Exploring Simultaneity: Learning Earliest-time Semantics for Automated Planning

Ángel Aso-Mollar[1], **Óscar Sapena**[1], **Eva Onaindia**[1]
[1] Valencian Research Institute for Artificial Intelligence (VRAIN),
Universitat Politècnica de València (UPV)
aaso@vrain.upv.es, ossaver@upv.es, onaindia@dsic.upv.es

Abstract

In this paper, we aim to explore the potential of learning parallel plans and handle the execution of simultaneous actions in Automated Planning, in the form of Markov Decision Processes. Our objective is to investigate the upsides and downsides when learning a policy that not only involves the action to apply but also when to apply it. Concretely, we focus on guiding an agent to learn *earliest-time* semantics, wherein actions are to be executed as early as possible. Our solution approximates this theoretical normal form of parallel plans with an MDP that models the execution of actions along with their time steps, and rewards actions to be executed as early as possible. We solved the MDP with several methods and proved that RL adheres very well to the earliest-time semantics in the solved problems for a variety of domains, instead of other classical learning techniques.

Introduction

This paper explores the adequacy of learning simultaneous action application in the context of planning. We investigate a novel approach for learning parallel action application using a Markov Decision Process (MDP) framework, in which the simultaneous application of a set of actions is regarded as a constructive procedure. Our definition of the problem as an MDP allows us to inject semantic planning knowledge in the form of a reward signal. In this work, we focus on a semantics that executes actions as early as possible, the **process** semantics, as defined in (Rintanen, Heljanko, and Niemelä 2006). State-of-the-art SAT planners like Madagascar (Rintanen 2014) explicitly comply with this semantics, so the ultimate objective of this work is to answer the question: is it possible to learn to plan in parallel following a certain semantics instead of explicitly embedding the semantics within a resolution schema, as it is the case of the Madagascar planner? As an exploratory approach, we focus on solving particular planning instances and investigate the strengths and weaknesses of learning what we call **process** plans (parallel plans that follow the **process** semantics) using different techniques.

Hegemonically in the literature, when one has wanted to deal with the goal-oriented planning problems using MDPs, a 1-1 mapping has generally been established between the MDP action space and the grounded planning operator space, either to solve concrete instances by using goal MDPs (Geffner and Bonet 2013a) or by learning generalist policies that work for several problems (Gehring et al. 2022; Rivlin, Hazan, and Karpas 2020; Ståhlberg, Bonet, and Geffner 2023). This restricts learned policies to work only in the context of sequential planning.

But what happens if we want to increase the complexity of the policies, so that they can produce parallel plans? What has been done so far in this context equates the MDP action space to the power set of grounded planning operators. This is the case of (Aberdeen, Thiébaux, and Zhang 2004), in the field of decision-theoretic military operations, or (Aso-Mollar and Onaindia 2024), in the field of generalized planning, with the concept of meta-operator.

This one-to-many correspondence becomes an issue for medium to large sized instances, since the generation of the MDP action space leads to a combinatorial explosion. This is a problem especially for model-based, dynamic programming approaches such as LRTDP (Bonet and Geffner 2003) or LAO* (Hansen and Zilberstein 2001). The significant increase of the MDP action space also impacts the goal search, as the number of options increases exponentially. Our work seeks to alleviate these problems by shifting the action space expansion to the state space, which may be benefited by certain model-free solving techniques such as Reinforcement Learning (Sutton and Barto 2018). Moreover, injecting semantic knowledge to the search alleviates the sparse-reward problem of learning to plan.

The purpose of this work is thus to define a novel MDP framework that allows to plan in parallel, and to inject parallel planning semantic knowledge into the learning process, in the form of a reward signal. We will compare two different methods for learning parallel plans using that model: (1) solving the MDP using dynamic programming; and (2) approximate the policy using Reinforcement Learning. A comparison will be made of the adequacy of each method for this task, using a state-of-the-art SAT planner, Madagascar. We will prove that it is possible to learn plans that follow specific semantics using our MDP framework, and we will discuss which approach is more suitable for that purpose. Results of the work also indicate that yielded plans are competitive with respect to Madagascar, even though our goal differs from directly competing against a planner.

This paper is structured as follows. The next section introduces the background including the main basic notions on classical planning and MDPs to support our approach. Section **Modeling process semantics in an MDP** presents the formalization of the MDP model establishing the link with a planning problem and following section proves several properties of the MDP. Section **Experimental evaluation** shows the empirical results, provides details on the implementation and auxiliary algorithms to undertake the experimentation and includes a discussion on the results. Finally, last section concludes and summarizes future work.

Background

This section introduces the basic notions on which our approach is built. In this work, we focus our attention to STRIPS operators (Fikes and Nilsson 1971).

Classical planning

A *planning problem* is a tuple $P = \langle F, O, I, G \rangle$, where F is a set of fluents denoting propositional state variables with values f or $\neg f$; O is a set of deterministic grounded operators, I is the initial state of the problem and G is the goal condition. A problem state s is a full assignment of truth values to F; we denote the state space of P as S, and $\forall s \in S, |s| = |F|$. A state $s \in S$ is *terminal* for P if $G \subseteq s$. An operator o is defined by its preconditions $\mathsf{pre}(o)$ and effects $\mathsf{eff}(o)$, which are both partial truth assignments of F. True and false effect assignments, respectively, are denoted by $\mathsf{eff}^+(o)$ and $\mathsf{eff}^-(o)$.

An operator $o \in O$ is *executable* in a state s if $\mathsf{pre}(o) \subseteq s$ and $\mathsf{eff}(o)$ is *consistent* (does not contain both f and $\neg f$ for any fluent). We define $exe_o(s)$, for an executable o, as the state obtained as follows: $exe_o(s) = (s \setminus \mathsf{eff}^-(o)) \cup \mathsf{eff}^+(o)$. The execution of a sequence of operators $\langle o_1, o_2, \ldots, o_n \rangle$ in a state s is denoted by $exe_{\langle o_1, o_2, \ldots, o_n \rangle}(s) = exe_{o_n}(\ldots(exe_{o_2}(exe_{o_1}(s)))\ldots)$. Let $P = \langle F, O, I, G \rangle$ be a planning problem and $\phi = \langle o_1, o_2, \ldots, o_n \rangle$ a sequence of operators such that $exe_\phi(I) = s$, $G \subseteq s$; ϕ is called a sequential plan for P.

We consider now plans that are sequences of *sets of operators*. Let O be a set of operators, s_0 an initial state, and $\Phi = \langle L_0, \ldots, L_{t-1} \rangle$ a sequence of sets of operators where $L_i = \{o_{ij}\}_{j=1}^{n_i}$, $o_{ij} \in O$. Assuming the $\forall$**-step** plan semantics defined in (Rintanen, Heljanko, and Niemelä 2006):

Definition 1. *Φ is a $\forall$-**step** plan for O and s_0 if there is a sequence of states $\langle s_0, \ldots, s_t \rangle$, such that for all L_i and every total ordering $\langle o_{i1}, o_{i2}, \ldots, o_{in} \rangle$ of L_i, $exe_{\langle o_{i1}, o_{i2}, \ldots, o_{in} \rangle}(s_i)$ is defined and generates the same unique state s_{i+1}. Abusing notation, we will also use $exe_{L_i}(s_i) = s_{i+1}$ to denote the execution of a set of operators L_i in state s_i (any total ordering of operators in L_i generates the same state), and $exe_\Phi(s_0) = exe_{L_{t-1}}(\ldots(exe_{L_1}(exe_{L_0}(s_0)))\ldots)$ to denote the execution of a sequence of sets of operators Φ.*

In (Rintanen, Heljanko, and Niemelä 2006), the authors characterize $\forall$**-step** plans by requiring that no operator falsifies the precondition of any other operator that is executed simultaneously with the following theorem:

Theorem 1. *Let O be a set of operators, s_0 a state and $\Phi = \langle L_0, \ldots, L_{t-1} \rangle$ a sequence of sets of operators $L_i \subseteq O$. Then Φ is a $\forall$-step plan for O and s_0 if and only if there is a sequence of states $\langle s_0, \ldots, s_t \rangle$ such that*

1. *$s_{i+1} = exe_{L_i}(s_i)$ for all $i \in \{0, \ldots, t-1\}$*
2. *for no $i \in \{0, \ldots, t-1\}$ and two operators $o, o' \in L_i$, there exists a fluent $f \in \mathsf{eff}(o)$ such that $\neg f \in \mathsf{pre}(o')$, and vice versa.*

The above statement about $\forall$**-step** plan semantics lays the principles of simultaneous action execution and yields the standard and most common definition of parallel plans for STRIPS operators (for example that of SAT-encodings (Kautz and Selman 1996; Robinson et al. 2009) or Graphplan (Blum and Furst 1997)). The $\forall$**-step** plan semantics amounts to saying that two operators o and o' can be executed in parallel if they do not *interfere*. In other words, a set of operators L_i can be simultaneously executed if they are pairwise independent.

Following, we define the **process** plans (Rintanen, Heljanko, and Niemelä 2006), which we will refer to as the *earliest-time* semantics, that imposes the condition over $\forall$**-step** plans that operators are executed as early as possible.

Definition 2. *Let O be a set of operators, s_0 a state and $\Phi = \langle L_0, \ldots, L_{t-1} \rangle$ a sequence of sets of operators, $L_i \subseteq O$; Φ is a **process** plan for O and s_0 if it is a $\forall$-**step** plan for O and s_0 such that there is no $i \in \{1, \ldots, t-1\}$ and $o \in L_i$ so that $\langle L_0, \ldots, L_{i-1} \cup \{o\}, L_i \setminus \{o\}, \ldots, L_{t-1} \rangle$ is a $\forall$-**step** plan for O and s_0 with the execution $\langle s'_0, \ldots, s'_t \rangle$ such that $s_j = s'_j$ for all $j \in \{0, \ldots, i-1, i+1, \ldots, t\}$.*

A process plan is a unique $\forall$**-step** plan that executes every operator as early as possible so it can be seen as a canonical form for $\forall$**-step** plans.

Given a planning problem $P = \langle F, O, I, G \rangle$, a $\forall$**-step** plan $\Phi = \langle L_0, \ldots, L_{t-1} \rangle$, $L_i \subseteq O$, with execution $\langle s_o, \ldots, s_t \rangle$, is *valid* for P if $exe_\Phi(I) = s_t$ is a terminal state for P.

Markov Decision Process

A deterministic goal-oriented MDP (Haddawy and Hanks 1998; Geffner and Bonet 2013b) with finite horizon is defined as $M = \langle \mathbf{S}, A, R, T, \mathbf{s}_0, \mathbf{S}_G, H \rangle$, where $\mathbf{S}$ is a set of states; A is a set of deterministic actions; $R : \mathbf{S} \times A \rightarrow \mathbb{R}$ is a reward function; $T : \mathbf{S} \times A \rightarrow \mathbf{S}$ is a transition function; $\mathbf{s}_0 \in \mathbf{S}$ is an initial state of the MDP; $\mathbf{S}_G \subseteq \mathbf{S}$ is a set of goal states and H is the horizon. H is defined to avoid endless loops.

At each time step $0 \leq t \leq H$, an agent takes an action $a_t \in A$ in state $\mathbf{s}_t$ among all available actions following a *policy* π that maps states into a probability distribution over actions. In our work, $\pi(a|\mathbf{s})$ is an stationary stochastic policy $\pi : \mathbf{S} \times A \rightarrow [0, 1]$.

For a state $\mathbf{s}_t$, the policy π outputs an action a_t with probability $\pi(a_t|\mathbf{s}_t)$, which applied to s_t returns $T(\mathbf{s}_t, a_t) = \mathbf{s}_{t+1}$ with reward $R(\mathbf{s}_t, a_t) = r_t$. The objective is to learn an optimal policy π^* that maximizes the expected cumulative discounted reward (formally shown in Equation (1)) where $\mathbf{s}_0$ is the initial state and $\gamma \in [0, 1]$ is the discount factor used to weight future rewards.

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t \geq 0}^{H} \gamma^t r_t \mid \mathbf{s}_0 \right] \quad (1)$$

Modeling process semantics in an MDP

This work aims to design a Markov Decision Process M for a given planning problem P such that every policy for M produces a $\forall$**-step** plan for problem P, and the optimal policy π^* for M produces a **process** plan for P. The idea is to link a planning problem P to a deterministic goal-oriented MDP M with a finite horizon by viewing the execution of parallel operators in the problem-solving of P as encapsulated states of M. Before formally defining the mapping P-M, we introduce a series of intermediate concepts, which we label as *process*, that will help us to establish the mapping.

Definition 3. ***(Process state)*** *Let $P = \langle F, O, I, G \rangle$ be a planning problem and S the set of all states of P. Given a planning state $s \in S$, a process state for P is an extension of s denoted as s^L, where $L \subseteq O$ is an executable set of operators in s; i.e., $exe_L(s)$ is defined.*

The notion of process state extends the concept of planning state by considering a candidate set of operators to be executed in state s.

Definition 4. ***(Process action)*** *Let $P = \langle F, O, I, G \rangle$ be a planning problem, we define two types of process actions for P: (1) (add o), for every $o \in O$; and (2) (timestep).*

For a process state s^L, actions of type $(add\ o)$ represent the inclusion of a new operator $o \in O$ in the set L; and the action $(timestep)$ represents the execution of L in s.

The two above definitions are used to formally establish the mapping between a planning problem P and the model we aim to define, which is a process MDP.

Definition 5. ***(Process MDP)*** *Given a positive integer k, a process MDP for a planning problem $P = \langle F, O, I, G \rangle$, with S as the set of all states of P, is a goal-oriented MDP $M_k^P = \langle \mathbf{S}, A, R, T, \mathbf{s}_0, \mathbf{S}_G, H \rangle$ such that:*

- $\mathbf{S} = \{s^L : s \in S, L \subseteq O$, *and* $exe_L(s)$ *is defined*$\}$*; i.e., the set of all process states for P*
- $A = \{(add\ o) : o \in O\} \cup \{(timestep)\}$*; i.e., the set of all process actions for P*
- $\mathbf{s}_0 = I^\emptyset$ *($L = \emptyset$ for the initial process state)*
- $\mathbf{S}_G = \{s^\emptyset : s \in S$ and $G \subseteq s\}$
- $T : \mathbf{S} \times A \rightarrow \mathbf{S}$ *such that:*
  1. $T(s^L, (add\ o)) = s^{L \cup \{o\}}$*, if* $exe_o(s)$ *and* $exe_{L\cup\{o\}}(s)$ *are defined and* $L \cup \{o\}$ *follows condition 2 of Theorem 1 (that is, there is no fluent in o that is negated in L, and there is no fluent in L that is negated in o)*
  2. $T(s^L, (timestep)) = exe_L(s)^\emptyset$*, if* $L \neq \emptyset$*;* $exe_L(s)$ *is uniquely defined by condition 1 of Theorem 1 (the successor process state is the result of executing the set of operators L in the planning state s)*
- $R = R_k^P$*, where*

$$R_k^P(s^L, a) = \begin{cases} 0 & a \neq (timestep) \\ \frac{|L|}{k} & a = (timestep), T(s^L, a) \notin \mathbf{S}_G \\ 1 + \frac{|L|}{k} & a = (timestep), T(s^L, a) \in \mathbf{S}_G \end{cases} \quad (2)$$

  and k is an integer to bound the reward function for convergence purposes.
- $H \in \mathbb{N}$

Definition 5 establishes a 1-to-many correspondence between S (planning states) and $\mathbf{S}$ (process states). A single planing state s is mapped to the set of all s^L, with L a set of planning operators for which $exe_L(s)$ is defined. Moreover, a planning operator o is directly mapped to a process action $(add\ o)$, and the set O is mapped to A plus the extra action $(timestep)$. Semantically speaking, every iteration starts with the initial planning state, denoted as the process state $I^\emptyset$, and the set of operators to be executed in s is built iteratively by adding a single operator o each time. The state $I^\emptyset$ is semantically defined as "no operator is yet to be executed to the planning state I". Given a state s^L, the application of $(add\ o)$ generates a new process state that includes o in the set of operators to be executed in s: $s^{L\cup\{o\}}$. Also, the application of action $(timestep)$ to a process state s^L represents the execution of L to the corresponding planning state s and, as a result, the current time step moves from t to $t+1$.

The reward definition encourages the early application of operators, as we will demonstrate later. Intuitively, it is better to include actions as early as possible because we reinforce the maximum insertion of operators at early points of execution. As other goal-oriented MDP approaches, our reward function is goal-conditioned determining whether a goal state of $\mathbf{S}_G$ is reached or not (Zhu et al. 2021). The $\forall$**-step** plan is a sequence of sets of operators where each set L_i is retrieved when executing L_i to s_i, that is, when applying $(timestep)$ to $s_i^{L_i}$, thus generating the process state ${s_{i+1}}^\emptyset$. The horizon H ensures that reaching the goal would be preferred over executing $(timestep)$ actions indefinitely.

Plans produced by a process MDP can be easily translated to the usual syntax for representing parallel plans in PDDL. For example, for a planning problem $P = \langle F, O, I, G \rangle$ and a valid $\forall$**-step** plan $\Phi = \langle L_0, L_1, L_2 \rangle$, such that $L_0 = \{o_1, o_3\}$, $L_1 = \{o_2\}$, $L_2 = \{o_1, o_4\}$, the corresponding process MDP sequentialization $\langle add\ o_1,\ add\ o_3,\ timestep,\ add\ o_2,\ timestep,\ add\ o_1,\ add\ o_4,\ timestep \rangle$ can be translated to:

```
[0] o1
[0] o3
[1] o2
[2] o1
[2] o4
```

Following with the example, Figure 1 shows different sequences of process actions with which a process MDP M for

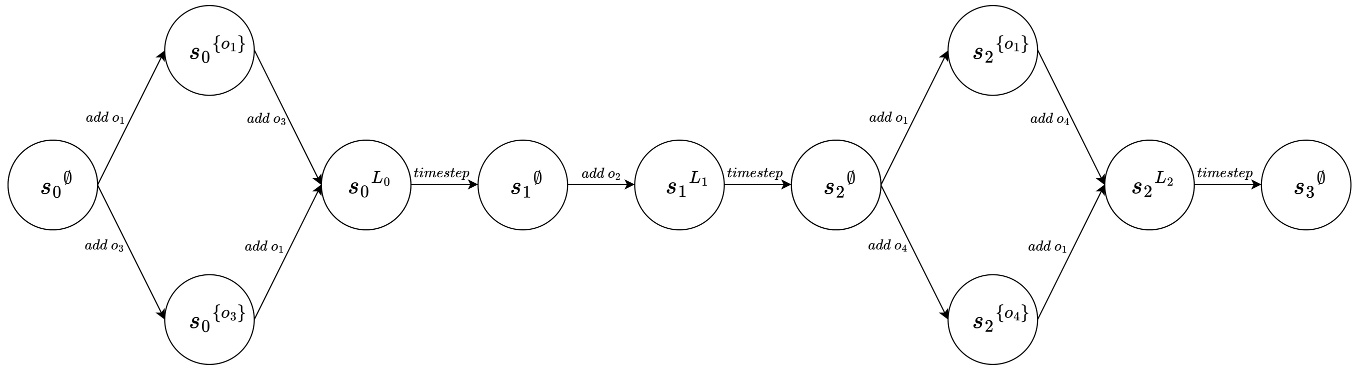


Figure 1: For a planning problem $P = \langle F, O, I, G\rangle$ let us assume that a valid $\forall$-**step** plan exists: $\Phi = \langle L_0, L_1, L_2\rangle$ with execution $\langle s_0, s_1, s_2, s_3\rangle$, $s_0 = I$, such that $G \subseteq s_3$ and $L_0 = \{o_1, o_3\}$, $L_1 = \{o_2\}$, $L_2 = \{o_1, o_4\}$. The graph shows four different ways to sequentially produce Φ using actions from the correspondent process MDP, with nodes/edges denoting MDP states/actions: $\langle add\ o_1,\ add\ o_3,\ timestep,\ add\ o_2,\ timestep,\ add\ o_1,\ add\ o_4,\ timestep\rangle$, $\langle add\ o_3,\ add\ o_1,\ timestep,\ add\ o_2,\ timestep,\ add\ o_1,\ add\ o_4,\ timestep\rangle$, $\langle add\ o_1,\ add\ o_3,\ timestep,\ add\ o_2,\ timestep,\ add\ o_4,\ add\ o_1,\ timestep\rangle$ and $\langle add\ o_3,\ add\ o_1,\ timestep,\ add\ o_2,\ timestep,\ add\ o_4,\ add\ o_1,\ timestep\rangle$.

problem P would find the plan Φ. We represent all the possible sequences that lead to Φ as an acyclic-directed graph, where nodes represent process states of M and edges represent process actions of M. The figure enhances the way a process MDP produces $\forall$-**step** plans for a given planning problem P, which consists of inserting one action at a time and executing all of the actions in a state when needed. The figure reflects that there are several different ways to make sequential decisions to produce the same $\forall$-**step** Φ.

We note that Definition 5 assumes there cannot be a situation in which a process state s of the MDP has no applicable actions. That is, we are assuming that an executable operator can always be added to the set L. However, this might not be the case in planning domains that feature dead-ends. Extending the state space to account for situations with dead-ends has been addressed in the literature of MDPs and Automaton Theory (Kolobov, Mausam, and Weld 2012; Björklund, Björklund, and Zechner 2014). This situation can be modeled with an extra *failure* state in the MDP that is reached when no operator is applicable in a state plus a transition from a dead-end state to that failure state.

Properties of process MDPs

In this section, we prove that a learned policy for a process MDP M adheres to the $\forall$-**step** plan semantics, and that the optimal policy of M guarantees that the plans follow the **process** plan semantics. Proofs for the case of encountering dead-end process states in M are not included but can be easily deduced from the theoretical properties presented in this section.

Proposition 1. *Given a positive integer k and a planning problem $P = \langle F, O, I, G\rangle$, a policy π for a process MDP $M_k^P = \langle \mathbf{S}, A, R, T, \mathbf{s}_0, \mathbf{S}_G, H\rangle$ defines a $\forall$-**step** plan Φ for O and I.*

Proof. Starting from state $\mathbf{s}_0 = I^{\emptyset}$, the set of available actions is $\{(add\ o) : o \in O$ is executable in $I\}$. Let $(add\ o_i)$ be the chosen action to apply from the distribution of the policy, $\pi(\cdot|\mathbf{s}_0)$. The application of $(add\ o_i)$ in $I^{\emptyset}$ leads to the process state $I^{\{o_i\}}$. Subsequently, there are two alternatives:

- The action chosen by the policy is $(add\ o_j)$, with $o_j \in O$, $o_i \neq o_j$, which leads to the process state $I^{\{o_i, o_j\}}$. In this case, the set $\{o_i, o_j\}$ follows condition 2 of Theorem 1 by definition of the transition function T of a process MDP for *add* actions.
- The action chosen by the policy is $(timestep)$, which leads to the process state $s_1^{\emptyset}$, $s_1 = (exe_{\{o_i\}}(I))$. In this case, by definition of the transition function T of a process MDP for a $(timestep)$ action, s_1 is a well-defined planning state compliant with condition 1 of Theorem 1.

The successive application of the actions chosen by the policy π will lead to a sequence $\Phi = \langle L_0, \ldots, L_{t-1}\rangle$ of sets of operators (the ones built with $(add\ o)$ actions) and to a sequence of states $\langle s_0, \ldots, s_t\rangle$ (resulting from the execution of $(timestep)$), which will be the result of the successive execution of the sets of the sequence Φ, starting from $s_0 = I$. Therefore, sequence Φ is, by definition and application of Theorem 1, a $\forall$-**step** plan for O and I.

□

Once proved that a policy for a process MDP yields a $\forall$-**step** plan starting from an initial state, we are now left to prove that the optimal policy π^* produces a valid **process** plan; that is, a $\forall$-**step** plan that reaches the goal and that follows the process plan semantics.

Theorem 2. *Given a sufficiently large positive integer k and a planning problem $P = \langle F, O, I, G\rangle$, the optimal policy π^* for M_k^P yields valid plans for P that follow the process plan semantics.*

Proof. The proof is divided in two parts. We will prove, firstly, that the optimal policy leads to a valid $\forall$-**step** plan (a $\forall$-**step** plan that reaches the goal) and, secondly, that the optimal policy leads to a $\forall$-**step** plan in which actions are inserted at their earliest time point, that is to say, a **process** plan. By proving these two statements, we will have proven that the optimal policy leads to a valid **process** plan.

From Proposition 1, any policy for a process MDP yields a $\forall$**-step** plan for O and I. We want to prove that the plan from the optimal policy is also valid for P, i.e., that it reaches the goal. By definition, an optimal policy maximizes the expected reward value from the initial state following the policy distribution. This means that more probability is given to the situations that return higher reward (see Equation 1). Restricting γ to the (open) interval $(0, 1)$ without loss of generality, we will demonstrate that the reward of the trajectories that do not reach a goal state is dominated by the reward of those that do reach a goal state.

We can characterize the reward of an arbitrary trajectory $\mathcal{T}$ that does not reach a goal state as:

$$\begin{aligned} r_{\mathcal{T}} = & \overbrace{0 + \cdots + 0}^{|L_0|} + \gamma^{|L_0|} \cdot \frac{|L_0|}{k} + \\ & + \overbrace{0 + \cdots + 0}^{|L_1|} + \gamma^{(|L_0|+|L_1|+1)} \cdot \frac{|L_1|}{k} + \dots \\ & \dots + \overbrace{0 + \cdots + 0}^{|L_i|} + \gamma^{(\sum_{t=0}^{i} |L_t|+i)} \frac{|L_i|}{k} \end{aligned} \tag{3}$$

As we can see in Formula 3, among the trajectories that do not reach a goal state, we focus on those whose last action is $timestep$ as these trajectories receive the highest reward according to Equation 2 (otherwise the trajectory would end with an add action that would yield a reward of 0).

Observe that in $\mathcal{T}$, $|L_0|$ add actions are executed, and then $timestep$, then $|L_1|$ actions, and then $timestep$, and so on until $|L_i|$ actions, and then $timestep$, but the resulting state is not a goal state (we would have added 1 to the sum otherwise). In this case, $\sum_{t=0}^{i} |L_t| + i = H$. Compactly:

$$r_{\mathcal{T}} = \frac{1}{k} \left[\sum_{j=0}^{i} \gamma^{(\sum_{t=0}^{j} |L_t|+i)} |L_i| \right]$$

We must guarantee that the reward signal $r_{\mathcal{T}}$ is dominated by the reward of a trajectory that does reach the goal. For that purpose, we will prove that $r_{\mathcal{T}}$ is dominated by the reward of reaching a goal state, using a suitable k. Then, a trajectory that reaches the goal would get more reward than any trajectory that does not.

The worst situation that can happen is that a goal state is reached at the last time step H. By forcing $r_{\mathcal{T}} < 1 \cdot \gamma^H$, which can trivially hold as k is a sufficiently large integer value, we have that $r_{\mathcal{T}} < (1 + \frac{|L|}{k}) \cdot \gamma^H$, so the reward of a non-goal trajectory can be dominated by the reward of reaching the goal (third row of Equation 2) using a suitable k value. This needs to hold for all possible non-goal trajectory $\mathcal{T}$. As we have a finite number of steps H, the set of trajectories is finite, so we can define k^* as the maximum k for every non-goal trajectory. With this, we assure the reward from trajectories that do not reach the goal to be always dominated by the reward of reaching the goal. Thus, trajectories that reach a goal state will have more reward than trajectories that do not, as its reward would include the goal reward. That is why the optimal policy will always converge to a trajectory where the goal is reached, yielding then a valid $\forall$**-step** plan, i.e., a plan that reaches the goal, q.e.d.

The second part that needs to be proven is that, knowing that the optimal policy yields a valid plan, whether this plan follows the process semantics or not. The proof is as simple as observing that earlier insertions for a particular action, a, yield more reward. Assuming that the action a can happen in two different steps i, j, with $i < j$, and knowing that the reward without a would be as follows:

$$\begin{aligned} & \overbrace{0 + \cdots + 0}^{|L_0|} + \gamma^{|L_0|} \cdot \frac{|L_0|}{k} + \\ & + \overbrace{0 + \cdots + 0}^{|L_1|} + \gamma^{(|L_0|+|L_1|+1)} \cdot \frac{|L_1|}{k} + \dots \\ & \dots + \overbrace{0 + \cdots + 0}^{|L_i|\text{, here is } i} + \gamma^{(\sum_{t=0}^{i} |L_t|+i)} \frac{|L_i|}{k} + \dots \\ & \dots + \overbrace{0 + \cdots + 0}^{|L_j|\text{, here is } j} + \gamma^{(\sum_{t=0}^{j} |L_t|+j)} \frac{|L_j|}{k} + \dots \end{aligned}$$

Having a in i would change the reward of time step i to $\gamma^{(\sum_{t=0}^{i} |L_t|+i+\mathbf{1})} \frac{|L_i|+\mathbf{1}}{k}$, while having a in j would change the reward of time step j to $\gamma^{(\sum_{t=0}^{j} |L_t|+j+\mathbf{1})} \frac{|L_j|+\mathbf{1}}{k}$. We can assume that inserting a results in only a single element difference for readability purposes, so what is left to prove is that the first reward dominates the second. As $i < j$, then

$$A = \sum_{t=0}^{i} |L_t| + i + 1 < \sum_{t=0}^{j} |L_t| + j + 1 = B$$

because $\sum_{t=0}^{j} |L_t| = \sum_{t=0}^{i} |L_t| + \sum_{t=i+1}^{j} |L_t|$. And then, $\gamma^A > \gamma^B$, because $\gamma \in (0, 1)$. Thus, $\gamma^A \cdot \frac{1}{k} > \gamma^B \cdot \frac{1}{k}$, q.e.d. The insertion of a in i yields more reward than in j, and as $i < j$, the optimal policy will insert actions as early as possible. □

Experimental evaluation

This section presents the experimental evaluation to calculate an optimal policy π^* for a process MDP and analyze whether the resulting plans adhere to the process semantics. To this end, we defined a process MDP for a set of planning problems from several IPC domains (see subsection *Domains*) and solved the MDPs with a model-based heuristic-search algorithm. For each problem, we also tested a model-free learning approach to approximate the optimal policy as well as a planner compliant with the process semantics:

(1) SAT-based planner. We solved the planning problems with the state-of-the-art SAT planner Madagascar[1] (Rintanen 2014), which enforces the earliest-time semantics in the search space of the problem defined through the PDDL domain model. We used two configurations of the Madgascar planner, **Mp** and **MpC**; **Mp** is more robust than **MpC** but often behaves poorly in small but hard combinatorial instances. In general, as no version is superior to the other, we opted for solving the problems with both of them.

[1]https://research.ics.aalto.fi/software/sat/madagascar/

(2) Heuristic search to solve the MDP. We solved the MDP using a model-based, dynamic programming algorithm called Labeled Real Time Dynamic Programming **(LRTDP)**, a heuristic-search DP algorithm for solving MDPs with full observability (Bonet and Geffner 2003). The policy returned by LRTDP for each problem is then used to obtain the plan that solves the problem.

(3) Learning an approximated policy. We learned a policy assuming the transition model of the underlying process MDP of the problems is unknown. To do so, we used a model-free state-of-the-art Reinforcement Learning (RL) algorithm, called Proximal Policy Optimization (Schulman et al. 2017) (**PPO**), an algorithm that constrains policy updates by clipping a surrogate objective function in the training phase. **PPO** approximates the problem policy following the reward function defined in Eq. 2.

Ultimately, the purpose of this experimentation is to evaluate the approximation of the parallel-plan policy obtained by **RLTDP** and **PPO** algorithms and compare the quality of the obtained plans with the plans returned by a SAT-based planner that complies with the earliest semantics.

This section is structured as follows; in the next subsection we present details of the domains and problems used in the experimentation; the next subsection explains our implementation of both **RLTDP** and **PPO**; subsection *Results* presents the obtained results and, finally, we discuss some relevant aspects of the experiments.

Domains

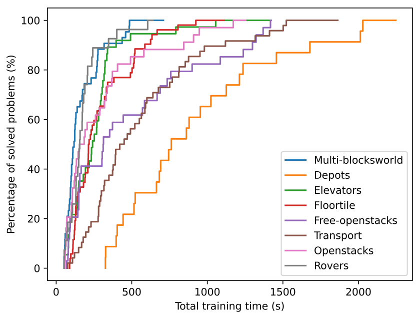
Domain	# Objects	# Instances
Multi-blocksworld	4-9	50
Depots	13-21	64
Elevators	8-15	48
Floortile	7-11	56
Free-openstacks	7-19	35
Transport	6-15	64
Openstacks	7-19	35
Rovers	3-9	60

Table 1: Number of objects present in the generated instances for each domain, and number of instances.

We generated a variety of medium-size problems for eight different domains from the International Planning Competition (IPC): *multi-blocksworld* (a *blocksworld* domain with several arms), *depots*, *elevators*, *floortile*, *openstacks*, *free-openstacks* (a special case of *openstacks* in which there is no restriction on the number of stacks), *transport* and *rovers*. Table 1 shows the number of instances tested for each domain and the range of the problem sizes as a function of the number of objects.

Heuristic search vs Reinforcement learning

LRTDP is an improved version of Real Time Dynamic Programming (RTDP) (Bonet and Geffner 2000). As stated in (Bonet and Geffner 2003), RTDP involves simulated greedy searches in which the heuristic values of the visited states are updated in a DP fashion, making them consistent with the



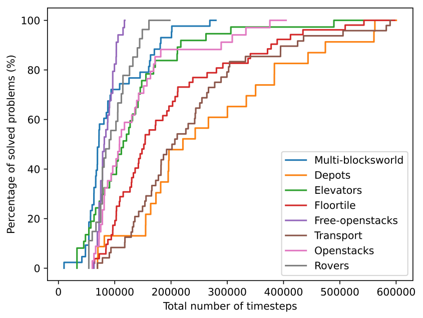


Figure 2: In these two graphs, we depict the total number of time steps taken by **PPO** to converge and the total training time on the x-axis, while on the y-axis, we represent the overall percentage of problems solved for each scenario, for every analyzed domain.

heuristic values of their possible successor states. LRTDP includes a labeling process that keeps track of the states for which the value function has already converged, which improves overall performance. Note that, in our case, we use LRTDP to solve a deterministic MDP, which can be regarded as a special case and a subset of non-deterministic MDPs.

We implemented the original algorithm of (Bonet and Geffner 2003), incorporating some of the adjustments exposed in the authors' Github repository[2], including adaptations for finite-horizon MDPs and permitting cycles in the search. We conducted **LRTDP** iterations for each problem with the h_{min} heuristic defined in the original paper, and $\epsilon = 10^{-4}$. Heuristic (h_{min}) calculations turned out to be computationally very expensive for large instances (this issue is also discussed in the original paper), so we finally opted for using $h = 0$ for retrieving the optimal policies. By using $h = 0$, every policy was retrieved in less than 10 seconds.

PPO is a state-of-the-art actor-critic RL algorithm to optimize policy functions by iteratively updating them and limiting changes to a given range, ensuring stable and efficient learning in complex environments, with the use of a clipped objective function. We note that PPO is used to learn the MDP process of one particular problem. **PPO** was run up to 600,000 time steps, 500 time steps per episode, 3,000 time

[2]https://github.com/bonetblai/mdp-engine

steps per batch, 5 updates per iteration, a 0.2 epsilon value, a discount factor of 0.99, and an entropy regularization coefficient of 0.01. We used full batch updates and single advantage estimation. We also included an early stopping to finish the training if the difference between the makespan of the plans was less than 0.05 for 10 complete and terminating episodes. Calculations were optimized using Pytorch Geometric library (Fey and Lenssen 2019). The reward function was defined with $k = 1000$ and the horizon $H = 500$. The value of k is empirically selected.

We used one-hot encoded vector representations for both states and actions. Using fancier vectorial planning state representations like those in (Dong et al. 2019; Gehring et al. 2022; Zhou et al. 2020) is beyond the scope of this work and we intend to investigate this issue in future work. Experiments were conducted on a machine with a Nvidia GeForce RTX 3090 GPU, a 12th Gen Intel(R) Core(TM) i9-12900KF CPU and Ubuntu 22.04 LTS operating system.

As **PPO** calculations are computationally far more expensive than **LRTDP**, we conducted an experiment to analyze the **PPO** training statistics. The upper plot of Figure 2 shows total training times of **PPO**. Each training lasted approximately between 20 seconds and 30 minutes, with an average of 396.15 secs. The lower plot of the figure shows the total number of time steps taken by **PPO** to converge. Results show that percentiles 50, 75 and 90 of the number of time steps for all the domains stand respectively in 123,038, 196,275 and 432,146 time steps (e.g., percentile 50 indicates that half of the MDP trainings were completed in fewer than 123,038 time steps).

Results

Madagascar is a state-of-the-art SAT planner that yields plans compliant with the process semantics definition (Rintanen 2014). For this reason, we set Madagascar plans as the ground-truth for process plans. We run both **Mp** and **MpC** configurations of Madagascar (Rintanen 2014); according to the planner documentation, either version may work better than the other depending on the domain.

In inference, we rolled out the policy obtained with **LRTDP** and **PPO** to solve the problems, until reaching the goal (or up to a horizon of $H = 500$ steps). Additionally, we implemented an algorithm that transforms a plan $\Phi = \langle L_0, \ldots, L_{t-1} \rangle$ into another one that contains the same actions and complies with the **process** semantics. This proprietary software (attached in the supplementary material) validates the plan Φ and searches the earliest schedule for each operator $o \in L_i$, determining whether $t = i$ is the earliest time at which o can be scheduled or otherwise o could be executed earlier. The problem of finding a **process** plan given an arbitrary **$\forall$-step** plan is a non-trivial and computationally expensive task as it involves finding the earliest insertion of every operator in the **$\forall$-step** plan. We will refer to this software as **process checker**.

Results of the experimental evaluation are shown in Table 2. The top table depicts the average makespan of the solution plans for all the problems by domain and the bottom table the average *process deviation* of the plans w.r.t. the earliest-time semantic plan calculated by the **process checker**. Let us assume an operator o is scheduled at time t in a plan Φ and that the **process checker** returns that the earliest time for o is $t' < t$. The *process deviation* is the sum of the deviations $t - t'$ for each operator of the plan. An earlier insertion of an operator in Φ does not necessarily imply a makespan deviation, but it sure implies a *process deviation*. The average values are shown along with their corresponding 95% confidence interval.

In the top part of Table 2, it is evident that **LRTDP** demonstrates poor performance in terms of plan makespan. It is noticeable that planners **Mp** and **MpC**, serving as baseline for this analysis, deliver much better results than the model-based approach. Moreover, **PPO** slightly outperforms both **Mp** and **MpC** in all the domains except *depots*, in which it only surpasses **Mp** and practically matches **MpC**. In summary, **PPO** achieves much better-quality plans than **LRTDP** when approximating the **process** semantics, even surpassing in most cases the baseline.

In the lower section of Table 2 we can notice that **LRTDP** column is empty. This is due to scalability issues, the **process checker** was unable to find a **process** plan for **LRTDP** because of the excessive length of its plans. It is also noticeable that **PPO** adheres very well to the semantics, showing an upper bound of *process deviation* of less than 1.25 actions and almost zero for the majority of domains, and compared to **Mp** and **MpC**, in which the *process deviation* is 0 in all domains except in the *rovers* domain. This indicates that **PPO** inserts actions in the plans as early as possible in the vast majority of cases.

Additionally to the *process deviation*, we also analyzed whether earliest-time insertions of plan operators lead to a shortening in the plan makespan. Results for **LRTDP** are not applicable either due to the search complexity of the **process checker**. For **PPO**, there is an average makespan reduction of 0 time steps for every domain but *floortile* and *depots*, in which it is less than 0.14 time steps. This shows that although there may be actions that can be applied earlier, the quality of the analyzed plans in terms of makespan is optimal or very close to optimal in every case.

We can conclude that **PPO** adequately approximates **process** semantics in almost all our experiments. We also empirically demonstrated that Reinforcement Learning outperforms model-based approximations, such as LRTDP, for this type of problems, which makes sense due to the increasing complexity of parallel planning compared to sequential planning.

Discussion

Several aspects of the experiments are worth discussing. First of all, it is observed that enforcing the process semantics in the reward function positively favors the RL model in some cases. The reason for this may be found in the very nature of some domains such as *Free-openstacks*, *Transport* or *Rovers*, where there is hardly restrictions for the simultaneous application of planning operators. The application of the earliest-time semantics in this type of domains favorably impacts the exploration of the RL algorithm and dealing with the exponential growth of states and actions in high dimensional spaces. However, there is also a counterpart to

Avg. makespan	**LRTDP**	**Mp**	**MpC**	**PPO**
Multi-blocksworld	112 ± 31.24	4.66 ± 0.87	4.39 ± 0.79	$\mathbf{3.57 \pm 0.60}$
Floortile	241.61 ± 11.53	5.16 ± 0.65	$\mathbf{4.64 \pm 1.53}$	$\mathbf{4.56 \pm 0.62}$
Transport	178.05 ± 41.58	4.64 ± 0.31	4.58 ± 0.28	$\mathbf{4.04 \pm 0.40}$
Openstacks	n/a	13.63 ± 1.37	14.03 ± 1.45	$\mathbf{12.23 \pm 1.40}$
Free-openstacks	16.00 ± 2.34	$\mathbf{6.71 \pm 0.32}$	$\mathbf{6.71 \pm 0.32}$	$\mathbf{6.71 \pm 0.32}$
Rovers	42.24 ± 23.62	4.52 ± 0.52	4.52 ± 0.52	$\mathbf{4.06 \pm 0.45}$
Elevators	195.52 ± 27.79	5.38 ± 0.94	$\mathbf{4.86 \pm 0.77}$	$\mathbf{4.64 \pm 0.77}$
Depots	247.75 ± 2.60	6.20 ± 0.70	$\mathbf{6.07 \pm 0.67}$	$\mathbf{6.16 \pm 0.77}$

Process avg. dev.	**LRTDP**	**Mp**	**MpC**	**PPO**
Multi-blocksworld	n/a	0.00 ± 0.00	0.00 ± 0.00	0.02 ± 0.03
Floortile	n/a	0.00 ± 0.00	0.00 ± 0.00	1.25 ± 1.01
Transport	n/a	0.00 ± 0.00	0.00 ± 0.00	0.10 ± 0.16
Openstacks	n/a	0.00 ± 0.00	0.00 ± 0.00	0.17 ± 0.24
Free-openstacks	n/a	0.00 ± 0.00	0.00 ± 0.00	0.14 ± 0.16
Rovers	n/a	0.50 ± 0.29	0.50 ± 0.29	0.08 ± 0.07
Elevators	n/a	0.00 ± 0.00	0.00 ± 0.00	0.47 ± 0.85
Depots	n/a	0.00 ± 0.00	0.00 ± 0.00	1.05 ± 0.82

Table 2: Average makespan of solution plans (top table) and process deviation of the plans (bottom table). **LRTDP**: Labeled Real-Time Dynamic Programming; **Mp** and **MpC**: Madagascar planner configurations; **PPO**: Proximal Policy Optimization

this situation; in other domains such as *Depots* or *Floortile* with more relations or restrictions between actions, forcing the execution of actions as soon as possible may result in the introduction of useless actions entangling what is truly important and resulting in an impairment of the algorithm convergence. This situation could explain the slightly worse performance in those domains.

Algorithms such as LRTDP fail to adequately approximate an optimal policy in parallel planning wherein the state space grows considerably. Model-based algorithms have proven useful to learn parallel plans in small problems and specific situations, such as in (Aberdeen, Thiébaux, and Zhang 2004), where the action space is defined as the power set of planning operators. This consideration has been perceived as a coarse approximation for approaching parallel planning in works such as (Aso-Mollar and Onaindia 2024); i.e., generating all possible combinations of operators explicitly defined in the action space makes this type of approximations not scalable at all. The novel MDP definition of this work equals the MDP action space to the planning action space plus one extra action, which makes it comparable to the current sequential plan learning models in terms of complexity. This mapping could be reduced even further by applying specific techniques aimed at the reduction of the action space in planning (Kokel et al. 2023).

We believe that this new way of learning to solve planning problems can be used in more complex techniques. The encoding of the proposed RL model in complex systems is almost immediate, since the action space of the MDP is fully equated (plus one action) to the grounded operator space of planning, as in the majority of sequential approaches. By shifting the issue of action-space explosion to the state space, the use of model-free methods enables more scalability. We believe that considering a parallel planning model can be an improvement in tasks that use sequential models trained with model-free methods.

Conclusions and Future Work

This paper represents a first attempt in learning to plan in parallel using a sequential decision-making scheme. We established a mapping of a planning problem to a process MDP that models parallel plans and rewards executing actions as earliest as possible. We trained a policy for a variety of problems using several model-free and model-based learning methods, and the results confirm that the model-free (RL) policies adhere to the behavior of SAT planners such as Madagascar.

A key strength of our approach is that the inclusion of parallel operators largely improves the model as there exist now different trajectories that lead to the same sequence of sets of operators. In addition, the inclusion of parallel execution of actions in the states themselves allows the problem to be treated as if it were a sequential resolution scheme, which is beneficial with respect to other approaches (Aberdeen, Thiébaux, and Zhang 2004; Aso-Mollar and Onaindia 2024). We acknowledge though that the approach is problem-dependent and not yet fully scalable due to the limitations of RL for approximating optimal policies. All in all, we believe it is a promising approach that is worth further investigation.

For future work, we aim to extend the model with partial-order plan (POP) semantics. We are also interested in investigating the application of process MDPs to Generalized Planning so as to alleviate the problem dependency and learn policies that generalize to large size problems. To this end, we intend to explore ways of representing process states in a vectorial form. All in all, this work represents a first step in the direction of non-sequential planning learning.

Acknowledgments

This work is supported by the Spanish AEI PID2021-127647NB-C22 project and Ángel Aso-Mollar is partially supported by the FPU21/04273.

References

Aberdeen, D.; Thiébaux, S.; and Zhang, L. 2004. Decision-theoretic military operations planning. In *Proceedings of the Fourteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'04, 402–411. AAAI Press. ISBN 1577352009.

Aso-Mollar, A.; and Onaindia, E. 2024. Meta-operators for Enabling Parallel Planning Using Deep Reinforcement Learning. arXiv:2403.08910.

Björklund, H.; Björklund, J.; and Zechner, N. 2014. Compression of finite-state automata through failure transitions. *Theoretical Computer Science*, 557: 87–100.

Blum, A. L.; and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1): 281–300.

Bonet, B.; and Geffner, H. 2000. Planning with Incomplete Information as Heuristic Search in Belief Space. In *Artificial Intelligence in Planning Systems (AIPS)*, 52–61.

Bonet, B.; and Geffner, H. 2003. Labeled RTDP: improving the convergence of real-time dynamic programming. In *International Conference on Automated Planning and Scheduling*, ICAPS'03, 12–21. AAAI Press. ISBN 1577351878.

Dong, H.; Mao, J.; Lin, T.; Wang, C.; Li, L.; and Zhou, D. 2019. Neural Logic Machines. *CoRR*, abs/1904.11694.

Fey, M.; and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4): 189–208.

Geffner, H.; and Bonet, B. 2013a. *A Concise Introduction to Models and Methods for Automated Planning: Synthesis Lectures on Artificial Intelligence and Machine Learning*, chapter 6. Morgan & Claypool Publishers, 1st edition. ISBN 1608459691.

Geffner, H.; and Bonet, B. 2013b. MDP Planning: Stochastic Actions and Full Feedback. In *A Concise Introduction to Models and Methods for Automated Planning*, Synthesis Lectures on Artificial Intelligence and Machine Learning, chapter 6, 79–96. Morgan & Claypool Publishers.

Gehring, C.; Asai, M.; Chitnis, R.; Silver, T.; Kaelbling, L. P.; Sohrabi, S.; and Katz, M. 2022. Reinforcement Learning for Classical Planning: Viewing Heuristics as Dense Reward Generators. In *ICAPS*, 588–596. AAAI Press.

Haddawy, P.; and Hanks, S. 1998. Utility Models for Goal-Directed, Decision-Theoretic Planners. *Computational Intelligence*, 14(3): 392–429.

Hansen, E. A.; and Zilberstein, S. 2001. LAO: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1): 35–62.

Kautz, H. A.; and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. In Clancey, W. J.; and Weld, D. S., eds., *AAAI 96, IAAI 96, Volume 2*, 1194–1201. AAAI Press / The MIT Press.

Kokel, H.; Lee, J.; Katz, M.; Srinivas, K.; and Sohrabi, S. 2023. Action Space Reduction for Planning Domains.

Kolobov, A.; Mausam; and Weld, D. S. 2012. A Theory of Goal-Oriented MDPs with Dead Ends. In de Freitas, N.; and Murphy, K. P., eds., *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, 438–447. AUAI Press.

Rintanen, J. 2014. Madagascar : Scalable Planning with SAT. https://research.ics.aalto.fi/software/sat/madagascar/.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12): 1031–1080.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized Planning With Deep Reinforcement Learning. *CoRR*, abs/2005.02305.

Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2009. SAT-Based Parallel Planning Using a Split Representation of Actions. In *ICAPS*. AAAI.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347. .

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning General Policies with Policy Gradient Methods. In Marquis, P.; Son, T. C.; and Kern-Isberner, G., eds., *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, 647–657.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2020. Graph neural networks: A review of methods and applications. *AI Open*, 1: 57–81.

Zhu, M.; Liu, M.; Shen, J.; Zhang, Z.; Chen, S.; Zhang, W.; Ye, D.; Yu, Y.; Fu, Q.; and Yang, W. 2021. MapGo: Model-Assisted Policy Optimization for Goal-Oriented Tasks. In Zhou, Z., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, 3484–3491. ijcai.org.