First-Order Dead End Situations for Policy Guidance in Relational Problems

Jun Hao Alvin Ng[1,2] , **Ronald P. A. Petrick**[1]
Edinburgh Centre for Robotics
[1]Department of Computer Science, Heriot-Watt University
[2]School of Informatics, University of Edinburgh
Edinburgh, Scotland, United Kingdom
{Alvin.Ng, R.Petrick}@hw.ac.uk

Abstract

Dead ends in some domains pose a major challenge to reinforcement learning (RL) algorithms. When dead ends are reached, a model-free RL method typically treats the observation as any other observations while a model-based RL method relies on the learned model to predict dead ends. We introduce a learning algorithm to learn effectively from observed dead ends and influence a policy to avoid dead ends in subsequent episodes. The algorithm is extended to learn first-order representations which can be generalised to unobserved state-action pairs and to new problems. This allows transfer learning by directly transferring the learned first-order knowledge to any problem such that dead ends can be avoided. A second extension avoids situations which lead to dead ends regardless of which action is executed, or dead end traps. We evaluate the efficacy of our method in two domains and show that our method significantly reduces the number of dead ends reached and achieves transfer learning.

1 Introduction

The ability to perceive, deliberate, and act is a form of intelligence necessary for solving complex sequential decision-making problems. Reinforcement learning (RL) [Sutton and Barto, 2018] is an approach to solve such problems if the model is not known. An agent acts in an environment and obtains some feedback, or observation, about its action in return. Through a trial-and-error iterative learning process it learns where to incrementally improve its policy which in turn provides more meaningful observations that allows the agent to improve the policy further.

In problems with dead ends—absorbing states with at least one unachieved goal—learning is less effective as no further observations can be obtained when a dead end is reached. Dead ends could also lead to potential dangers for the agent or its environment. In an exact representation of the problem, that is, a problem where each state is treated as a unique state and every observation only applies to the state in which it was obtained, RL can be inefficient as it takes several observations of encountering a dead end to learn a policy which can effectively avoid the dead end. In other words, the sample complexity is high.

If a problem is structured such that acting similarly in some states produces similar outcomes, then generalisation of the observations is possible and can be useful in reducing the sample complexity. Some problems are relational in nature where actions change the relations between an arbitrary number of interacting objects. The interaction between an agent and its environment is governed by the properties of objects and relations between objects. The repeated structure in the model can be represented compactly rather than enumerated explicitly in an exact representation. We refer to problems with a relational structure as **relational problems**. This perspective of looking at states and actions with no regards to objects, but rather variables, leads to a relational representation of the problem that can facilitate efficient learning.

By aggregating multiple different but similar situations as one abstract situation, learning can be made more efficient. The agent no longer needs to revisit the same state over and over again to learn the correct action as there are other states similar to it. Generalised knowledge learned from an observation in a particular state can instead be applied to other similar states for more informed decision making.

In this paper, we present a novel family of algorithms aimed at learning and avoiding the situations that lead to dead ends. Observed dead ends are represented in a first-order representation which generalises to unobserved situations. A policy considers these situations to avoid actions which are known to lead to dead ends. The first-order representation also generalises to other problems, allowing transfer learning between problems of the same domain regardless of the differences in their objects, initial states, and goal states.

The rest of the paper is organised as follows. First, we review related work on dead ends in decision-making problems and present the technical background. Then, we describe our method and present empirical results in two domains. Lastly, we conclude and discuss future work.

2 Related Work

Dead ends are an important area of study in planning which aims to produce plans or policies that avoid dead ends with certainty [Lipovetzky *et al.*, 2016], analyse goal reachability, and determine if problems are solvable [Steinmetz and Hoffmann, 2017]. Existing work uses the transition function to de-

tect or avoid dead ends during search while our work does not require knowledge of the transition function.[1] Instead, our work avoids situations, or state-action pairs, leading to dead ends; this does not require predictions of next states following action execution. While our focus is on online learning and execution, many offline planners consider dead ends in order to guide search with the use of heuristics [Cserna *et al.*, 2018; Ståhlberg *et al.*, 2021] or by pruning the search space [Camacho *et al.*, 2016; Kolobov *et al.*, 2010].

There is also prior RL work which is more closely related to our work. Safety in RL [García and Fernández, 2015] encompasses a diverse range of approaches such as those that change the optimisation criterion or modify the exploration policy to consider risk. The notion of safety (or risk) varies as well. For example, [Moldovan and Abbeel, 2012] defines safety in terms of the probability of returning to the initial state. [Fatemi *et al.*, 2019] defines dead ends as states from which all trajectories lead to an undesired terminal state with probability 1. Suppose that a negative reward is given for reaching undesired terminal states. Learning in a discounted MDP could be inefficient as the discounted return for dead ends are diminished as the trajectories leading to the terminal states can be many. [Fatemi *et al.*, 2019] proposes an exploration MDP which is a modification of the original MDP where there is no discount and the reward function gives -1 for reaching an undesired terminal state and 0 otherwise. Two Q-functions are learned in parallel using Q-learning, one for the original MDP and one for the exploration MDP. The behaviour policy is generated from the Q-function for the exploration MDP. Our work does not rely on the Q-function to avoid dead ends. Furthermore, our approach is able to generalise over unobserved states and problems.

3 Preliminaries

Here we present relevant background on representations for relational problems and RL.

States and Actions. States and actions are ubiquitous in decision-making problems. A state can be described with a conjunction of literals where a literal is a state predicate or a negated state predicate.

Definition 1 (State Predicates)**.** *A state predicate* $\mathtt{p}(x_1, \ldots, x_n)$ *consists of a symbol* $\mathtt{p}$ *and possibly some terms* $x_1, \ldots, x_n$*. Each term is associated with a type and holds an object of the same type. A state predicate represents a fact, a property of an object, or a relation between two or more objects. A symbolic state predicate has variables in its terms instead of objects. It can be ground to a state predicate by substituting variables with objects.*

Given a set of state predicates $\boldsymbol{P}$, a state s is represented as $s = \bigwedge_{i=1}^{|\boldsymbol{P}|} p_i$ where p_i is the i-th literal.[2] Assuming that all possible combinations of state predicates or their negation are valid states, then the size of the set of states $\boldsymbol{S}$ is $|\boldsymbol{S}| = 2^{|\boldsymbol{P}|}$.

[1] We assume that the preconditions of actions are known though this is not a hard requirement.

[2] A symbol with boldface denotes a set and $|\cdot|$ denotes the cardinality of a set.

Similar to state predicates, an action $\mathtt{a}(y_1, \ldots, y_n)$ consists of a symbol $\mathtt{a}$ and possibly some terms $y_1, \ldots, y_n$ which are objects while a symbolic action has variables as terms.

Markov Decision Process. Markov decision processes (MDPs) model fully-observable environments for sequential decision making under uncertainty.

Definition 2 (Markov Decision Process)**.** *A finite-horizon MDP is a tuple* $(\boldsymbol{S}, \boldsymbol{A}, \mathscr{T}, \mathscr{R}, s_0, H, \gamma)$ *where* $\boldsymbol{S}$ *is a set of discrete states,* $\boldsymbol{A}$ *is a set of discrete actions,* $\mathscr{T} : \boldsymbol{S} \times \boldsymbol{A} \times \boldsymbol{S} \to [0, 1]$ *is the transition function which defines a probability distribution over possible next states after executing an action,* $\mathscr{R} : \boldsymbol{S} \times \boldsymbol{A} \to \mathbb{R}$ *is the reward function which specifies rewards for executing actions in states,* s_0 *is the initial state,* H *is the time horizon or the maximum number of time steps, and* $\gamma \in (0, 1]$ *is the discount factor.*

Relational Markov Decision Process. The transition and reward functions can be represented by tables or tabular forms. $\mathscr{T}$ is a $|\boldsymbol{S}| \times |\boldsymbol{A}| \times |\boldsymbol{S}|$ matrix where an element stores $\mathscr{T}(s'|s, a)$, the probability of observing s' after executing a in s. Similarly, $\mathscr{R}$ is a $|\boldsymbol{S}| \times |\boldsymbol{A}|$ matrix which stores the reward $\mathscr{R}(s, a)$ for every state-action pair. This exact or flat representation makes no assumptions about the structures of $\mathscr{T}$ and $\mathscr{R}$. For large state-action spaces, tabular forms are impractical as the number of elements scales exponentially with the number of state predicates.

Factored MDPs [Boutilier *et al.*, 2000] are one approach to represent large scale MDPs compactly if $\mathscr{T}$ and $\mathscr{R}$ are structured. First, $\mathscr{T}$ is a factored transition function if the value of each state predicate is determined independently of each other, conditioned on the state and action:

$$\mathscr{T}(s'|s, a) = \prod_i \mathscr{T}(p'_i|s, a), \tag{1}$$

where $\mathscr{T}(p'_i|s, a)$ is a discrete probability distribution for a state predicate $p_i \in \boldsymbol{P}$ and p'_i is the value of p_i in s' (i.e., at the next time step). If $\mathscr{T}(p'_i|s, a)$ depends only on a small number of state predicates $\boldsymbol{P}^- \subset \boldsymbol{P}$, then $\mathscr{T}$ is expressed as:

$$\mathscr{T}(s'|s, a) = \prod_i \mathscr{T}(p'_i|\boldsymbol{P}^-, a). \tag{2}$$

Second, $\mathscr{R}$ is a factored reward function if it can be decomposed as a sum of localised reward functions, each of which depends on an action and a subset of $\boldsymbol{P}$. A factored MDP is an MDP but with factored transition and reward functions. A relational Markov decision process (RMDP) is a first-order representation of a factored MDP which generalises over objects through the use of variables and represents relational problems. We use the formalism of RMDPs from [Mausam and Weld, 2003].

Definition 3 (Relational Markov Decision Process)**.** *An RMDP is a tuple* $(\boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{A}}, \boldsymbol{O}, \mathcal{T}, \mathcal{R}, s_0, H, \gamma)$ *where* $\boldsymbol{\mathcal{C}}$ *is a set of classes or types,* $\boldsymbol{\mathcal{P}}$ *is a set of symbolic state predicates,* $\boldsymbol{\mathcal{A}}$ *is a set of symbolic actions,* $\boldsymbol{O}$ *is a set of objects and each object is associated with a type* $C \in \boldsymbol{\mathcal{C}}$*,* $\mathcal{T}$ *is the parameterised transition function,* $\mathcal{R}$ *is the parameterised reward function,* s_0 *is the initial state,* H *is the time horizon, and* γ *is the discount factor.*

An MDP can be constructed from an RMDP where $\boldsymbol{P}$ is the grounding of $\boldsymbol{\mathcal{P}}$ with $\boldsymbol{O}$, $\boldsymbol{S}$ is $\wp(\boldsymbol{P})$, $\boldsymbol{A}$ is the grounding of $\boldsymbol{\mathcal{A}}$ with $\boldsymbol{O}$, and $\mathscr{T}$ and $\mathscr{R}$ are the grounding of $\mathcal{T}$ and $\mathcal{R}$, respectively. $\wp$ denotes the power set.

Relational Dynamic Influence Diagram Language (RDDL). Planning languages are human-interpretable languages used to write domains and problems. RDDL [Sanner, 2011b] is a planning language used in the last three International Probabilistic Planning Competitions (IPPCs) in 2011 [Sanner, 2011a], 2014 [Grzes *et al.*, 2014], and 2018. Semantically, RDDL describes parameterised DBNs extended with an influence diagram. We use RDDL to write problems which are represented by RMDPs.

Reinforcement Learning (RL). When the transition function is not known, RL can be used for sequential decision making. The sample complexity of an RL algorithm is the number of observations needed to achieve near-optimal results. The learning objective is to find a policy π which maximises $\sum_{t=0}^{H} \gamma^t r_t$, where r_t is the immediate reward received at time step t. π can be generated from the Q-function which gives the expected return, or Q-value, for executing an action a in a state s:

$$Q_H^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{H} \gamma^t r_t \,\middle|\, s_0 = s, a_0 = a \right]. \tag{3}$$

TD learning methods update their estimates of the Q-function given observations (s_t, a_t, r_t, s_{t+1}). TD(λ) methods average over n-step returns by using eligibility traces. The update rule for the Q-function is given by:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha \delta_t e_t(s_t,a_t), \tag{4}$$

$$\delta_t = r_t + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t), \tag{5}$$

where α is the learning rate, $e_t(s,a)$ is the eligibility trace for (s,a), and δ_t is the TD error at time step t.

4 Learning from Dead Ends

We propose a method to learn from dead ends in relational problems formulated as an RMDP where the transition function is unknown. Our method is used in conjunction with any online RL algorithm and records observed dead ends in order to avoid them in future episodes.

Definition 4 (Terminal States, Goals, and Dead Ends)**.** *A terminal state is an absorbing state in which executing any action will only lead back to itself; the immediate reward for executing any action in a terminal state is zero. A problem has at least one goal which is represented by a state predicate or* ***goal predicate****. A goal state is a terminal state where all goal predicates hold. A terminal state where at least one goal predicate does not hold is a dead end.*

Our definition of dead ends is different from [Lipovetzky *et al.*, 2016] which defines dead ends as states where the goal state is unreachable. We consider only terminal states as dead ends (i.e., at least one goal is not achieved and executing any action does not change the state). If a goal can no longer be achieved, then the goal state is unreachable. In [Lipovetzky *et al.*, 2016], this is a dead end. However, since some problems have more than one goal, there remains incentive (i.e., immediate rewards) to attempt to achieve the remaining goals. Thus, we do not consider states where the goal state is unreachable as dead ends. In other words, we are interested in a different type of problem from [Lipovetzky *et al.*, 2016], where agents can attempt to achieve the remaining goals even when some of the goal states are not reachable. Our method learns and avoids the situations leading to dead ends in this context. We assume that a goal-directed policy exists such that dead ends can be avoided with certainty.

Definition 5 (Dead End Situation)**.** *A dead end situation χ describes a situation where executing an action a in a state s has a non-zero probability of reaching a dead end. χ is either (s,a) or $(\bar{s},\bar{a})$ where $\bar{s}$ is the abstract state of s and $\bar{a}$ is the symbolic action of a.*

It might seem straightforward to simply define a reward function such that a negative reward is given for reaching dead ends. However, as noted by [Fatemi *et al.*, 2019], this is not a general solution and can worsen performance in some domains. When a Q-function approximation is used and it has not yet converged, the negative rewards and Q-values are generalised to other state-action pairs such that the generated policy avoids them even if a trajectory leading to the goal must pass through them. In our empirical results, we observed that this is indeed the case for one of the domains tested.

An alternative method to learn from dead ends is to record every dead end situation in a failure buffer $\boldsymbol{\chi}$. Given an observation (s,a,r,s') where s' is a dead end, the dead end situation $\chi = (s,a)$ is added to $\boldsymbol{\chi}$. A policy generated from a Q-function is augmented with an auxiliary rule: do not select an action a in a state s if (s,a) is in $\boldsymbol{\chi}$. There are two issues with this naive method: (1) lack of generalisation because only previously observed dead ends can be avoided, and (2) high computational and space complexities in problems where there are many dead ends. The space complexity for storing $\boldsymbol{\chi}$ is $O(|\boldsymbol{\chi}|)$. The time complexity for checking if a state-action pair is in $\boldsymbol{\chi}$ is $O(|\boldsymbol{\chi}|)$. This cost is incurred in every time step but can be reduced by binning dead end situations according to their actions.

Example 1 (Dead End in `Triangle Tireworld`)**.** *In the `Triangle Tireworld (TT)` [Little and Thiebaux, 2007] domain, a vehicle moves in a grid environment to reach a goal location (indicated by the goal predicate* `GOAL_LOCATION`(WP)*).*[3] *A particular problem for `TT` is illustrated in Figure 1. There is a probability of* 0.5 *of getting a flat tire when moving. The tire needs to be replaced with a spare tire; if there isn't one, a dead end is reached. The vehicle can load the spare tire if there is one at its current location. A dead end is reached if the vehicle has a flat tire (*$\neg$`not_flattire`*), it does not have a spare (*$\neg$`hasspare`*), and its current location WP does not have a spare (*$\neg$`spare_in`(WP)*). One possible dead end situation*

[3] Uppercase letters denote variables and lowercase letters denote objects and types.

is $\chi = (s, a)$ *where:*

$$s = \bigwedge \big(\texttt{GOAL\_LOCATION}(la1a5), \texttt{ROAD}(la1a1, la1a2), \ldots, \neg\texttt{goal\_reward\_received}, \neg\texttt{hasspare}, \neg\texttt{spare\_in}(la1a1), \neg\texttt{spare\_in}(la1a2), \ldots, \texttt{vehicle\_at}(la1a1), \neg\texttt{vehicle\_at}(la1a2), \ldots, \texttt{not\_flattire}\big),$$

and $a = \texttt{move\_vehicle}(la1a1, la1a2)$*. This describes the situation where the vehicle moves to* $la1a2$*, has a flat tire, and has no means of replacing it.*

It is desired to have a generalised and compact representation such that $|\boldsymbol{\chi}|$ is reduced and an observed dead end situation can be generalised to unobserved dead end situations. Recording a state-action pair (s, a) as χ could be an over-specialised description of a dead end situation as some literals in s might be inconsequential. This is especially so for problems with factored transition functions where the transition of a state predicate depends on a subset of the state. In Example 1, the ground literals of $\texttt{spare\_in}(WP)$ for locations other than the vehicle's current location ($la1a1$) or the location it is moving to ($la1a2$) are inconsequential.

Suppose that there are two dead end situations in $\boldsymbol{\chi}$: $\chi_i = (s_i, a)$ and $\chi_j = (s_j, a)$ where $s_i = p_1 \wedge \ldots p_{m-1} \wedge p_m$ and $s_j = p_1 \wedge \ldots p_{m-1} \wedge \neg p_m$. The difference between s_i and s_j is the value of p_m. s_i and s_j can be abstracted to the same abstract state, $\bar{s} = p_1 \wedge \ldots p_{m-1}$, by eliminating p_m. χ_i and χ_j are replaced with $\chi_k = (\bar{s}, a)$ in $\boldsymbol{\chi}$. The condition for a state-action pair (s_t, a_t) to be in $\boldsymbol{\chi}$ is now as follows:

$$\exists(\bar{s}, a) \in \boldsymbol{\chi}(a = a_t \wedge \bar{s} \subseteq s_t). \tag{6}$$

Instead of an exact match for states, an abstract state matches a state if the abstract state is a subset of the state. χ_k covers or subsumes χ_i and χ_j since $\bar{s} \subset s_i$ and $\bar{s} \subset s_j$. That is, χ_k is a generalisation of χ_i and χ_j.

Definition 6 (Coverage and Subsumption)**.** *A dead end situation* $\chi = (\bar{s}, \bar{a})$ *covers any state-action pair* (s_t, a_t) *if* $\bar{a} = a_t \wedge \bar{s} \subseteq s_t$ *is true (see Equation 6). The set of state-action pairs covered by* χ *is denoted by* $coverage(\chi)$*. Due to the transitive relation of* $\subseteq$*, a dead end situation* χ_i *subsumes another dead end situation* χ_j *if* $coverage(\chi_j) \subset coverage(\chi_i)$ *is true; then* χ_i *is said to be more general than* χ_j*. In other words, the generality of a dead end situation is measured by the cardinality of its coverage.*

Subsumption is only applicable when two dead end situations differ by one literal. It can be done recursively (e.g., χ_k can be subsumed by another dead end situation). Dead end situations which are subsumed by a (more general) dead end situation are removed from $\boldsymbol{\chi}$. A literal which is eliminated due to subsumption is inconsequential in representing a dead end situation. Another possibility for subsumption is when a dead end situation $\chi = (s, a)$ is subsumed by χ_k. We refer to the aforementioned naive method with subsumption as LDE (Learning from Dead Ends). It is domain-independent and does not require any background knowledge. Subsumption reduces the cardinality of $\boldsymbol{\chi}$ which in turn reduces the space complexity. Since subsumption is only possible between dead end situations involving the same action, the computational complexity of checking for possible subsumption is $O\big(\binom{|\boldsymbol{\chi}|}{2}\big)$. While this can be costly, subsumption is only attempted when a dead end situation is added to $\boldsymbol{\chi}$ and only between dead end situations involving the same action. In practice, for every ten dead end situations which involves an action is added to $\boldsymbol{\chi}$, we check for subsumption of its dead end situations.

Conceptually, dead end situations are similar to *no-goods*—conjunctions of literals that all states in which such a conjunction holds are dead ends [Kolobov *et al.*, 2010]. Our work differs from [Kolobov *et al.*, 2010] in how this knowledge is learned and utilised. Furthermore, in the next section, we discuss first-order abstractions of dead end situations which generalise over all problems in a domain.

4.1 First-Order Generalisation

Subsumption produces a more general dead end situation only when all of its subsumed dead end situations are observed. Thus, LDE does not generalise to unobserved state-action pairs. To address this, we map a state-action pair (s, a) to a first-order representation $(\bar{s}, \bar{a})$ where $\bar{a}$ is the symbolic action of a and $\bar{s}$ is a conjunction of lifted literals given by:

$$\bar{s} = \bigwedge_{p \in \boldsymbol{P}_s} \mathscr{L}(p, a), \tag{7}$$

where s is a conjunction of literals $\boldsymbol{P}_s$ and $\mathscr{L}$ is a function which lifts a literal p by substituting objects in its terms with bound variables if they are also terms of a; otherwise, they are substituted with free variables. Variables are typed and have the same type as the objects they substituted.

Example 2 (First-Order Dead End Situation)**.** *Following Example 1, the first-order dead end situation* $\chi = (\bar{s}, \bar{a})$ *where:*

$$\bar{s} = \bigwedge \big(\texttt{GOAL\_LOCATION}(\star WP), \texttt{ROAD}(WP_1, WP_2), \texttt{ROAD}(WP_1, \star WP), \texttt{ROAD}(\star WP, WP_2), \texttt{ROAD}(\star WP, \star WP), \texttt{ROAD}(WP_2, \star WP), \neg\texttt{goal\_reward\_received}, \neg\texttt{hasspare}, \neg\texttt{spare\_in}(WP_1), \neg\texttt{spare\_in}(\star WP), \texttt{spare\_in}(\star WP), \neg\texttt{spare\_in}(WP_2), \texttt{vehicle\_at}(WP_1), \neg\texttt{vehicle\_at}(\star WP), \neg\texttt{vehicle\_at}(WP_2), \texttt{not\_flattire}\big),$$

and $\bar{a} = \texttt{move\_vehicle}(WP_1, WP_2)$*.*[4] *The coverage of* χ *is measured by the number of states which map to abstract states that are subsets of* $\bar{s}$*.* χ *covers state-action pairs where the vehicle has no spare, its current location* WP_1 *has no spare (which is inconsequential), and it moves to a location* WP_2 *that has no spare. This demonstrates the generality of first-order dead end situations.*

If (s, a) leads to a dead end, then the first-order dead end situation $\chi = (\bar{s}, \bar{a})$ is added to $\boldsymbol{\chi}$. LDE-FO denotes this extension to LDE. A state-action pair (s_t, a_t) is in $\boldsymbol{\chi}$ if its

[4] $\star\mathcal{C}$ denotes a free variable where $\mathcal{C}$ is the variable type.

Algorithm 1: Find dead end traps

```
1  Function LDE_DT(Γ, A, χ):
       Input: Trajectory Γ,
              Set of actions A,
              Failure buffer χ
2      while Γ is not empty do
3          (s, a, r, s') ← Pop from back of Γ
4          A_app ← Get_Applicable_Actions(s, A, χ)
5          if |A_app| > 1 then
6              Add (s, a) to χ
7              break
8      return χ
```

first-order representation $(\bar{s}_t, \bar{a}_t)$ is in $\boldsymbol{\chi}$. Equation 6 is applied here where actions are replaced with symbolic actions. Although LDE-FO is computationally more expensive than LDE due to the additional step of converting a state-action pair to a first-order representation, LDE-FO generalises to unobserved dead end situations and over different problems regardless of its number of objects. First-order dead end situations can be transferred to any problem of the same domain such that dead ends can be avoided right away.

LDE-FO assumes that (1) the problem is relational and (2) **deictic objects** or objects which are not in the terms of an action can be treated as a homogeneous typed entity (i.e., deictic objects of the same type are mapped to the same free variable). This is analogous to existential quantification in RDDL which treats deictic objects as homogeneous entities. Our assumptions are valid for problems written in RDDL if each conditional probability function has at most one deictic object of each type. If (2) is violated, the deictic objects can be included as terms in the action.

4.2 Dead End Traps

There might be states where a dead end could be reached eventually regardless of which action is executed. These states are referred to as dead end traps.

Definition 7 (Dead End Trap)**.** *A dead end trap is a state where there is a non-zero probability of reaching a dead end trap or a dead end eventually regardless of which action is executed.*

Definition 7 includes actions which do not change the state (e.g., inapplicable actions) since they lead back to the same state which is a dead end trap. Our dead end traps are semantically similar to traps in traditional planning. For example, [Lipovetzky *et al.*, 2016] defines traps as conditional invariant formulas: if the formula is satisfied in a state s, it is satisfied in all states reachable from s.

LDE and LDE-FO avoid dead ends by influencing the policy to avoid actions which could lead to dead ends. However, this does not prevent dead ends in the presence of dead end traps. Suppose that a state s is a dead end trap. Given sufficient observations, the state-action pairs $(s, a), \forall a \in \boldsymbol{A}$ will be added to $\boldsymbol{\chi}$ and no actions remain for selection. We extend LDE and LDE-FO to avoid dead end traps, denoted by LDE-DT and LDE-DT-FO, respectively. LDE-DT is outlined in Algorithm 1 which looks back at a previous time step to determine the situation which led to a dead end trap. Algorithm 1 is used only when a dead end is reached. The inputs are a trajectory Γ which is a sequence of observations (s, a, r, s') from the initial state to the dead end, the set of actions $\boldsymbol{A}$, and the failure buffer $\boldsymbol{\chi}$. The observations in Γ are checked sequentially, starting from the most recent one (lines 2 to 7). For each observation (s, a, r, s'), the set of applicable actions in s is determined (line 4). Actions with preconditions that are satisfied in s and which form a state-action pair with s that is not in $\boldsymbol{\chi}$ are deemed to be applicable. If there is more than one applicable action (line 5), s might not be a dead end trap, and thus LDE-DT stops looking backwards and adds (s, a) (or $(\bar{s}, \hat{a})$ for LDE-DT-FO) to $\boldsymbol{\chi}$ (line 6). Otherwise, s is a dead end trap and the preceding observation is considered next. This continues until the condition in line 5 is satisfied; situations which lead to dead end traps are added to $\boldsymbol{\chi}$. If the condition is not satisfied, then the problem has an unavoidable dead end and $\boldsymbol{\chi}$ will not be updated.

Example 3 (Looking Back in the Face of Dead End Traps)**.** *We consider two particular episodes for a problem instance of* TT *which are illustrated in Figure 1. A dead end is reached in both episodes. In the first episode, illustrated in Figure 1 (top), the vehicle moves from* $la1a1$ *to* $la1a2$ *at time step* $t = 0$*. At* $t = 1$*, the vehicle moves to* $la1a3$*. It has a flat tire at* $t = 2$*. Since the vehicle did not have a spare and there are no spares at* $la1a3$*, a dead end is reached at* $t = 2$*. The state-action pair (*$s_{1,1}$, move_vehicle$(la1a2, la1a3)$*) is added to* $\boldsymbol{\chi}$ *where* $s_{1,1}$ *denotes the state at episode* 1*, time step* 1*.*

In the next episode, which is illustrated in Figure 1 (middle and bottom), the vehicle moves from $la1a1$ *to* $la1a2$ *at* $t = 0$*. At* $t = 1$*, the policy considers the dead end situation encountered in the first episode and will not select* move_vehicle$(la1a2, la1a3)$*. The only available action is to move to* $la1a2$*. At* $t = 1$*, the vehicle moves to* $la1a2$*. It then has a flat tire at* $t = 2$*. A spare is loaded at* $t = 2$ *and the tire is changed at* $t = 3$*. At* $t = 4$*, the vehicle moves to* $la1a3$*. It has a flat tire at* $t = 5$ *and a dead end (*$s_{2,5}$*) is reached. The state-action pair (*$s_{2,4}$, move_vehicle$(la2a2, la1a3)$*) can be added to* $\boldsymbol{\chi}$ *but since there is only one applicable action,* $s_{2,4}$ *is a dead end trap.*

To avoid $s_{2,4}$*, an alternative action has to be selected at* $t = 3$*. At* $t = 3$*, the vehicle has a flat tire and the only applicable action is* changetire*. Thus,* $s_{2,3}$ *is also a dead end trap. We continue to look backwards at the previous time step. Similarly,* $s_{2,2}$ *(only* loadtire$(la2a2)$ *is applicable) and* $s_{2,1}$ *(only* move_vehicle$(la1a2, la2a2)$ *is applicable because* move_vehicle$(la1a2, la1a3)$ *is ruled out by* $\boldsymbol{\chi}$*) are dead end traps. Lastly, at* $t = 0$*, there are two applicable actions. The dead end trap* $s_{2,1}$ *can be avoided by executing the other action,* move_vehicle$(la1a1, la2a1)$*. The state-action pair (*$s_{2,1}$, move_vehicle$(la1a1, la1a2)$*) is added to* $\boldsymbol{\chi}$*. In contrast, LDE will add (*$s_{2,4}$, move_vehicle$(la2a2, la1a3)$*) to* $\boldsymbol{\chi}$*. This will not prevent a dead end since there are no other applicable actions in* $s_{2,4}$ *and* move_vehicle$(la2a2, la1a3)$ *will be executed.*

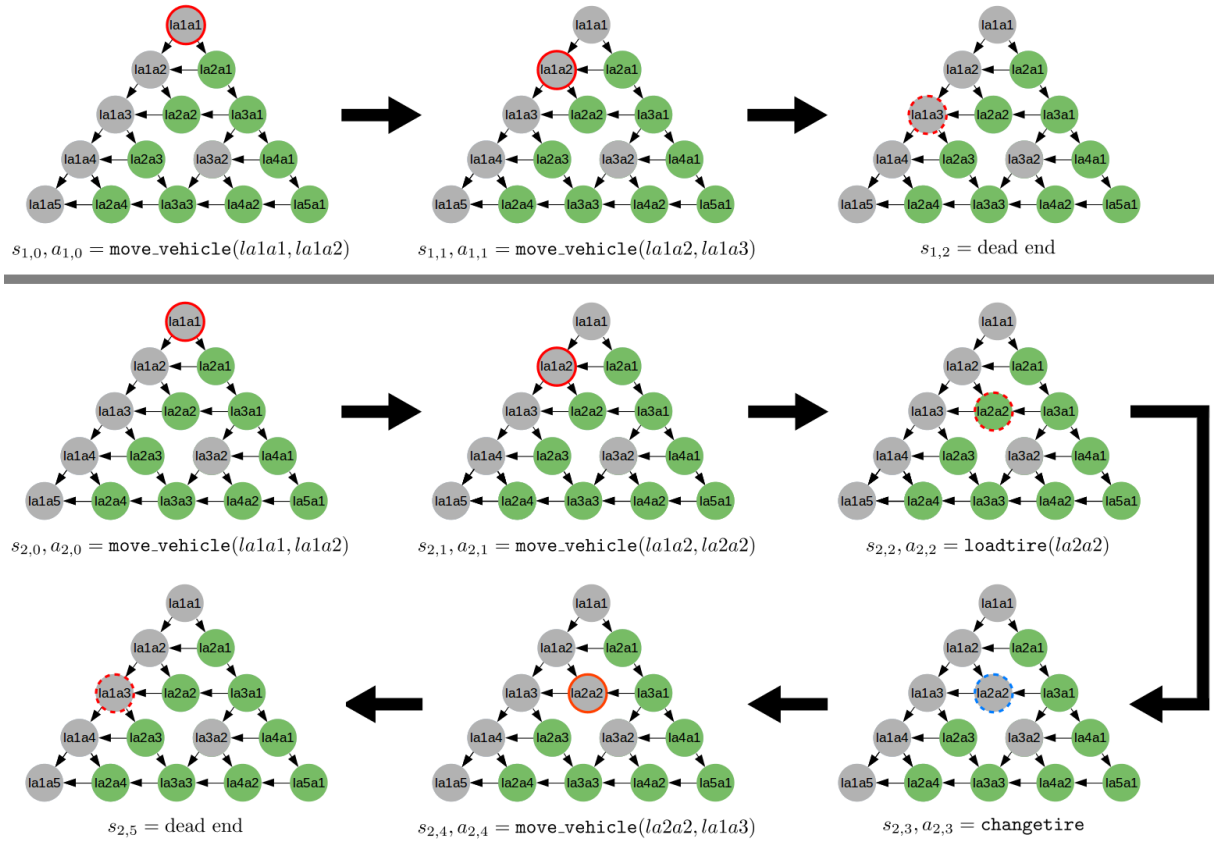

Figure 1: Trajectories in episodes 1 (top row) and 2 (middle row and bottom row) of a particular problem instance of TT. $s_{i,j}$ and $a_{i,j}$ denote the state and action executed, respectively, at episode i, time step j. Each subfigure illustrates a state that an action is executed in. The next state is illustrated in the next subfigure. Locations are represented by circles. A green circle indicates that the location has a spare. A circle with a border indicates that the vehicle is at that location. A blue border indicates that the vehicle has a spare while a red border indicates otherwise. A dotted border indicates that the vehicle has a flat tire while a solid border indicates otherwise.

4.3 Soundness

LDE and its variants are sound if they do not erroneously determine that executing an action could lead to a dead end. For LDE-DT and LDE-DT-FO, this extends to dead end traps.

Theorem 1 (Soundness of LDE and its Variants)**.** *LDE, LDE-DT, LDE-FO, and LDE-DT-FO are sound.*

Proof (sketch). It is straightforward to see that LDE is sound as it only adds observed dead end situations to $\boldsymbol{\chi}$. Subsumption finds a more general dead end situation among observed ones and does not generalise to unobserved state-action pairs. Therefore, a state-action pair (s, a) is in $\boldsymbol{\chi}$ if and only if there is an observation (s_t, a_t, r_t, s_{t+1}) where $s = s_t$, $a = a_t$, and s_{t+1} is a dead end. The same reasoning applies for dead end traps; thus, LDE-DT is also sound.

Next, we prove that LDE-FO is sound. It uses a first-order abstraction (Equation 7) to generalise to unobserved state-action pairs. LDE-FO is sound if this generalisation does not cause it to erroneously determine state-action pairs lead to dead ends. Let (s_t, a_t, r_t, s_{t+1}) be an observation where s_{t+1} is a dead end and $(\bar{s}_t, \bar{a}_t, r_t, \bar{s}_{t+1})$ be its first-order representation. Any state-action pair which maps to $(\bar{s}_t, \bar{a}_t)$ must lead to a next state, with non-zero probability, which maps to $\bar{s}_{t+1}$. Formally, $\forall s \in \boldsymbol{S}, a \in \boldsymbol{A}$, if $s \mapsto \bar{s}_t$ and $a \mapsto \bar{a}_t$, then $\exists s' \mathscr{T}(s'|s, a) > 0$ such that $s' \mapsto \bar{s}_{t+1}$. This is true if $\mathscr{T}(s_{t+1}|s_t, a_t) = \mathcal{T}(\bar{s}_{t+1}|\bar{s}_t, \bar{a}_t)$.

In relational problems, transition functions are parameterised and factored:

$$\begin{aligned}\mathscr{T}(s_{t+1}|s_t, a_t) &= \prod_i \mathscr{T}(p'_i|\boldsymbol{P}^-, a_t) \\ &= \prod_i \mathcal{T}\big(\bar{p}'_i \big| \bigcup_{p \in \boldsymbol{P}^-} \mathscr{L}(p, a_t), \bar{a}_t\big) \\ &= \prod_i \mathcal{T}(\bar{p}'_i|\boldsymbol{\mathcal{P}}^-, \bar{a}_t) \\ &= \mathcal{T}(\bar{s}_{t+1}|\bar{s}_t, \bar{a}_t),\end{aligned}$$

where p'_i is the value of the state predicate $p_i \in \boldsymbol{P}$ in s_{t+1}, $\bar{p}_i$ is the symbolic state predicate of p_i, and $\boldsymbol{\mathcal{P}}^-$ results from the lifting of $\boldsymbol{P}^-$ using $\mathscr{L}$. $\mathscr{L}$ substitutes objects in p which are not terms of a_t with free variables. The validity of the abstraction due to $\mathscr{L}$ requires our assumption that deictic objects of the same type can be treated as a homogeneous entity, which completes the proof for LDE-FO. The same reasoning applies for dead end traps; thus, LDE-DT-FO is also sound.

4.4 Online Reinforcement Learning

LDE (including LDE and its variants unless stated otherwise) is used in an online RL algorithm where a behaviour policy π considers the failure buffer $\boldsymbol{\chi}$ to select an action in every time step. π is generated from the Q-function approximation and avoids any action which forms a state-action pair that is in $\boldsymbol{\chi}$. We use the relational RL algorithm from our previous work [Ng and Petrick, 2021b] for this purpose. A relational RL learns in an abstract state-action space which achieves generalisation to unseen state-action pairs and to new problems. This is done by approximating the Q-function with a linear function approximation using first-order features. A first-order feature is a lifted literal or a conjunction of lifted literals which contains bound and/or free variables. The Q-function approximation is updated in every time step with Double Q-learning [Hasselt *et al.*, 2016] and replacing eligibility traces [Singh *et al.*, 1995]. We refer readers to [Ng and Petrick, 2021b] for further details.

Observations are recorded in Γ as $\{(s_0, a_0, r_0, s_1), \ldots, (s_{H-1}, a_{H-1}, r_{H-1}, s_H)\}$ for an episodic trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots, s_{H-1}, a_{H-1}, r_{H-1}, s_H)$. If the episode terminates with a dead end, the state-action pair (s_{t-1}, a_{t-1}) which leads to the dead end s_t is added to $\boldsymbol{\chi}$ (e.g., with Algorithm 1). For first-order dead end situations, $\boldsymbol{\chi}$ can be transferred to another problem of the same domain regardless of differences in their sets of objects ($\boldsymbol{O}$), actions ($\boldsymbol{A}$), or state predicates ($\boldsymbol{P}$). In this target problem, the policy can utilise $\boldsymbol{\chi}$ to avoid dead ends which are relationally identical to those encountered in previous problems.

Dead end situations and the Q-function approximation are learned in parallel but represented individually. This decoupled approach has two advantages. First, LDE can be used with any online RL algorithm such as value-based or policy-based methods. Second, $\boldsymbol{\chi}$ can be transferred to problems with different reward functions as long as the transition function is the same and the notion of dead ends remain unchanged (e.g., the agent is not rewarded for reaching dead ends in the target problem). Transferring the policy or Q-function, if the online RL algorithm permits (e.g., [Ng and Petrick, 2021b]), could deteriorate performance in this case.

5 Experiments

We evaluate the performance of our online relational RL algorithm from [Ng and Petrick, 2021b] combined with LDE or its variants. Experiments were conducted on an Intel Xeon E5-2660 v3 2.60 GHz with 8 cores and 32 GB of RAM. Results are averaged over 10 independent runs. The Q-function approximation and failure buffer are updated across episodes while no information is exchanged between runs. We used ϵ-$greedy$ policy with an exponentially decaying ϵ over episodes. The parameters used are $\epsilon = 1, \alpha = 0.3$, $\gamma = 0.9$, and $\lambda = 0.7$. α decays linearly over episodes to 0.05. In all of our results, the shading in the figures represents one standard deviation.

Benchmark Domains. We ran experiments on two domains: TT (see Example 1) and Robot Inspection (RI) [Ng and Petrick, 2021a]. TT is used in IPPC 2014 [Grzes *et al.*, 2014]. It has problems numbered from 1 to 10, where a larger number represents a problem with a larger state space. We used TT3 and TT6 where the size of the state-action spaces are $2^{33} \times 242$ and $2^{59} \times 814$, respectively.

In RI, a mobile robot needs to find objects by surveying a location where the objects are at before it can inspect them. The goals are to transmit information on inspected objects at the communication tower. An immediate reward of 20 is given for achieving each goal. The robot can move between any two locations directly and there is a probability of 0.08 that it is low on energy after moving. It needs to return to the docking station immediately to recharge, otherwise it is stranded and a dead end is reached. Its camera can also lose calibration during inspection with a probability of 0.15 which reduces the success rate of surveying to 0.2 and inspection to 0.9 (the state is unaffected if these actions fail). The robot can calibrate its camera at the docking station which restores the success rate back to 1. The small and large scale problems are denoted by RI1 and RI2 and the size of their state-action spaces are $2^{23} \times 27$ and $2^{29} \times 36$, respectively. The time horizon is 40 for RI1 and RI2. The set of objects $\boldsymbol{\mathcal{O}}$ in RI1 includes five locations and three objects while the set of objects $\boldsymbol{\mathcal{O}}$ in RI2 includes ten locations and six objects. The problems are randomised by randomising the locations of objects, the docking station, and the communication tower $\big($COMM_TOWER_AT$(OBJ, WP)\big)$.

Figure 2 shows the results for the large scale problems of both domains. Since there is no penalty for reaching a dead end in RI, an immediate reward of -1 is given for the remaining time steps if a dead end is reached.[5] This is only for the analysis of the results. For LDE-DT-FO, we consider the case where first-order dead end situations are learned in the small scale problems, TT3 and RI1, and transferred to the large scale problems, TT6 and RI2. This is denoted by LDE-DT-FO (transfer) in the figure. Dead end situations can still be added if dead ends are encountered in the large scale problems. The Q-function approximation is not transferred and is learned from scratch.

The performance is measured by the total rewards received and the number of dead ends reached. In both problems, the performance improved when learning from dead ends. In TT6, LDE-DT and LDE-FO outperform LDE. The former is due to the prevalence of dead end traps in TT and the latter is due to the generalisation property of LDE-FO. The combination of the two, LDE-DT-FO, gives the best performance, requiring only a few episodes to learn the optimal policy. When first-order dead end situations are transferred, the optimal policy is obtained almost immediately.

RI2 has no dead end traps. As such, learning from dead end traps does not improve the performance. Similar to TT6, learning first-order dead end situations improves performance. Although the transfer of dead end situations does not

[5] Suppose that this is not done. An episode which terminates in a dead end at $t = 1$ will have a total reward of -1 while another episode which achieves a goal and does not terminate in a dead end will have a total reward of $r - H = -20$ where $r = 20$ is the reward for achieving the goal, $H = 40$, and the action cost is -1. Clearly, the latter episode is better than the former but the total rewards suggest otherwise.

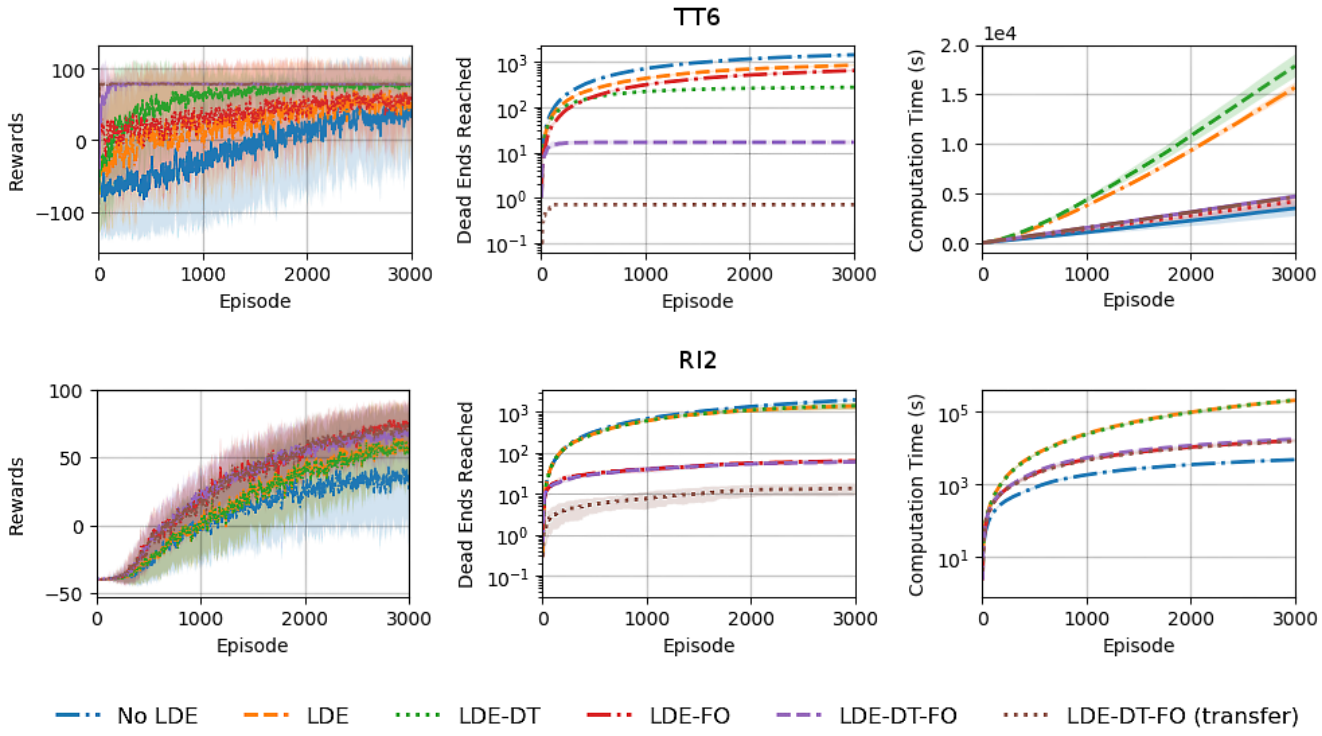

Figure 2: Performance of LDE and its variants for the problems TT6 (top) and RI2 (bottom). For LDE−DT−FO (transfer), first-order dead end situations which are learned in the small scale problems, TT3 and RI1, are transferred to the large scale problems, TT6 and RI2.

lead to an increase in rewards, it reduced the number of dead ends reached. Unlike TT which gives an immediate reward of −100 for reaching a dead end, RI does not give a penalty for reaching a dead end (other than terminating the episode which prevents further accumulation of rewards). Thus, the difference in rewards between the variants of LDE is of a lesser magnitude as compared to TT. We attempted to assign a negative reward for reaching dead ends in RI. Empirical results (not shown) indicate that this does not always produce the intended effect. During exploration, if the agent reaches a dead end more often than achieving a goal (i.e., transmit information about an object), then RL learns a policy which avoids any move actions since the agent can only be low on energy after moving to another location. This results in poor performance where the agent remains in the same location where possible, for example, by docking and undocking repeatedly.

The computational time for LDE and its variants are dependent on the cardinality of the failure buffer χ. With a first-order representation, the cardinality of χ is reduced which decreases the computational time. The computational time of LDE−DT is higher than LDE due to the additional cost of checking for dead end traps. In RI2, the computational time for LDE and LDE−DT is prohibitively long. This is because there are many dead end situations in RI; a dead end is reached if the robot is low on energy and does not move to the base immediately. This increases the cardinality of χ and, subsequently, the time complexity of searching through χ for a matching state-action pair. This highlights the scalability of first-order dead end situations.

6 Conclusion and Future Work

We introduced a novel family of algorithms to learn dead end situations or state-action pairs which were previously observed to lead to dead ends. LDE learns from dead ends and influences the policy to avoid previously encountered dead ends. The efficacy of LDE is improved by generalising observations with a first-order representation and detecting dead end traps. LDE−FO allows transfer learning while LDE−DT learns from dead end traps. Empirical results showed that learning from dead ends is effective for avoiding dead ends and significantly improves the performance of an online RL algorithm in relational problems. Transfer learning also accelerates learning by directly transferring first-order dead end situations over problems of different scales. In future work, we plan to investigate the use of learned transition functions for multi-step lookahead to avoid dead ends more effectively.

Acknowledgements

This work was partially funded by the EPSRC ORCA Hub (http://orcahub.org/) under grant number EP/R026173/1.

References

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

[Camacho *et al.*, 2016] Alberto Camacho, Christian Muise, and Sheila A. McIlraith. From FOND to robust prob-

abilistic planning: Computing compact policies that bypass avoidable deadends. In *Proceedings of the International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, pages 65–69, 2016.

[Cserna *et al.*, 2018] Bence Cserna, William J. Doyle, Jordan S. Ramsdell, and Wheeler Ruml. Avoiding dead ends in real-time heuristic search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[Fatemi *et al.*, 2019] Mehdi Fatemi, Shikhar Sharma, Harm van Seijen, and Samira Ebrahimi Kahou. Dead-ends and secure exploration in reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[García and Fernández, 2015] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[Grzes *et al.*, 2014] Marek Grzes, Jesse Hoey, and Scott Sanner. International Probabilistic Planning Competition (IPPC) 2014. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2014.

[Hasselt *et al.*, 2016] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with Double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2094–2100, 2016.

[Kolobov *et al.*, 2010] Andrey Kolobov, Mausam, and Daniel S. Weld. SixthSense: Fast and reliable recognition of dead ends in MDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1108–1114, 2010.

[Lipovetzky *et al.*, 2016] Nir Lipovetzky, Christian Muise, and Hector Geffner. Traps, invariants, and dead-ends. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 211–215, 2016.

[Little and Thiebaux, 2007] Iain Little and Sylvie Thiebaux. Probabilistic planning vs. replanning. In *Proceedings of the ICAPS Workshop on the International Planning Competition: Past, Present and Future*, 2007.

[Mausam and Weld, 2003] Mausam and Daniel S Weld. Solving relational MDPs with first-order machine learning. In *Proceedings of the ICAPS Workshop on Planning Under Uncertainty And Incomplete Information*, 2003.

[Moldovan and Abbeel, 2012] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1451–1458, 2012.

[Ng and Petrick, 2021a] Jun Hao Alvin Ng and Ronald P. A. Petrick. Generalised linear function approximation with first-order features. In *Proceedings of the IJCAI Workshop on Generalization in Planning (GenPlan)*, 2021.

[Ng and Petrick, 2021b] Jun Hao Alvin Ng and Ronald P. A. Petrick. Generalised task planning with first-order function approximation. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2021.

[Sanner, 2011a] Scott Sanner. ICAPS 2011 International Probabilistic Planning Competition (IPPC). http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/, 2011. Accessed: 18.12.2020.

[Sanner, 2011b] Scott Sanner. Relational dynamic influence diagram language (RDDL): Language description. http://users.cecs.anu.edu.au/ ssanner/IPPC_2011/RDDL.pdf, 2011.

[Singh *et al.*, 1995] Satinder Singh, Richard Sutton, and P. Kaelbling. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22, 11 1995.

[Steinmetz and Hoffmann, 2017] Marcel Steinmetz and Jörg Hoffmann. Search and learn: On dead-end detectors, the traps they set, and trap learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4398–4404, 2017.

[Ståhlberg *et al.*, 2021] Simon Ståhlberg, Guillem Francès, and Jendrik Seipp. Learning generalized unsolvability heuristics for classical planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 4175–4181, 2021.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.