Online Planning in MDPs with Stochastic Durative Actions

Tali Berman[1] , **Ronen I. Brafman**[1] , **Erez Karpas**[2]
[1]Ben-Gurion University
[2]Technion
bermant@post.bgu.ac.il,brafman@bgu.ac.il, karpase@technion.ac.il

Abstract

Stochastic planning problems are typically modeled as Markov Decision Processes, in which actions are assumed to be instantaneous and applied sequentially. Yet, real-world actions often have durations and are applied concurrently. This paper presents an online planning approach that can deal with durative actions with stochastic outcomes. Our approach relies on Monte Carlo Tree Search with a new backpropagation procedure and temporal reasoning techniques that address the need to not only choose which action to execute, but also when to execute it. We also introduce a novel heuristic that combines reasoning about time and probabilities. Overall, we present the first online planner for stochastic temporal planning, solving a richer problem representation than previous work while achieving state-of-the-art empirical results.

1 Introduction

In various applications, the controlled system can perform multiple durative (i.e., non-instantaneous) actions concurrently. Examples include robots with multiple actuators, cooperative multi-agent systems, and smart homes. Often, such systems must respect temporal constraints such as deadlines (the meal should be ready by 5 pm) and time windows (the store is open 10 am to 6 pm), constraining both relative and absolute timing of actions in a successful plan. Classical temporal planning deals with these issues [Long and Fox, 2003; Schoenauer *et al.*, 2006; Vidal and Geffner, 2006; Coles *et al.*, 2010b; Bit-Monnot *et al.*, 2020; Panjkovic and Micheli, 2023], but assumes action effects are deterministic. Yet, in many applications actions can fail in various ways and have potential side effects. Although Markov decision processes (MDPs) capture stochastic actions, they assume that actions are applied sequentially, have instantaneous effects, and typically do not consider temporal constraints. Multi-agent extensions of MDPs allow for (concurrent) joint-actions [Bernstein *et al.*, 2002], but they are instantaneous and synchronized.

The combination of durative actions with stochastic effects forces us to consider interactions between temporal constraints, success probability, and failure modes. Suppose we must select between a short, risky action with 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. Without a deadline, an optimal policy would typically reapply the safe action until it succeeds. However, if there is insufficient time to try the safe action enough times, the risky action is better. Finding the optimal policy in this case requires reasoning both over temporal constraints (to understand what the deadline is), and stochastic effects (to understand how likely the risky action is to lead to a dead-end).

Planning with stochastic, durative actions was considered by [Little *et al.*, 2005; Buffet and Aberdeen, 2009; Mausam and Weld, 2008] only in the *offline* setting, which severely limits scalability and requires an explicit model. Moreover, past algorithms insert actions only in *pivot* points – points in time in which some action's execution terminates. Hence, they cannot solve problems with required concurrency and complex temporal constraints [Mausam and Weld, 2008].

In this paper, we define CoMDP+, a model that extends MDPs with actions with deterministic durations, concurrency, and temporal constraints. Our main contribution is TP-MCTS (Temporal Planning MCTS), an algorithm that combines Monte Carlo Tree Search (MCTS) [Coulom, 2006; Kocsis and Szepesvári, 2006] with classical temporal planning techniques. 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 makes the following algorithmic contributions: (1) We develop a novel heuristic, called PTRPG (probabilistic temporal relaxed planning graph), which estimates the probability of plan success from a given node, taking temporal information such as deadlines into account. We use it to replace rollouts, which provide poor information about node values in the presence of temporal constraints. (2) We develop a novel backpropagation method for MCTS, which manages information about time *intervals* instead of scalar values. (3) Combining the above, we develop a novel algorithm for deciding not only which action to execute but also *when* to dispatch it.

Overall, this paper presents: (A) The first online algorithm for stochastic planning with concurrent durative actions. (B) The first (online or offline) algorithm able to insert actions

in non-pivot points, and hence solve problems unsolvable by prior methods. (C) Better scalability than previous planners. (D) A comprehensive empirical evaluation on a problem set larger than prior work, including existing and novel domains, demonstrating improved scalability and scope.

Code, domains, additional results and discussion can be found at https://github.com/taliBerman5/TP_MCTS.

2 Background

Factored Markov Decision Process

We focus on *goal-oriented, factored* MDPs (fMDPs) [Boutilier *et al.*, 2000] specified using a variant of PPDDL [Younes and Littman, 2004]. fMDPs assume that states are assignments to variables. Goal-oriented MDPs assume a set of terminal goal states. We model them as a four-tuple: $\langle \mathcal{P}, \mathcal{A}, \mathcal{G}, s_0 \rangle$. $\mathcal{P}$ is a set of propositional variables that induce a state space $\mathcal{S}$ consisting of all truth assignments to $\mathcal{P}$.[1] 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* $\subseteq \mathcal{P}$ are a's preconditions, and *Eff* are a's effects, consisting of a set of triples (C, E, p). C and E are conjunctions of literals representing a context and an effect. $p \in (0, 1]$ is its probability. The set of contexts within *Eff* are mutually exclusive and exhaustive. The sum of probabilities of different effects E for each context C is 1.

Action a =*(Pre, Eff)* is applicable in state s only if $s \models$*Pre*. Let $(C, E_1, p_1), \ldots (C, E_k, p_k)$ be all triples in *Eff* such that $s \models C$ and $s \models Pre$. If a is applied in s then effect E_i will occur with probability p_i, and the resulting state s' will be identical to s on every proposition $q \in \mathcal{P}$ such that q 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 considered goal states.

Monte-Carlo Tree Search

MCTS [Coulom, 2006; Kocsis and Szepesvári, 2006] is a class of online, anytime, sampling-based search algorithms for sequential decision problems. In the time allocated for decisions, the algorithm performs multiple path-sampling iterations. Then, it selects the next action to perform, executes it, and the process is repeated.

In each iteration, MCTS traverses the current tree from the root down the tree until it reaches a node with an unexpanded child or a terminal state (*selection*). It adds a new child node to this leaf node (*expansion*). It estimates the added node's value by executing some default policy (*rollout/simulation*). Finally, it back-propagates the value up the tree, updating the value of all nodes on the path to the root (*back-propagation*).

Temporal Planning Concepts

A *simple temporal network* (STN) [Dechter *et al.*, 1991] models temporal constraints between events and supports efficient inference algorithms. It 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: $Y - X \leq \delta$, where $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$.

[1] We often abuse notation, using $\mathcal{P}$ to refer to all literals.

A solution to an STN S is called a *schedule*. It is a function $\sigma : \mathcal{T} \to \mathbb{R}^+$, assigning a time-point to each event in $\mathcal{T}$ such that all constraints in $\mathcal{C}$ are satisfied. If such a schedule exists the STN is called *consistent*.

Temporal planning (TP) problems that can be solved only if actions are applied concurrently are called problems with *required concurrency* [Cushing *et al.*, 2007]. For example, a shorter *makespan* (i.e., plan execution length) may be essential for meeting a deadline. Another classic example is *match-cellar*. A fuse must be replaced to fix the lights. Fuse repair requires light, too, obtainable by lighting a match. This example requires actions 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.

CoMDP

CoMDP [Mausam and Weld, 2008] extends fMDP by associating a deterministic duration with each action and allowing concurrent execution of non-interacting actions. Action preconditions must hold at their starting point, with effects obtained at their endpoint. Actions a and a' interact if: (1) their preconditions are inconsistent; (2) their effects are contradictory; (3) the preconditions of a contradict a possible effect of a'; or (4) an effect of a modifies a proposition influencing the transition probabilities of a'. We extend this model with start effects, overall conditions, end conditions, and deadlines.

3 Related Work

(Classical) temporal planning (TP) solves planning problems with concurrent actions that have deterministic action durations or durations confined to some interval [Fox and Long, 2003]. Recent work solves contingent domains with deterministic actions and non-deterministic sensing using search [Carreno *et al.*, 2022]. TP-MCTS does not deal with partial observability, but it models stochastic action effects and uses a different search technique in which temporal information is maintained in the search tree.

Unlike classical methods, Markov decision processes (MDPs) [Puterman, 2005] model stochastic actions. Semi-MDPs (SMDPs) extend them to model stochastic actions with stochastic durations. However, action execution in SMDPs 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/SMDPs seeking to maximize expected cumulative reward while satisfying constraints, typically in expectation only. The constraints are defined on $g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ where $\mathcal{S}$ are system states, and $\mathcal{A}$ are actions, and $g(s, a, s')$ is the cost of a transition from $s \in S$ to $s' \in S$ using $a \in A$. Durative actions could be defined by setting transition costs to equal action durations and constraining their sum to capture deadlines. This model, too, captures sequential execution only. Furthermore, temporal constraints have specific features, and TP-MCTS exploits this by using STNs and altering the backpropagation step, while allowing concurrent execution.

Few works tackle concurrent probabilistic durative domains, and all in the *offline* setting. Prottle [Little *et al.*, 2005]

develops an AND/OR tree 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 allows effects at (fixed) intermediate time points and outcomes with different durations. Our implementation does not support these extensions and considers only effects at the start and end of the action (as in classical TP) and identical, deterministic durations for different outcomes. Yet, our algorithm easily extends to handle these features. Importantly, Prottle restricts the decision epochs to pivot points, which implies incompleteness in the general case [Mausam and Weld, 2008]. FPG [Buffet and Aberdeen, 2009] uses factored policy gradients and also restricts action insertion to pivot points. Foss and Onder (2005) use STNs to capture temporal constraints, like us, but their uncertainty is not probabilistic and only over time. Beaudry, Kabanza, and Michaud (2010) 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.

The planner of [Mausam and Weld, 2008], denoted *MW*, addresses the setting closest to ours. MW reduces a CoMDP to an MDP whose actions are sets of non-interacting CoMDP actions, with a potential exponential blowup, greatly increasing the branching factor. The state is extended with variables denoting the remaining execution time of currently executing actions. An action can be applied only when an action in a currently executed action set terminates. They solve this MDP using real-time dynamic programming (RTDP) [Barto *et al.*, 1995]. TP-MCTS employs a compilation that only doubles the action set size, allows for flexible action scheduling not restricted to pivots and supports a more expressive CoMDP+ model.

4 The Model

A Goal-oriented CoMDP+ is a tuple $\langle \mathcal{P}, \mathcal{A}, \mathcal{G}, s_0, D \rangle$ where $\mathcal{P}, \mathcal{G}, s_0$ are as defined in an fMDP. $D \in \mathbb{R}^+$ is a deadline.[2] $\mathcal{A}$ 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 a's *start* effects and *end* effects, respectively. E_S and E_E are defined as in fMDPs, via sets of triples (c, e, p), with similar constraints on the sets of contexts c and their probabilities. We assume no effect in E_S contradicts P_O.
- d is the action duration.

If a is applied at time t, two instantaneous changes of the system's state occur: immediately after time t, the state changes according to E_S. Immediately after $t + d$ (when a ends), the state changes according to E_E. In both cases, the state changes as in the definition of effects in fMDPs.

The semantics of actions applied concurrently must be well defined, either by explicitly specifying the outcome of any two potentially conflicting actions, or by restricting such applications. The former can be quite laborious, and even more so when action timing can impact the effect. As in classical TP, we opt for the latter. We require certain actions to be mutually exclusive (mutex) [Smith and Weld, 1999], and assume that *otherwise*, an action's effects are not impacted by other actions. To simplify the description, our mutex definition is global and ignores contexts. Weaker, state-dependent mutex are easy to define, as well as treating contradictory effects as implying plan failure, but are not supported by our code.

Mutex:[3] 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 an overall condition of a' or vice-versa. Or, a and a' have contradictory potential effects at some state s.

Soft Mutex: $a' = (P', E', d')$ is *soft mutex* with $a = (P, E, d)$ if E'_E is inconsistent with P_O. That is, an end effect of a' violates the overall condition of a.

Action $a = (P, E, d)$ is *applicable* at time t if P_S is satisfied at time t, P_O is satisfied in the interval $(t, t + d)$, P_E is satisfied at $t + d$, and no action a' mutex with a is executed within the interval $[t, t + d]$. Notice that for P_O to be satisfied in the interval $(t, t + d)$, any action a' that is soft-mutex with a and which overlaps a, must end *after* a ends. Hence, actions whose preconditions are satisfied can be applied at any time point, including concurrently, as long as they do not conflict with or compromise the end and overall conditions of an executing action.

Finally, we assume that any effect in E_S that supplies some $q \in P_O$ is deterministic. We note that one can extend CoMDP+ with a reward function, but we focus on goal-oriented problems with the objective of maximizing goal achievement probability subject to the temporal constraints.

5 TP-MCTS

TP-MCTS works in two stages: Offline: transform the CoMDP+ domain into an fMDP with instantaneous actions plus temporal constraints. Online: select an action *and* its timing by developing a search tree whose root corresponds to the current state. Apply the action at the chosen time. Repeat.

TP-MCTS operates under the following assumptions: (1) The goal is achieved if all propositions $g \in \mathcal{G}$ are satisfied, even if this occurs during action execution. (However, with a simple change, we can require that all actions have terminated, as well.) (2) An executing action cannot be stopped. (3) The algorithm does not take into account the passage of time during the search, i.e., time is considered stopped during the search phase. (4) As in classical TP, actions cannot be started simultaneously and must be ϵ-separated.[4]

In practice, online decision time acts as a minimal ϵ value, which (in line with (3)), we assume is negligible w.r.t. ac-

[2] CoMDP+ supports *timed initial literals* (TILs), too. TIL (l, t) denotes that literal l will become true at time t. TILs can capture deadlines and time windows. They can be compiled into a CoMDP+ action executed at time 0 [Cresswell and Coddington, 2003].

[3] When clear from context, we abuse notation and use E_x to denote the set of possible effects $\{e | (c, e, p) \in E_x\}$.

[4] Otherwise, an action could use the simultaneous effect of a concurrent action as a precondition, which is undesirable semantically.

tion durations. True simultaneity can be achieved by defining explicit action combinations, sometimes called *collaborative* actions [Shekhar and Brafman, 2020].

5.1 Offline Preprocessing

We split every durative action a into two instantaneous actions a_{start} and a_{end} (called *snap* actions [Coles *et al.*, 2010a; Benton *et al.*, 2012; Jiménez *et al.*, 2015]) and a constraint that a_{end} is scheduled d time units after a_{start}. a_{start} takes care of the start conditions and effects, and a_{end} takes care of the end conditions and effects. 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. We can 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 a is executing. Its negation is a precondition of the start (respectively, end) of any action that is mutex (resp. soft-mutex) with a. To ensure that a_{end} occurs d time units after a_{start}, we use an STN to keep track of such constraints.

Formally, given a CoMDP+ $\langle \mathcal{P}, \mathcal{A}, \mathcal{G}, s_0, D \rangle$, generate the fMDP $\langle \mathcal{P}', \mathcal{A}', \mathcal{G}, s_0 \rangle$ (plus some auxiliary data), where $\mathcal{P}' = \mathcal{P} \cup \{InExecution(a) | a \in \mathcal{A}\}$; $\mathcal{A}' = \{a_{start} = (P_{a_{start}}, E_{a_{start}}) | a \in \mathcal{A}\} \cup \{a_{end} = (P_{a_{end}}, E_{a_{end}}) | a \in \mathcal{A}\}$. Assuming $a = (P, E, d), P = (P_S, P_O, P_E)$ and $E = (E_S, E_E)$:

- $P_{a_{start}} = P_S \cup (P_O \setminus E_S) \cup \neg InExecution(a)$
- $E_{a_{start}} = \{(c, e \wedge InExecution(a), p) \mid (c, e, p) \in E_S\}$
- $P_{a_{end}} = P_E \cup InExecution(a)$
- $E_{a_{end}} = \{(c, e \wedge \neg InExecution(a), p) | (c, e, p) \in E_E\}$

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}.

We assume that two copies of the same ground action cannot overlap.[5] 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. These conditions ensure that both strict and soft mutex conditions of CoMDP+ are respected in the generated fMDP.

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.

5.2 Online Solution

Online, TP-MCTS generates an initial root node containing state s_0 and *initSTN*, an initial STN with $startPlan$ and $endPlan$ events and a constraint $endPlan - startPlan \leq D$. It then repeats the following process until either the goal is achieved or the deadline passes: *Search* to find the next action for *root*; *Schedule* this action; *Step*: execute it; *Update root* to contain the current state and an updated STN.

Search

Starting with the current root, TP-MCTS uses the time allocated per decision online to construct a search tree with state and action nodes. State nodes contain a state, while action nodes contain an action and an STN representing the temporal constraints associated with the branch ending with this action. Both nodes maintain $n.V$, their value, whose form differs between the two TP-MCTS versions (discussed later), and $n.N$, their visit counter.

Search consists of the four stages of MCTS: *Select, Expand, Evaluate,* and *Backpropagate*. We assume the reader is familiar with MCTS. Hence, for space reasons, the pseudo-code focuses on the changes made w.r.t. MCTS, mostly ignores action nodes (and therefore, associates the action node's STN with its state node children) and the depth bound, and omits the Select, Expand, Step, and UpdateSTN functions. See our GIT for full pseudo-code.

Select (omitted – follows standard MCTS.) Traverse the tree from the root. Apply the UCB criterion [Auer *et al.*, 2002] to select among actions. Sample the next state based on the action's effect distribution. Terminate once reaching a goal state, a state whose STN is inconsistent (e.g., any solution will miss the deadline), or a new child state, s.

Expand (omitted). Add the new state s to the tree, and for every legal action a in s, add a child action node containing a and the STN obtained by updating the parent's STN with the relevant events and with constraints (c1)-(c4):

c1 $startPlan - a \leq 0$

c2 $a - endPlan \leq 0$

c3 $pa(a) - a < 0$; $pa(a)$ is the action at a's parent node.

c4 If a is a start action and end_a its end action then:

1. $end_a - a = d$; (and end_a is add to the STN)
2. For every end action $end_{a'}$ in the STN that has not been selected yet: $a - end_{a'} < 0$

(c3)+(c4.2) ensure that a is scheduled after any action selected earlier, and prior to any End action not yet selected.

Evaluate. MCTS' *evaluate* uses a simulation (rollout) step to assess the node's value. Instead, TP-MCTS uses *PTRPG*, a novel heuristic estimate of the probability of reaching the goal from this node in time.

Backpropagate. Information from the leaf node is backpropagated up the tree using two methods explained later.

Schedule

In standard MDPs, the only decision is *what* action to execute next. As in standard MCTS, we select the action with the highest value. However, in temporal domains, the value of an action is contingent on *when* it is executed. For example, scheduling it after the deadline has no value. Therefore, we must also select its execution time.

Ideally, we would like to choose an action a and an execution time t for which the probability of successfully terminat-

[5] Otherwise, planning is undecidable [Gigante *et al.*, 2020]

Algorithm 1 TP-MCTS (root-interval version)

1: **procedure** TP-MCTS(*CoMDP+* $\langle \mathcal{P}, \mathcal{A}, \mathcal{T}r, \mathcal{G}, s_0 \rangle$)
2: $\langle \mathcal{P}', \mathcal{A}', \mathcal{G}, s_0 \rangle$ = TransformTofMDP($\langle \mathcal{P}, \mathcal{A}, \mathcal{T}r, \mathcal{G}, s_0 \rangle$)
3: $root = (s_0, initSTN())$ //node's state+STN components
4: **repeat**
5: $a, V_a() \leftarrow$ SEARCH($root$)
6: $t \leftarrow$ SCHEDULE($a, V_a()$)
7: $root \leftarrow$ (Step(a, t),UPDATESTN($a, t, root.STN$))
8: **until** Terminal($root.state$)
9: **end procedure**
10: **procedure** SEARCH(*Node root*)
11: **while** within computational budget **do**
12: $n_{leaf} \leftarrow$ SELECT($root$)
13: EXPAND(n_{leaf})
14: $V() =$ EVALUATE($n_{leaf}, n_{leaf}.STN$)
15: BACKPROP($root, n_{leaf}, V()$)
16: **end while**
17: $a_{best} \leftarrow argmax_{\text{root actions } a} \max_t(a.V(t)/a.N)$
18: **return** $a_{best}, a_{best}.V()$
19: **end procedure**
20: **procedure** BACKPROP(*Node root*, Node n_{leaf}, *func* $V_{leaf}()$)
21: **for all** node n on branch from n_{leaf}'s parent to $root$ **do**
22: $\forall t \in [0, deadline]$, $n.V(t) \leftarrow n.V(t) + V_{leaf}(t)$
23: $n.N \leftarrow n.N + 1$
24: **end for**
25: **end procedure**
26: **procedure** SCHEDULE(*action* a, *func* $V_a()$)
27: $T_{\max} \leftarrow \arg\max_t V_a(t)$ //returns an interval
28: $t \leftarrow \min_t T_{max}$
29: **return** t
30: **end procedure**
31: **procedure** EVALUATE(*Node* n, *STN* stn)
32: $c \leftarrow PTRPG(n, stn, \mathcal{G})$
33: $I \leftarrow stn.legalRoot$
34: $\forall t \in I, V(t) \leftarrow c$; $\forall t \notin I, V(t) \leftarrow 0$
35: **return** $V()$
36: **end procedure**

ing is maximized. In that case, we would have an asymptotically optimal algorithm for problems with deadlines because (due to the deadline), the tree has bounded depth, and hence, leaf nodes would eventually be goal-achieving or dead-ends, and because, in the limit, the frequency MCTS samples each stochastic effect converges to its true probability.

Unfortunately, the best current algorithm for finding a policy that *always* succeeds given stochastic actions is polynomial in tree size, hence, exponential in its depth [Hunsberger and Posenato, 2016]. Our setting is much more complex: we must assess the maximal probability of success of each policy sub-tree, maximizing over possible policies, while checking which branches can consistently schedule common actions. This can be done, in principle, using mixed-integer programming, but 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. Consequently, our algorithm is *not* guaranteed to be asymptotically optimal, but its empirical performance will be shown to be superior to previous methods.

We consider two combinations of the *Evaluate*, *BackPropagate*, and *Schedule* steps: *earliest* and *root-interval*.

Earliest. Like UCT [Kocsis and Szepesvári, 2006], it maintains at each node a *value* V and a *visit* count N. Both initialized to 0. The estimated success probability of a new leaf node (computed by *evaluate*) is added to each of its ancestors' V value, 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 this action considers consistent.

Earliest relies only on the selected action node's STN. Hence it ignores constraints on the root action's execution time stemming from future actions in the plan. For example, suppose the root action a_{start} adds q, which a_{end} negates, and q is a precondition of a later action a'_{start}. If a_{start} is scheduled too early, the constraint that a'_{start} precedes a_{end} would not be satisfiable. *Root-Interval* uses the STNs of leaf nodes to deduce such information and propagate it upwards.

Root-Interval. In this version, each node n maintains a counter N and a function $V(t)$ from $t \in I_0 = [0, D]$ to $[0, 1]$, instead of a single value. $n.V(t)$ is an estimate of the probability the goal will be achieved if the root action leading to n is scheduled to t.

Evaluate initializes the function for leaf nodes using their STN and PTRPG's estimate as follows: Let n be a leaf node. Let p 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. (Denoted $STN.legalRoot$ in l.33.) For STNs, I is always an interval, efficiently computable from n's STN. For all $t \in I$, set $n.V(t) = p$. For all $t \notin I$ set $n.V(t) = 0$. Propagate this information backward to n's ancestors as follows: Increment their visit and (conceptually) update the value function point-wise – i.e., $\forall t \in I_0$: add $n.V(t)$ to every ancestor's $V(t)$. When *Search* terminates, it returns the action maximizing $\max_t V(t)/N$ (l.17). *Schedule* then selects the earliest time point achieving this maximal value (l.28). Because at leaf nodes I is an interval, at non-leaf nodes V can be represented as a set of constant-valued intervals. Each update is very efficient, but the set size at the root is worst-case linear in the number of leaf nodes, i.e., worst-case exponential in depth.

Step and Update

We apply the chosen action a at the selected time t (*Step* – omitted) and generate a new root node for the next decision (l.7). This node contains the state resulting from a's application in the real world, and an STN with the action that extends the current root node's STN with constraints (c2-c4) above and the constraint $a - startPlan = t$.

As noted, the constraints ensure a is scheduled after any previously selected action. And since *search* follows *step*, the next action is selected only after a is executed. So if at t we schedule a to $t + \delta$, no other action will be executed prior to $t+\delta$. During $[t, t+\delta]$, we can further develop the tree. The sub-tree corresponding to a's actual outcome can be re-used for the next decision, as in some MCTS implementations.

PTRPG

We estimate the probability of achieving the goal on time from a leaf node n using a heuristic function we developed that combines ideas from the temporal relaxed planning graph heuristic TRPG [Coles *et al.*, 2008], sampling, and Bayesian techniques. PTRPG is stochastic and is executed once for each leaf node. It maintains a set L of literals initialized to contain all literals in node n's state, and the current time t, initialized to the smallest consistent value for *endPlan* according to n's STN, and a set C of inapplicability constraints, restricting the application of End actions in the STN, not selected yet, until their earliest possible execution time.

At each iteration, all actions whose preconditions are contained in L that are not inapplicable according to C at time t are "applied". That is, for every effect context whose context conditions are contained in L, one effect e is sampled and added to L. Because L is typically an inconsistent set of literals, multiple effects corresponding to inconsistent contexts might be added. For every action a_{start} applied, we add to C the constraint that a_{start} and a_{end} cannot be applied before $t + d$, where d is the duration associated with a_{start}. Actions applied can enable other actions' application at the same time, and this continues until no new action can be applied at t. At this point, t is incremented to $t + d'$ where d' is the minimal duration of an action a_{start} applied at t. PTRPG terminates if either $\mathcal{G} \subseteq L$ or $t > D$.

Suppose we terminated with literal set L at time t_g, such that $\mathcal{G} \subseteq L$. We use t_g to derive a heuristic estimate of the success probability p using a Bayesian probabilistic modeling technique [Gelman *et al.*, 2013]: We transform from the time domain to the 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]. More specifically: given parameters a, b, c: Let D be the deadline. Define $(1)D' = D + c$; $(2)z_1 = \ln(\frac{t_g}{D'-t_g})$; $(3)z_2 = a * z_1 + b$; $(4)p = \frac{1}{1+\exp(-z2)}$. If $t_g > D$ we set $p = 0$. Otherwise, compute p with $a = -0.5, b = c = 1$.

Although noisy, t_g is almost always an underestimate of the time required to reach the goal. Hence, if $t_g > D$, it is highly unlikely to succeed and for this reason, we set $p = 0$. If $t_g = D$, we consider goal achievement minimally possible ($p = \frac{1}{1+\exp(-(a*\ln(D/c)+b))}$). As t_g gets smaller, there is more time to compensate for the relaxed assumptions made by the heuristic, and so p increases. By adjusting the parameters of the affine transformation, we can adjust the relationship between the heuristic's outcome and the probability of meeting the deadline, reflecting the properties of each particular domain. Here, we made no effort to optimize these values or adjust them to different domains. We selected simple constants that seemed to make sense and still got good results. Learning better parameters, possibly per domain, would be an interesting direction for future work.

6 Experiments

We compare the two TP-MCTS variants (*earliest* and *root-interval*) with MW, the closest relevant algorithm [Mausam and Weld, 2008]. MW-RTDP is the original MW variant that uses RTDP, adapted to the online setting. For a fairer comparison, we also tested MW-MCTS, which replaces RTDP with 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 1-4 hold for all versions.

		Success Rate (%)			
Problem	$(\lvert\mathcal{A}\rvert, \lvert\mathcal{P}\rvert)$	TP-MCTS		MW-RTDP	MW-MCTS
		earliest	root		
Stuck Car(1)	(7,9)	68	**91**	33	83
Stuck Car(2)	(24,18)	60	**81**	14	24
Hosting-1	(2,3)	0	**100**	0	0
Nasa Rover(1)	(33,27)	91	67	79	**92**
Nasa Rover(2)	(66,80)	**83**	79	0	0
Nasa Rover(3)	(99,159)	**71**	70	-	-
Machine Shop(2)	(30,26)	**96**	75	21	52
Machine Shop(3)	(75,45)	**84**	50	0	0
Machine Shop(4)	(148,68)	40	**48**	-	-
Simple-10	(10,10)	**100**	**100**	**100**	31
Simple-11	(11,11)	**100**	**100**	59	16
Simple-12	(12,12)	**100**	**100**	0	29
Simple-13	(13,13)	**100**	**100**	0	30
Simple-15	(15,15)	**100**	**100**	0	23
Conc	(9,9)	**100**	0	0	0
Prob Conc+7	(11,5)	**94**	**94**	72	64
Prob Conc+8	(12,5)	94	**95**	37	76
Prob Conc+9	(13,5)	**96**	86	24	88
Prob Conc+10	(14,5)	**93**	92	37	92
Hosting-2	(4,4)	0	**87**	Cannot	Cannot
P. Match Cell.(1)	(2,4)	**95**	90	Process	Process
P. Match Cell.(2)	(6,8)	**85**	**85**	and	and
P. Match Cell.(3)	(12,12)	**75**	72	Solve	Solve
P. Match Cell.(4)	(20,16)	**70**	56	"	"
P. Match Cell.(5)	(30,20)	**46**	46	"	"

Table 1: Success % with 1 sec./search step. "-": Didn't compile in 8h. **Bold**: Highest success rate. Ties broken based on avg. makespan. $|\mathcal{A}|, |\mathcal{P}|$ – # of actions and propositions.

Our comparison covers more domains than previous work. We consider five structured domains that model real-world problems: NASA Rover and Machine Shop, as used by [Mausam and Weld, 2008], Stuck-Car, a new domain with more interesting stochastic effects, and two new domains with required concurrency: Hosting and Probabilistic Match-Cellar. We also use three synthetic domains to study basic properties of the algorithms. We conducted experiments with two possible decision-time budgets: 1 and 10 seconds and domain instances of different sizes.

All algorithms were implemented in Python. Experiments were run on an AMD EPYC 7702P 64-Core Processor. Each experiment was repeated 100 times. Our TP-MCTS implementation supports domain representations written in the Unified Planning Framework [Kapellos *et al.*, 2023].

Domains

We describe the main properties of the domains used. Detailed descriptions appear in the supplementary material and their definition appears in our git repository.

Stuck Car(C). C agents must get C cars out of the mud 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

	Success Rate (%)			
Problem	TP		MW-RTDP	MW-MCTS
	earliest	root		
S. Car(1)	91	**94**	31	91
S. Car(2)	**69**	62	13	37
Hosting-1	0	**100**	0	0
Rover(1)	**91**	90	65	**91**
Rover(2)	**86**	50	0	0
Rover(3)	**78**	65	-	-
Shop(2)	**98**	88	64	82
Shop(3)	**74**	42	0	0
Shop(4)	**58**	23	-	-
Simple-10	**100**	**100**	**100**	22
Simple-11	**100**	**100**	**100**	27
Simple-12	**100**	**100**	**100**	7
Simple-13	**100**	**100**	**100**	11
Simple-15	**100**	**100**	0	18
Conc	**100**	**100**	87	12
P.Conc+7	90	**95**	84	48
P.Conc+8	91	91	**94**	53
P.Conc+9	92	**96**	91	55
P.Conc+10	**97**	91	68	62
Hosting-2	0	**93**	Cannot Process and Solve	Cannot Process and Solve
P.MatchC(1)	**93**	89		
P.MatchC(2)	**83**	81	”	”
P.MatchC(3)	**80**	56	”	”
P.MatchC(4)	**79**	47	”	”
P.MatchC(5)	**69**	61	”	”

Table 2: Results for 10 sec. per step.

release. The rock's quality influences the probability of success, and it may be better to drop a rock of poor quality.

Hosting-v. We must clean our house and prepare food before guests arrive. We can clean and cook at the same time. Cooking may result in a dirty floor that requires cleaning, later. There are two versions. The more complex one has *required concurrency*. In it, 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.

Prob Match Cellar(O). A probabilistic variant of the Match Cellar domain [Coles *et al.*, 2009] with required concurrency, in which fix-fuse actions can fail.

Nasa Rover(R) and **Machine Shop(O)**. Classical domains to which MW added action durations and stochastic effects. Some actions can fail, and different actions (e.g., using different machines) have different durations.

Simple-x. Each of x non-interacting actions with duration 4 achieves a unique goal. Best is to execute all actions simultaneously. As x grows, MW's representation grows exponentially, but solution depth remains 1, while our action space, solution depth, and the number of propositions grows linearly.

Conc. Designed to test the ability to provide maximal concurrency needed to meet a deadline. Four actions take 1 sec.; two take 2 sec.; one takes 4 sec.; and one takes 9 sec. Meeting the 9 sec. deadline requires executing four actions concurrently, one from each class, except between times 4 and 5.

Prob Conc+G. A probabilistic variant of Conc. with four actions: A deterministic action with duration 8 and three probabilistic actions with duration 4,2,1, all of which must succeed to achieve the goal. G irrelevant non-interacting actions are added to this action set.

Results

TP-MCTS's 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 for Rovers(2) and 28 minutes for Machine Shop(3), and times out on larger versions of these problems. Tables 1 and 2 show the success rate of each planner on 100 trials for 1 and 10 seconds per decision. Deadlines were set to the length of an optimal plan + time for one action failure, so as to be challenging. Our GIT contains additional results.

The following general trends emerge: (1) MW-MCTS almost always dominates MW-RTDP. 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 Nasa Rover(1), MW benefits from its shorter solution depth given 1sec. This advantage disappears in the larger Nasa Rover(2). MW-RTDP has a higher success rate in Prob-Conc+8 given 10sec. Given the performance on other Prob-Conc variants, this may be due to the variance in the success estimates. The impact of domain size on the algorithms is clearly visible in the three structured domains. (3) Additional search time almost always leads to increased success rates for all algorithms, but trends (1)-(2) hold in both tables. (4) Hosting-2 and Prob MC(i) are not solvable by MW's method due to the need for required concurrency.

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. 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. With more time, it slightly scales up. TP-MCTS requires deeper solutions, but MCTS combined with PTRPG can focus exploration on more promising paths. Similar behavior is observed in *Nasa Rover(2)*, *Machine Shop(2)*, and *Machine Shop(3)*.

Comparing *earliest* and *root-interval*, *earliest* performs better given less time when temporal constraints are not complex because it can expand more nodes. Thus, trying to start an action as soon as possible is often a good heuristic. However, with sufficient time, *root-interval* succeeds more often, and in domains with more complex temporal dependencies, such as Hosting, only it can solve the problem reliably.

Additional experiments (see GIT) show that: (1) When deadlines are relaxed, success rates increase, but makespan is rarely impacted. (2) PTRPG's evaluation formula can have a significant impact on success rates, Nevertheless, but the relative strength of all variants remains similar, with TP-MCTS remaining much stronger than the MW variants.

7 Summary

We presented the first online algorithm for planning in MDPs with durative, concurrent actions. TP-MCTS combines MCTS and temporal planning techniques, including Start/End actions, STNs, with a new backpropagation scheme and the PTRPG heuristic to both select and schedule actions online. TP-MCTS uses a rich yet economical representation, is more scalable than previous methods, and is the first such online/offline algorithm to handle required concurrency.

Acknowledgments

This work was supported in part by the Lynn and William Frankel Center for Computer Science at Ben-Gurion University of the Negev and by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Initiative of Ben-Gurion University of the Negev

References

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

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

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

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

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

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

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

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

[Boutilier *et al.*, 2000] Craig Boutilier, Richard Dearden, and Mois e s Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121:49–107, 2000.

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

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

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

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

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

[Coles *et al.*, 2010b] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. In *ICAPS 2010*, pages 42–49. AAAI, 2010.

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

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

[Cushing *et al.*, 2007] William Cushing, Subbarao Kambhampati, Mausam, and Daniel S. Weld. When is temporal planning really temporal? In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1852–1859, 2007.

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

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

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

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

[Gigante *et al.*, 2020] Nicola Gigante, Andrea Micheli, Angelo Montanari, and Enrico Scala. Decidability and complexity of action-based temporal planning over dense time. In *AAAI'20*, pages 9859–9866, 2020.

[Hunsberger and Posenato, 2016] Luke Hunsberger and Roberto Posenato. A new approach to checking the dynamic consistency of conditional simple temporal networks. In Michel Rueher, editor, *CP 2016*, volume 9892, pages 268–286. Springer, 2016.

[Jiménez *et al.*, 2015] Sergio Jiménez, Anders Jonsson, and Héctor Palacios. Temporal planning with required concurrency using classical planning. *ICAPS 2015*, 25(1):129–137, 2015.

[Kapellos *et al.*, 2023] K Kapellos, A Micheli, and A Valentini. AIPlan4EU: Planning and scheduling for space applications. In *17th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2023.

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

[Little *et al.*, 2005] Iain Little, Douglas Aberdeen, Sylvie Thiébaux, et al. Prottle: A probabilistic temporal planner. In *AAAI'05*, 2005.

[Long and Fox, 2003] Derek Long and Maria Fox. Exploiting a graphplan framework in temporal planning. In *(ICAPS 2003)*, pages 52–61. AAAI, 2003.

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

[Panjkovic and Micheli, 2023] Stefan Panjkovic and Andrea Micheli. Expressive optimal temporal planning via optimization modulo theory. In *AAAI 2023*, pages 12095–12102. AAAI Press, 2023.

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

[Schoenauer *et al.*, 2006] Marc Schoenauer, Pierre Savéant, and Vincent Vidal. Divide-and-evolve: A new memetic scheme for domain-independent temporal planning. In *EvoCOP 2006*, volume 3906 of *Lecture Notes in Computer Science*, pages 247–260. Springer, 2006.

[Shekhar and Brafman, 2020] Shashank Shekhar and Ronen I. Brafman. Representing and planning with interacting actions and privacy. *Artif. Intell.*, 278, 2020.

[Smith and Weld, 1999] David E. Smith and Daniel S. Weld. Temporal planning with mutual exclusion reasoning. IJCAI'99, page 326–333. Morgan Kaufmann Publishers Inc., 1999.

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

[Younes and Littman, 2004] Håkan LS Younes and Michael L Littman. PPDDL1.0: The language for the probabilistic part of IPC-4. In *Proc. International Planning Competition*, 2004.