying the constraints. The constraints are defined as a function $g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ where $\mathcal{S}$ are the system states, and $\mathcal{A}$ are the actions. The function g denotes the cost of the transition (s, a, s') where $s, s' \in S$, and $a \in A$. 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. Our work exploits such temporal planning techniques to handle these constraints, and models concurrent execution.

Few works tackle concurrent probabilistic durative domains. Little, Aberdeen, and Thiebaux introduced The Prottle planner (Little, Aberdeen, and Thiébaux 2005), which formalizes the search space as an AND/OR graph. The AND nodes represent a chance associated with the probabilistic events, and the OR nodes represent a choice associated with action selection. There are two options to use each node, as a selection or advancement. The selection choice selects actions and the advancement chance advance time to the next event. Their action description is more expressive than ours – they allow for effects at arbitrary time points and outcomes with different durations – their formulation restricts the decision epochs to pivot points. As discussed in (Mausam and Weld 2008), this restriction implies incompleteness in the general case. In contrast, our algorithm extends beyond the confines of pivot points, providing a more versatile and comprehensive solution. Buffet and Aberdeen (Buffet and Aberdeen 2009) introduced a factored policy gradient approach that also allows actions only in pivot points. Foss and Onder (Foss and Onder 2005) employ STN to capture temporal constraints, as in our approach but their uncertainty is not probabilistic and only over time. Beaudry, Kabanza and Michaud (Beaudry, Kabanza, and Michaud 2010a) construct a Bayesian Network, which is more general than an STN but uncertainty is over durations, not effects. Furthermore, their adaptation of the RPG heuristic is geared to this property, whereas our adaptation is for uncertain effects. Beaudry, Kabanza, and Michaud extended their work (Beaudry, Kabanza, and Michaud 2010b) to encompass resource uncertainty, a focus shared with Colas's work (Coles 2012), contrasting with our work, which centers on probabilistic effects.

The approach that addresses a problem setting most similar to ours is Mausam and Weld's Hybridized Planner (Mausam and Weld 2008) for solving concurrent Markov decision processes (CoMDPs). CoMDPs extend MDPs by modeling actions as durative 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. Their main weakness is that they reduce CoMDPs to an MDP with potentially exponentially larger action space and much larger state space, do not support temporal constraints, and cannot model and solve problems with required concurrency and other domains that require scheduling actions to non-pivot points.

Background

Markov Decision Process

In this paper, we focus on *goal-oriented, factored* MDPs. 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 three-tuple: $\langle \mathcal{P}, \mathcal{A}, \mathcal{G} \rangle$ where

- $\mathcal{P}$ is a set of propositional variables that induce a state space $\mathcal{S}$ consisting of all possible truth assignments to $\mathcal{P}$.

- $\mathcal{G}$ is a set of literals over $\mathcal{P}$.
- $\mathcal{A}$ 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 is 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. It is also possible to associate a cost with each action. States satisfying $\mathcal{G}$ are terminal and are considered goal states. There are various ways this can be modeled as a reward function. For example, one can associate a positive reward with every goal state and use a discount factor $\gamma < 1$, or one can associate a strictly negative reward with actions.

Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) (Coulom 2006; Kocsis and Szepesvári 2006; Keller and Helmert 2013) is a class of sampling-based search algorithms for sequential decision problems that attempt to balance exploration and exploitation. 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.

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.

Simple Temporal Networks

Simple Temporal Networks (STNs) (Dechter, Meiri, and Pearl 1991) 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:

$$Y - X \leq \delta \tag{1}$$

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*.

The Original CoMDPs

CoMDPs (Mausam and Weld 2008) extend MDPs to allow for concurrent execution of non-interacting actions by preprocessing 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 considered *mutex* if one of the following holds: (1) Their preconditions are inconsistent. (2) Their effects are contradictory. (3) One action's precondition conflicts with the possible effects 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 of the active actions. This compilation step is quite costly. It 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 branching factor of their search tree, but also leads to shorter solutions.

CoMDPs restrict transitions to specific time points. Mausam and Weld (MW) consider two schemes: In the Interwoven Epoch approach, transitions occur when an action within the current actions 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. In the Aligned Epoch approach, a new action can be applied only when all actions in the current combination terminate. In both cases, the result is an MDP over an extended state-space, which they solve using the RTDP algorithm (Barto, Bradtke, and Singh 1995). In comparison, TP-MCTS allows for flexible action scheduling handled by the STN, and the model it uses increases the number of actions by a factor of two, only.

MW's model considers only durative actions with a precondition that must hold when the action starts and an effect that holds at the end. The preconditions and the effects of a parallel action combination is simply the union of the preconditions and effects (suitably timed) of its component actions. This, combined with the restrictions on action timing, implies that their formalism cannot model scenarios such as the well-known *match cellar* problem (Coles et al. 2009) where you must fix a fuse using light provided by a match for which more flexible concurrent scheduling is required to solve the problem.

The Models

Our decision model is a concurrent MDP similar to MW's model, 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. There, the action *light-match* must overlap the action *fix-fuse* which has an overall condition *light*, which is a start effect of *light-match* and is negated by the end effect of *light-match*. Hence, *light-match* must start before *fix-fuse* and end after it ends. We also support deadlines and time windows through the use of *timed-initial-literals (TILs)* (Edelkamp and Hoffmann 2004). We will (re)use the term CoMDP to refer to this model class.

Formally, a Goal-oriented CoMDP is a tuple $\langle \mathcal{P}, \mathcal{A}, \mathcal{T}r, \mathcal{G}, s_0, \mathit{TILs} \rangle$ 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: $a = (P, E, d_I)$ 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_I is the (controllable) duration interval(s). The decision maker can select any duration within d_I. To simplify notation, we will assume henceforth that $d_I = [d, d]$ is a point interval.
- $\mathcal{G}$ is a set of literals denoting the goal condition.
- $\mathit{TILs} = \{(l_i, t_i) | i \in I\}$ are called *timed initial literals*. l_i is some literal over the propositions in $\mathcal{P}$ and $t_i \geq 0$ denotes a time in which this literal becomes true. TILs allow us to model deadlines and time windows.
- s_0 is the initial state.

The application of a at time t causes two instantaneous changes of the system's state: at time t, when a is applied, the state changes according to E_S. At time $t + d$, the state changes according to E_E. The semantics of the changes are identical to our description for factored MDPs.

These changes are well-defined only when a is applicable. To be applicable, the various preconditions of a must hold at the appropriate time, and all concurrently executing actions must not conflict with a. Below, we define a strict *mutex* concept: a and a' are mutex if some effect of one is inconsistent with a condition of the other.

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 preconditions 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]Weaker condition using state-dependent mutex can be defined.

Definition 2. *Soft Mutex: Actions $a = (P, E, d)$ is considered soft 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.*

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. However, as long as the impact of this event is not clear (e.g., it could be catastrophic), it is difficult to weigh it properly. The latter requires defining outcomes for joint actions given all their possible relative timings. This is highly complex and not likely to be realistic.

We say that $a = (P, E, d)$ is *applicable* at time t if the system's state is s, P_S is satisfied in s, P_O is satisfied in every time point t' such that $t < t' < t + d$, P_E is satisfied at $t + d$, no action a' that is mutex with a is executed within the interval $[t, t + d]$, and every action $a' = (P', E', d')$ that is soft-mutex with a that is executed within an interval $[t, t + d']$, satisfies $t + d' < t + d$ (i.e., a' ends before a ends).

Our ultimate optimization criterion is maximizing the sum of (possibly discounted) rewards. However, in our experiments, we consider goal achievement under deadlines, and so our focus will be on maximizing expected goal achievement while satisfying deadlines and other temporal constraints.

The Transformed Model

In the CoMDP model time is continuous but state changes are discrete events occurring at the start and end of an action. For this reason, it is possible and convenient to compile durative actions into instantaneous actions (e.g., (Coles et al. 2010b; Benton, Coles, and Coles 2012; Jiménez, Jonsson, and Palacios 2015) 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_{end}, where a_{start} captures the start preconditions and effects and a_{end} captures the end preconditions and effects. With this compilation, we can represent concurrent execution of actions a, a' by performing $a_{start}, a'_{start}, a'_{end}, a_{end}$, for example.

However, this compilation does not address two issues: the overall conditions of an action, and the duration of the action. To ensure that the overall conditions are satisfied, we add one new fluent for each action that is true while the action is executing. We make its negation a precondition of the start (respectively, end) action that is mutex (resp. soft-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 MDP with temporal constraints by specifying the new MDP and the temporal constraints associated with each state.

Given a CoMDP $\langle \mathcal{P}, \mathcal{A}, \mathcal{T}r, \mathcal{G}, s_0 \rangle$ (we are ignoring the TILs, whose treatment is straightforward), the transformed model is a factored MDP $M = \langle \mathcal{P}', \mathcal{A}', \mathcal{G}, s_0 \rangle$ with a set of constraints $\mathcal{C}$, 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 $a =$

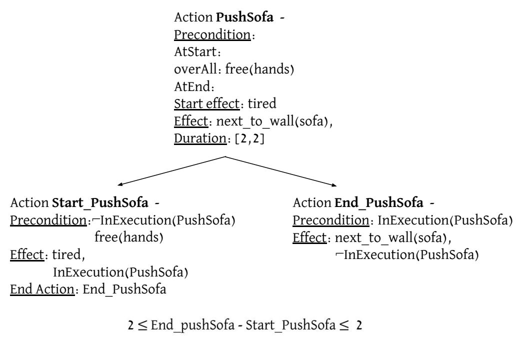
Action PushSofa -
Precondition:
AtStart:
overAll: free(hands)
AtEnd:
Start effect: tired
Effect: next_to_wall(sofa),
Duration: [2,2]
Action Start_PushSofa -
Precondition: ¬InExecution(PushSofa) free(hands)
Effect: tired, InExecution(PushSofa)
End Action: End_PushSofa
Action End_PushSofa -
Precondition: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

Figure 1: Spliting the durative action 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 two seconds the robot finishes pushing, and the sofa is aligned with the wall.

$(P, E, d), P = (P_S, P_O, P_E)$ and $E = (E_S, E_E)$:

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

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. Figure 1 illustrates the logic of splitting a durative action.

Additional preconditions and effects ensure that mutex and soft-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 soft-mutex with a', add $\neg InExecution(a)$ to the precondition of a'_{end}.

In addition, the correspondence between every pair a_{start} and a_{end} corresponding to an original action a must be maintained so that the algorithm can add the relevant constraint to the STN.

The TP-MCTS Algorithm

It is possible to solve a CoMDP offline optimally 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. 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 will be added between actions. The indicator variables turn on/off the associated constraints using a standard technique (Belotti et al. 2016). Yet, as observed there, this method does not work well in practice. Hence, we present below a heuristic approach that comes with no guarantees, yet, as our empirical evaluation shows, it is better than previous methods.

The TP-MCTS algorithm solves problems modeled as a CoMDP. It applies MCTS to the transformed domain constructing a search tree in which each tree node contains the state reached and an STN representing the temporal constraint associated with the branch ending at this node. Roughly speaking, if the STN is consistent, then a legal schedule representing this branch exists.

The proposed algorithm makes three assumptions:

Assumption 1. *The goal is considered achieved once all of the propositions $p \in G$ (where 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.*

We now describe TP-MCTS. Its pseudo-code is shown in Algorithm 1. It is intentionally simplified for clarity by ignoring two issues: (1) A search-depth parameter is used to constrain the depth of the search tree. Nodes deeper than this parameter are not considered. (2) The tree contains interleaved action nodes and state nodes. The pseudo-code treats them as a single node, although each action node may have multiple state-node children.

TP-MCTS starts by constructing a transformed problem and then repeats the following steps: *Search* to find the next action, *Schedule* the action, *Step* – apply the action, and *Update* the STN. *Search* is the main step, which modifies classic MCTS and consists of four stages: *Selection, Expansion, Evaluation,* and *Backpropagation*.

In the *Select* function, the tree is traversed starting from the root until reaching a new node. The UCB criterion (Auer, Cesa-Bianchi, and Fischer 2002) is used to select among actions (l.30), and sampling from $\mathcal{T}r$ is used to select the next state (l.31). *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. Notice that UCB entails that whenever we reach an existing node, we will sample a new action, if not all actions have been sampled, so far.

Expand properly generates and adds the new node, which contains the state reached, and an updated STN, obtained by updating the parent STN with events and constraints related to the relevant action, as described below.

The original MCTS algorithm now uses a simulation (rollout) step to evaluate the node's value. Instead, TP-MCTS uses the function *Evaluate* to estimate the node's value using a variant of temporal relaxed planning graph heuristic (TRPG) (Coles et al. 2008). Moreover, to reduce the

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

Algorithm 1: 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:          $a \leftarrow$ SEARCH($node(root, STN_{root})$)
7:          $t \leftarrow$ SCHEDULE($a, STN_{root}$)
8:          $root \leftarrow$ step($root, a$)
9:          $STN_{root} \leftarrow$ updateSTN($a, t, STN_{root}$)
10:     until Terminal($root$)
11: end procedure
12: procedure SEARCH(node $n$)
13:     while within computational budget do
14:         $(s_{new}, a_{new}, STN_{par.}) \leftarrow$ SELECT($n$)
15:         $n_{new} \leftarrow$ EXPAND($s_{new}, a_{new}, STN_{par.}$)
16:         $v \leftarrow$ EVALUATE($n_{new}$)
17:         BACKPROP($n_{new}, v$)
18:     end while
19:     return bestAction($n$)
20: end procedure
21: procedure EVALUATE(node $n$)
22:     $a_1, ..., a_k \leftarrow$ draw $k$ actions applicable in $n$
23:     for $i$ in $1, ..., k$ do
24:         $s_i \leftarrow$ step($n.State, a_i$)
25:         $v_i \leftarrow$ PTRPG($s_i$)
26:     end for
27:     return $\max_i v_i$
28: end procedure
29: procedure SELECT(node $n$)
30:     $a = \arg\max_{a'}$ a child of $_n$ $V(na') + c\sqrt{\frac{\log N(n)}{N(na)}}$
31:     $s$ = step($n.State, a$)
32:     if $n$ has a child $n'$ corresponding to $a$ and $s$ in the tree then
33:         Select($n'$)
34:     end if
35:     return $s, a, n.STN$
36: end procedure
37: procedure EXPAND(state $s$, action a, STN $STN$)
38:     if $a = a_{end}$ /*$a$ is an end action*/ then
39:         $STN_{new} = STN \cup t_{a_{new}} - endPlan \leq 0$
40:         $STN_{new} = STN_{new} \cup t_{a_{new}} \geq t_{pa}(a_{new})$
41:     else
42:         $STN_{new} = STN \cup$constraints (1)-(6)
43:     end if
44:     return $node(s, STN_{new})$
45: end procedure
46: procedure BACKPROP(node $n$,real $v$)
47:     Backpropagate $v$ to the parent until reaching the root. Average over states and maximize over actions.
48: end procedure
49: procedure SCHEDULE(action $a$, STN $S$)
50:     $t \leftarrow$ The earliest time the action can be executed according to $S$
51:     return $t$
52: end procedure
```

chance of under-estimating its value, *Evaluate* performs k estimates: It randomly selects k applicable actions, samples an effect for each action, and applies PTRPG (explained below) to this state. The node's value is the maximal value among the k estimates. These k samples are maintained in memory for later estimates of the node's value. The *Back-Prop* function backpropagates this value up the tree to the action applied at the root. Averaging is used between different action outcomes and *max* between different actions.

Once an action is returned from the *Search* function, the *Schedule* function determines the execution time of the action. The execution time is set to the earliest time the action can be scheduled according to its STN. Then, the *Step* function is called which applies the action online and generates a new STN that is updated to include the action constraints and its execution time. The root of the search tree uses this STN to select the next action.

To ensure time constraints regarding the duration of actions are met, if a_{start} was inserted into the tree at time t, a corresponding a_{end} must occur at $t + d$. We now specify these constraints and the ordering constraints. The STN starts with two events: $startPlan$ and $endPlan$. When we add a_{start} to the tree, we add two temporal variables/events to the STN: $start_a$ and end_a as well as the following constraints:

1. $startPlan - start_a \leq 0$
2. $start_a - endPlan \leq 0$
3. $start_{pa(a)} - start_a \leq 0$, where $pa(a)$ is the action at parent node of a.
4. $d \leq end_a - start_a \leq d$, where end_a is the end time of a.
5. $startPlan - end_a \leq 0$
6. $start_a - end_b \leq 0$ if b_{start} is an ancestor of a_{start} but b_{end} is not

The last constraint ensures that if a_{start} was inserted before some action b was completed, i.e., b_{start} appears before a_{start} but b_{end} does not, then a must start no later than b ends. The constraint $start_a + d - endPlan \leq 0$ is not included in the STN to remain consistent with Assumption 1, which allows the goal to be achieved even if not all actions have concluded. By slightly modifying these constraints, we can support more general controllable durations.

The PTRPG Heuristic

We estimate the value of a leaf node using TRPG – a well-known modification of the relaxed planning graph (RPG) heuristic for temporal domains. TRPG attempts to estimate the solution's execution time. It respects the relationship between start and end actions, i.e., an action A_{end} can be applied only if A_{start} has already been applied and a time corresponding to the action's duration has passed. Only then the end effects can be realized. Following the RPG, TRPG uses delete relaxation, where a literal remains true once it is achieved. Consequently, the TRPG heuristic does not consider the issue of mutex actions.

However, TRPG assumes actions are deterministic, and for TP-MCTS, we must adapt it to the case of stochastic actions. Once a deterministic action is applied, there is no need to reapply it since all its effects are already present following its first application. One could use a similar strategy with stochastic actions by assuming that all their possible effects hold. However, this is too strong a relaxation to be useful in interesting stochastic domains. When applying an action, our probabilistic TRPG (PTRPG) variant samples effects according to their probability. It continues to repeat sampling the effect once the action terminates to achieve all possible outcomes of the action. Effect samples are independent. For example, consider action a with a 2-second duration and a probabilistic end-effect. If a is first considered applicable at time 4, then at time 6 we sample a's end-effect for the first time. Then, from that point on, every 2 seconds, we resample a's end-effects.

Experiments

We compare TP-MCTS with the closest relevant algorithm – the Interwoven approach proposed by Mausam and Weld in their work (Mausam and Weld 2008). Since it uses RTDP, it can be adapted to the online setting. We also tested a variant of MW that uses MCTS instead of RTDP, since MCTS is often better suited for online planning. As with TP-MCTS, MW-MCTS uses PTRPG to estimate nodes, Assumption 3 is maintained in both versions. The same domain representation is used for both MW-MCTS and MW-RTDP.

The evaluation was performed on six domains. Three domains are structured domains that model real-world problems: NASA Rover, Machine Shop Simple adapted from MW's work and a new Stuck-Car domain that exhibits more interesting stochastic effects. 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.

Both 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[3].

Domains

Stuck Car(C). C cars are stuck in the mud. C agents must get them out of the mud before a deadline is reached. An agent can push a car, the car's gas pedal, or execute both actions simultaneously. Pushing the car has a higher success probability than pushing the gas, but the agent can get tired, forcing it to rest. Executing both actions simultaneously yields a better success probability than performing each action independently. However, execution time is longer and there is a probability the agent will get tired. Additionally, the agents can search for a rock that could potentially be placed beneath the car to aid in its release. The quality of the rock found by the agent influences the probability of getting the car out. If the found rock has poor quality, perhaps it is best not to invest time in placing the rock under the car.

Nasa Rover(R) A probabilistic variant of the well-known *NASA Rover* domain from the 2002 AIPS Planning Competition (Long and Fox 2003a) with R different rovers. Some rover actions' execution may fail. Each rover has two hands, 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.

Machine Shop(O) This 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.

Simple-x In this simple domain, there are x distinct actions, each of which accomplishes a unique goal upon completion. The duration of each action is four seconds. Importantly, there are no conflicting requirements or dependencies among the actions. The most efficient solution is to execute all actions simultaneously, resulting in a total execution time of four seconds. As x grows, MW's representation grows exponentially. However, the solution depth remains 1. With the transformed model, the action space grows linearly with x, as does the number of propositions (due to $InExecution$ propositions), while solution depth grows linearly.

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.

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 introduce the G parameter, which is the number of irrele-

[3]https://github.com/aiplan4eu/unified-planning

Table 1: Exp. Results: 1 sec. per search step. "**-**": Didn't compile within 8h. **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
Stuck Car(1)	(7,9)	87	33	**95**	10.1 (0.38)	8.9 (0.82)	10 (0.29)	0.003s	0.004s
Stuck Car(2)	(24,18)	**78**	14	31	13.8 (0.46)	13.1 (1.01)	11.7 (0.96)	0.002s	0.039s
Nasa Rover(1)	(33,27)	**97**	77	88	21.9 (0.59)	23.3 (0.57)	23.9 (0.6)	0.037s	0.04s
Nasa Rover(2)	(66,80)	**77**	0	0	27.2 (0.62)	-	-	0.105s	74.65m
Nasa Rover(3)	(99,159)	**59**	-	-	28.9 (0.5)	-	-	0.194	-
Machine Shop(2)	(30,26)	**94**	21	52	19.2 (0.33)	21.7 (0.85)	21.5 (0.55)	0.0224s	0.5s
Machine Shop(3)	(75,45)	**79**	0	0	21.8 (0.39)	-	-	0.0991s	28.9m
Machine Shop(4)	(148,68)	**18**	-	-	23.3 (0.65)	-	-	0.355s	-
Simple-10	(10,10)	**100**	**100**	45	**4 (0)**	**4.04 (0.04)**	14.1 (0.39)	0.003s	0.014s
Simple-11	(11,11)	**100**	51	39	4 (0)	13.8 (0.34)	13.6 (0.48)	0.003s	0.042s
Simple-12	(12,12)	**100**	0	29	4 (0)	-	10 (0.43)	0.003s	0.148s
Simple-13	(13,13)	**100**	0	22	4 (0)	-	11.3 (0.34)	0.003s	0.586s
Simple-15	(15,15)	**100**	0	28	5.4 (0.19)	-	11.4 (0.27)	0.004s	8.6s
Conc	(9,9)	**100**	0	0	10.7 (0.05)	-	-	0.003s	0.005s
Prob Conc+7	(11,5)	**96**	82	93	9.3 (0.9)	11.7 (0.19)	10 (0.24)	0.003s	0.046s
Prob Conc+8	(12,5)	**93**	47	92	9.25 (0.19)	13.6 (0.28)	10 (0.21)	0.003s	0.16s
Prob Conc+9	(13,5)	89	32	**95**	9.5 (0.22)	13.5 (0.28)	9.6 (0.22)	0.003s	0.61s
Prob Conc+10	(14,5)	**90**	28	90	**9.3 (0.22)**	13.1 (0.35)	9.5 (0.22)	0.003s	2.28s

Table 2: Results for 10 sec. per step. Names abbreviated.

	Success Rate (%)			Average Makespan (std)		
Problem	TP	MW R.	MW M.	TP	MW R.	MW M.
S. Car(1)	**98**	40	96	9 (0.29)	7.9 (0.73)	8.3 (0.26)
S. Car(2)	**76**	18	26	11.9 (0.43)	12.5 (1.23)	13.5 (0.66)
Rover(1)	**98**	75	93	21.5 (0.53)	23.4 (0.69)	21.1 (0.48)
Rover(2)	**84**	0	0	24.7 (0.45)	-	-
Rover(3)	**75**	-	-	27.5 (0.49)	-	-
Shop(2)	**97**	71	95	19.3 (0.34)	21.5 (0.37)	21.2 (0.28)
Shop(3)	**92**	0	0	21.2 (0.28)	-	-
Shop(4)	**56**	-	-	21.5 (0.38)	-	-
Sim-10	**100**	**100**	56	4 (0)	4 (0)	13.4 (0.45)
Sim-11	**100**	**100**	36	4 (0)	4 (0)	12.8 (0.52)
Sim-12	**100**	**100**	10	4 (0)	4 (0)	10.4 (0.65)
Sim-13	**100**	60	26	4 (0)	13.9 (0.26)	10.2 (0.4)
Sim-15	**100**	0	18	4 (0)	-	10.9 (0.43)
Conc	**100**	89	0	11 (0)	13.8 (0.09)	-
P.Co+7	**92**	85	88	9.2 (0.18)	8.7 (0.18)	10 (0.21)
P.Co+8	95	**98**	91	9.2 (0.19)	9.2 (0.19)	10.1 (0.24)
P.Co+9	91	**96**	87	9 (0.17)	10.4 (0.17)	10.2 (0.22)
P.Co+10	**92**	66	**92**	9.3 (0.2)	11.9 (0.22)	9.5 (0.17)

vant actions added 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.

Tables 1 and 2 present the experimental results for a decision-time budget of one and ten seconds, respectively. 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 Stuck Car(1), MW benefits from its shorter solution depth. This advantage no longer holds in the larger Stuck Car(2). MW MCTS has a higher success rate in Prob-Conc+9. Given the performance on other Prob-Conc variants, perhaps this is 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) As expected, MW compilation time is larger than TP-MCTS. There are many instances where 8 hours were not enough to generate the domain and others where minutes were required compared with fractions of a second. (4) As expected, additional search time leads to increased success rates for all algorithms. Sometimes slightly but, in a few cases, significantly. Occasionally, the average makespan improves, too, but only slightly. Yet, the general trends (1)-(3) hold in both tables.

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 difficulty is observed in other domains such as *Nasa Rover(2)*, *Machine Shop(2)*, and *Machine Shop(3)*.

Conclusion

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. This results in a richer yet more economical representation and a more scalable planning algorithm. There is much potential for future work improving the heuristic or possibly considering sampling action times using MCTS algorithms for continuous action spaces (Couëtoux et al. 2011).

Acknowledgement

This work was supported by ISF Grant 1651/19, the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Initiative, by the Marcus Endowment Fund, and by the Lynn and William Frankel Center for Computer Science at Ben-Gurion University of the Negev.

References

Altman, E. 1999. *Constrained Markov decision processes*, volume 7. CRC press.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2-3): 235–256.

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

Beaudry, E.; Kabanza, F.; and Michaud, F. 2010a. Planning for concurrent action executions under action duration uncertainty using dynamically generated bayesian networks. In *ICAPS 2010*, volume 20, 10–17.

Beaudry, E.; Kabanza, F.; and Michaud, F. 2010b. Planning with Concurrency under Resources and Time Uncertainty. In *ECAI*, volume 2010, 217.

Belotti, P.; Bonami, P.; Fischetti, M.; Lodi, A.; Monaci, M.; Nogales-Gómez, A.; and Salvagnin, D. 2016. On handling indicator constraints in mixed integer programming. *Comput. Optim. Appl.*, 65(3): 545–566.

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In McCluskey, L.; Williams, B. C.; Silva, J. R.; and Bonet, B., eds., *In ICAPS 2012*. AAAI.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Math. Oper. Res.*, 27(4): 819–840.

Beutler, F. J.; and Ross, K. W. 1986. Time-Average Optimal Constrained Semi-Markov Decision Processes. *Advances in Applied Probability*, 18(2): 341–359.

Bit-Monnot, A.; Ghallab, M.; Ingrand, F.; and Smith, D. E. 2020. FAPE: a Constraint-based Planner for Generative and Hierarchical Temporal Planning. *CoRR*, abs/2010.13121.

Buffet, O.; and Aberdeen, D. 2009. The factored policy-gradient planner. *Artificial Intelligence*, 173(5-6): 722–747.

Carreno, Y.; Petillot, Y.; and Petrick, R. 2022. Temporal Planning with Incomplete Knowledge and Perceptual Information. *arXiv preprint arXiv:2207.09709*.

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

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with Problems Requiring Temporal Coordination. In *AAAI*, 892–897.

Coles, A. J. 2012. Opportunistic Branched Plans to Maximise Utility in the Presence of Resource Uncertainty. In *ECAI*, volume 2012, 252.

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

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010b. Forward-Chaining Partial-Order Planning. In *ICAPS 2010*.

Couëtoux, A.; Hoock, J.-B.; Sokolovska, N.; Teytaud, O.; and Bonnard, N. 2011. Continuous Upper Confidence Trees. In Coello, C. A. C., ed., *Learning and Intelligent Optimization*, 433–445. Berlin, Heidelberg: Springer Berlin Heidelberg.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, 72–83. Springer.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence*, 49(1-3): 61–95.

Edelkamp, S.; and Hoffmann, J. 2004. PDDL2.2: the language for the classical part of IPC–4. In *ICAPS 2004*.

Foss, J. N.; and Onder, N. 2005. Generating temporally contingent plans. *Planning and Learning in A Priori Unknown or Dynamic Domains*, 62.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.*, 20: 61–124.

Jiménez, S.; Jonsson, A.; and Palacios, H. 2015. Temporal Planning With Required Concurrency Using Classical Planning. *ICAPS 2015*, 25(1): 129–137.

Keller, T.; and Helmert, M. 2013. Trial-Based Heuristic Tree Search for Finite Horizon MDPs. In *ICAPS 2013*.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *ECML*, 282–293.

Little, I.; Aberdeen, D.; and Thiébaux, S. 2005. Prottle: A Probabilistic Temporal Planner. 1181–1186. AAAI Press / The MIT Press.

Long, D.; and Fox, M. 2003a. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20: 1–59.

Long, D.; and Fox, M. 2003b. Exploiting a Graphplan Framework in Temporal Planning. In *ICAPS 2003*, 52–61. AAAI.

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

Panjkovic, S.; and Micheli, A. 2023. Expressive Optimal Temporal Planning via Optimization Modulo Theory. In *AAAI 2023*, 12095–12102. AAAI Press.

Puterman, M. L. 2005. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

Schoenauer, M.; Savéant, P.; and Vidal, V. 2006. Divide-and-Evolve: A New Memetic Scheme for Domain-Independent Temporal Planning. In *Evolutionary Computation in Combinatorial Optimization, 2006*, volume 3906 of *Lecture Notes in Computer Science*, 247–260. Springer.

Vidal, V.; and Geffner, H. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artif. Intell.*, 170(3): 298–335.