Reinforcement Learning for Stochastic Shortest Path with Dead-Ends

Gustavo De Mari Pereira

Thesis presented to the
Institute of Mathematics and Statistics
of the University of São Paulo
in partial fulfillment
of the requirements
for the degree of
Master of Science

Program:	Computer Science
Advisor:	Prof.ª Dr.ª Leliane Nunes de Barros

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001

São Paulo
July, 2025

Reinforcement Learning for Stochastic Shortest Path with Dead-Ends

Gustavo De Mari Pereira

Ficha catalográfica elaborada com dados inseridos pelo(a) autor(a)
Biblioteca Carlos Benjamin de Lyra
Instituto de Matemática e Estatística
Universidade de São Paulo

```
Pereira, Gustavo De Mari
   Reinforcement Learning for Stochastic Shortest Path with
Dead-Ends / Gustavo De Mari Pereira; orientadora, Leliane
Nunes de Barros. - São Paulo, 2025.
   121 p.: il.


   Dissertação (Mestrado) - Programa de Pós-Graduação
em Ciência da Computação / Instituto de Matemática e
Estatística / Universidade de São Paulo.
   Bibliografia
   Versão corrigida


   1. INTELIGÊNCIA ARTIFICIAL. 2. APRENDIZADO POR REFORÇO. 3.
PLANEJAMENTO PROBABILÍSTICO. 4. CAMINHO MAIS CURTO ESTOCÁSTICO.
I. Nunes de Barros, Leliane. II. Título.
```

Bibliotecárias do Serviço de Informação e Biblioteca
Carlos Benjamin de Lyra do IME-USP, responsáveis pela
estrutura de catalogação da publicação de acordo com a AACR2:
Maria Lúcia Ribeiro CRB-8/2766; Stela do Nascimento Madruga CRB 8/7534.

To my parents, Samuel and Rita.
To my uncle, Julio Angelo De Mari (in memoriam).
To my wife, Bruna.

Acknowledgments

To my parents, Samuel and Rita, for their love and tireless effort, for the sacrifices they made by setting aside their own needs to prioritize me, and for their unconditional support and encouragement throughout my life and education.

To my wife, Bruna, for encouraging me to pursue great achievements, for her loving companionship, patience, support through every moment, and for her inspiring example of determination.

To my advisor, Prof. Leliane, for her valuable guidance, for giving me the opportunity to be part of USP, and for the numerous discussions, meetings, and contributions that made this work possible.

To my uncle, Julio Angelo De Mari (in memoriam), for his support of my education and for the authenticity, playful humor, and joyful spirit that continue to inspire me.

To the professors who contributed to my education throughout my academic journey.

To the friends and colleagues from IME-USP and LIAMF, whose intellect, dedication, and enthusiasm were a source of inspiration.

To my family, my sister Amanda and friends who always cheered me on and motivated me to keep moving forward.

Resumo

Gustavo De Mari Pereira. **Aprendizado por Reforço para Caminho de Custo Mínimo Estocástico com Becos Sem Saída**. Dissertação (Mestrado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2025.

Problemas em Aprendizado por Reforço (AR) são frequentemente modelados usando Processos de Decisão de Markov de Horizonte Infinito Descontado (do inglês, *Infinite Horizon Discounted Markov Decision Processes* - IHD-MDPs). No entanto, a comunidade de Planejamento Probabilístico argumenta que problemas de Caminho Mais Curto Estocástico (do inglês, *Stochastic Shortest Path* - SSP) oferecem uma estrutura mais natural para tarefas orientadas a meta. Em particular, ao lidar com SSPs envolvendo becos-sem-saída (estados a partir dos quais o agente não pode mais alcançar a meta), MDPs descontados requerem o ajuste cuidadoso do fator de desconto e nem sempre é possível encontrar soluções de interesse. Com base nos resultados de pesquisas em Planejamento Probabilístico, neste trabalho, propomos modificações em algoritmos clássicos de AR, como o algoritmo Q-learning, para resolver três classes de SSPs: (i) SSPs sem becos-sem-saída, (ii) SSPs com becos-sem-saída evitáveis e (iii) SSPs com becos-sem-saída inevitáveis. Com essas modificações, é possível aplicar algoritmos de AR em uma ampla gama de problemas de tomada de decisão sequencial orientados a meta.

Palavras-chave: Aprendizado por Reforço. Caminho Mais Curto Estocástico. Becos sem Saída. Meta. Planejamento Probabilístico. Aprendizado de Máquina. Tomada de Decisão Sequencial.

Abstract

Gustavo De Mari Pereira. **Reinforcement Learning for Stochastic Shortest Path with Dead-Ends**. Thesis (Master's). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2025.

Problems in reinforcement learning (RL) are often modeled using infinite horizon discounted Markov decision processes (IHD-MDPs). However, the probabilistic planning community argues that stochastic shortest path (SSP) problems offer a more natural framework for goal-oriented tasks. In particular, when dealing with SSPs involving dead-ends (i.e. states from which the agent can no longer achieve the goal), discounted MDPs require careful discount factor tuning and in some cases, it is not possible to find a desired solution. Leveraged by the results of the probabilistic planning research, in this work, we propose modifications to classical RL algorithms, like Q-learning, to solve three classes of SSPs: (i) SSPs without dead-end states, (ii) SSPs with avoidable dead-end states, and (iii) SSPs with unavoidable dead-end states. With these modifications it is possible to apply RL algorithms to a broad range of goal-oriented problems.

Keywords: Reinforcement Learning. Stochastic Shortest Path. Dead-ends. Goals. Probabilistic Planning. Machine Learning. Sequential Decision Making.

List of Abbreviations

AI	Artificial Intelligence
DL	Deep Learning
DP	Dynamic Programming
ADP	Approximate Dynamic Programming
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
IHD-MDP	Infinite Horizon Discounted Markov Decision Process
SSP	Stochastic Shortest Path
SCC	Strongly Connected Components
VI	Value Iteration
SVI	Structured Value Iteration
TVI	Topological Value Iteration
PI	Policy Iteration
RL	Reinforcement Learning
IM	Intrinsic Motivation
Q-learning	Q-value Learning Algorithm
DQN	Deep Q-Network Algorithm
SARSA	State-Action-Reward-State-Action
RTDP	Real Time Dynamic Programming
LRTDP	Labeled Real Time Dynamic Programming
sRTDP	Symbolic Real Time Dynamic Programming
UCT	Upper Confidence bounds applied to Trees
PROST	Probabilistic Planning Based on UCT
ADD	Algebraic Decision Diagram
BDD	Binary Decision Diagram
A*	A-star search algorithm
AO*	And/Or star search algorithm
LAO*	Loop And/Or star search algorithm
SPUDD	Stochastic Planning using Decision Diagrams
SSPDE	Stochastic Shortest Path MDP with Dead Ends
BET	Bellman Backup with Escaping Traps

CMDP	Constrained Markov Decision Process
F&R	Find-and-Revise
FRET	Find, Revise, Eliminate Traps
GCRL	Goal-Conditioned Reinforcement Learning
GLIE	Greedy in the Limit with Infinite Exploration
GORL	Goal-Oriented Reinforcement Learning
IPPC	International Probabilistic Planning Competition
LP	Linear Programming
MORL	Multi-Objective Reinforcement Learning
Safe RL	Safe Reinforcement Learning
SCC	Strongly Connected Components
VI-FP	Value Iteration with Finite Penalty
VI-MaxProb	Value Iteration for MaxProb
XADD	Extended Algebraic Decision Diagram
SSPADE	Stochastic Shortest Path MDP with Avoidable Dead Ends
SSPUDE	Stochastic Shortest Path with Unavoidable Dead Ends
fSSPUDE	Finite Penalty Stochastic Shortest Path with Unavoidable Dead Ends
iSSPUDE	Infinite Penalty Stochastic Shortest Path with Unavoidable Dead Ends
FP	Finite Penalty Criterion
S^3P	Stochastic and Safest Shortest Path
MCMP	Min-Cost given Max-Prob
PPDDL	Probabilistic Planning Domain Definition Language
GPCI	Goal-Probability and Cost-Iteration
GUBS	Goals with Utility-Based Semantic

List of Symbols

$\mathcal{S}$	State space
$\mathcal{A}$	Action space
s_0	Initial state
$\mathcal{T}(s, a, s')$	Transition probability function
$\mathcal{C}(s, a)$	Cost function
$\mathcal{R}(s, a)$	Reward function
$V(s)$	Value function
$V^*(s)$	Optimal Value function
$Q(s, a)$	Action-value function
$Q^*(s, a)$	Optimal Action-value function
$\pi(s)$	Deterministic policy
$\pi(a\|s)$	Stochastic policy
$\mathbb{E}[\cdot]$	Expectation operator
$\mathcal{H}$	Horizon
α	Learning rate
γ	Discount factor
t	Timestep
τ	Trajectory $(s_0, a_0, ..., s_H)$
T	Set of Traces
δ	Temporal difference error
$\mathcal{G}$	Set of Goals
s_g	Goal state
Π	Set of Policies
D	Penalty
$x_{s,a}$	State-action occupancy measure (or state-visitation frequency)

List of Figures

List of Tables

List of Algorithms

Contents

Chapter 1

Introduction

Sequential decision-making problems are extensively studied in the area of artificial intelligence. These problems involve an agent making a sequence of decisions over time while optimizing an utility function and/or the achievement of a desired goal, where these decisions may either have a deterministic or probabilistic outcomes. Such problems are commonly modeled using **Markov Decision Processes** (MDPs) (PUTERMAN, 2014) which provide a mathematical formalism to represent the environment dynamics, including states, actions, transition probability and reward (cost) functions. In an MDP, an **agent** observes the current state of the environment, chooses an action, and goes with a certain probability to a next state, receiving an immediate reward and/or cost. The agent's objective is to determine a sequence of actions that yields the best cumulative rewards (or costs) along the process. The general objective of an MDP agent is to compute a **policy** (a mapping from states to actions) that maximizes (minimizes) the expected total reward (cost) over time.

MDPs can be categorized into: finite, infinite or indefinite horizon. In **finite-horizon MDPs**, the agent has a fixed number of time steps, h, per episode, and the goal is to optimize the expected cumulative reward (or cost) over this horizon. In **infinite-horizon MDPs**, the agent may take actions indefinitely and, since the expected cumulative reward can diverge to infinity in this setting, a discount factor ($\gamma < 1$) is introduced to ensure the sum converges. We call this type of MDP as **Infinite-Horizon Discounted MDPs (IHD-MDP for short)**.

In **indefinite-horizon MDPs**, there are terminal states that can end an episode at any time-step. A special type of indefinite undiscounted horizon MDPs are the **Stochastic Shortest Path MDPs** (SSP MDPs or SSPs for short) (D. P. BERTSEKAS and J. N. TSITSIKLIS, 1991) which considers a cost function, instead of a reward function, and explicitly include a set of goal states to be reached by the agent. Goal states are seen as a special cost-free termination state; once the system reaches that state, it remains there at no further cost. (D. BERTSEKAS, 2023).

Most research on SSPs makes the assumption that there are no dead-end states in the environment, which are states from which it is not possible to reach a goal state (KOLOBOV and WELD, 2012; TEICHTEIL-KÖNIGSBUCH, 2012; TREVIZAN *et al.*, 2017). However, in practice, this is hardly the case. According to TREVIZAN *et al.* (2017): "*if dead-ends are* *unavoidable, i.e., the probability of reaching the goal is always less than 1, SSPs implicitly become a multi-objective optimization problem with two potentially conflicting objectives: the maximization of the probability to reach the goal and the minimization of the expected cost to reach the goal*".

A first intuitive solution for this problem allows to solve it only with a single optimization objective, known as **Finite-Penalty optimization criterion** (Mausam and Kolobov, 2012), where we assign a finite and fixed penalty D for not reaching the goal. Although intuitive, the Finite-Penalty criterion has the drawback of considering any state with expected cost greater than D as a dead-end. For some problems, it might be easy to derive a good value for D, however this is not an assumption that can be made in domain independent planning (i.e., for algorithms that do not use knowledge about the problem being solved) which is the assumption of this work. According with Trevizan *et al.* (2017), *there is no domain-independent method to automatically derive D given a problem description and there is no known method to learn a function that returns, for all SSPs, a value of D that is always large enough but never so large that it leads to numerical instability or increased CPU time.*

To avoid the need for defining the fixed penalty D we can first maximize only the probability to reach the goal ignoring the expected cost. This approach is known as MaxProb (Kolobov, M. Mausam, *et al.*, 2011); however it would be unable to distinguish between policies with the same probability of reaching the goal but with different costs. Therefore, one must be able to distinguish such cases with an extra optimization step: minimizing the cumulative expected cost among the MaxProb policies, which then becomes a multi-objective problem. Several important solutions from the automated probabilistic planning area have been proposed (Mausam and Kolobov, 2012; Ghallab *et al.*, 2004; D. P. Bertsekas and J. N. Tsitsiklis, 1991) to solve SSPs that encompass important assumptions one must make when solving such problems, e.g. goal reachability assumptions and the existence of dead-end states (Kolobov and Weld, 2012; Teichteil-Königsbuch, 2012; Trevizan *et al.*, 2017).

While SSPs have been extensively explored by the planning community, they have received relatively less attention in reinforcement learning (RL). Most RL research is still centered on the IHD-MDP framework as a universal model, despite the fact that the SSP framework generalizes IHD-MDPs (Mausam and Kolobov, 2012, p. 23). As SSPs introduce distinct features such as explicit goal states, the required solution concepts are not always compatible with the standard IHD-MDP framework. This is especially true for problems of reaching a goal state with minimal expected cost in the presence of unavoidable dead-end states, a context where the IHD-MDP framework has clear limitations.

In this work, we bridge the gap between probabilistic planning and RL by proposing novel methods to solve SSP problems, particularly those with dead-ends. We introduce several extensions to the Q-learning algorithm, each integrating a specific optimization criterion from planning. These extensions are designed to address critical challenges such as ensuring convergence in the presence of dead-ends and handling the multi-objective task of balancing maximal goal probability with minimal expected cost. To validate our approach, we conduct an empirical analysis that compares our proposed algorithms against both their exact planning counterparts and standard IHD-MDP solutions.

1.1 Is the Infinite-Horizon Discounted MDP a Universal Model?

Notice that in a Infinite-Horizon Discounted MDP the discount factor γ indicates what horizon would be worth to look for future cumulative reward. E.g. for $\gamma = 0.99$ the effective horizon is 100 time-steps (considering unitary costs). Based on this idea D. Bertsekas (2023, p.44) states the following:

> *There is a widespread belief that discounted MDP can be used as a universal model, i.e., that in practice any other kind of problem (e.g., undiscounted problems with a termination state and/or a continuous state space) can be painlessly converted to a discounted MDP with a discount factor that is close enough to 1. This is questionable, however, for a number of reasons:*
>
> [...]
>
> (c) *For some practical shortest path contexts it is essential that the termination state is ultimately reached. However, when a discount factor γ is introduced in such a problem, the character of the problem may be fundamentally altered. In particular, the threshold for an appropriate value of γ may be very close to 1 and may be unknown in practice.*

Based on the work of Mausam and Kolobov (2012), IHD-MDPs is a subclass of SSPs. This is proved by showing how to convert an IHD-MDP with discount factor γ into an SSP as follows (Kolobov, 2013; Tarbouriech, 2022):

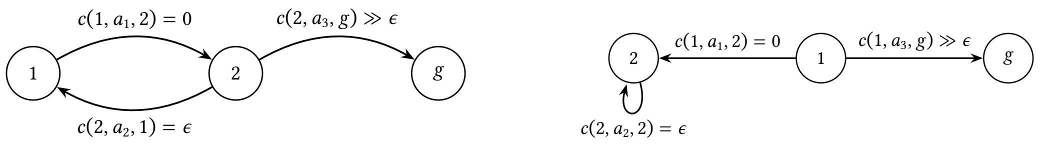
1. replace the reward function R with a cost function $C = -R$;
2. add a goal state s_g to the state space S;
3. add a transition that we can move to s_g in any timestep with probability $1 - \gamma$ and scale the transition probability function so that all previous transitions sum up to γ; and
4. the goal state s_g is absorbing with cost 0.

The conversion above shows that SSPs generalize the IHD-MDP framework, which means that IHD-MDP is a special case of SSP (i.e., IHD-MDP $\subset$ SSP). As a result, the solutions developed for SSPs could be valuable and helpful in solving complex problems that are still modeled as IHD-MDPs (Mausam and Kolobov, 2012), but not the contrary (Geffner and Bonet, 2013, p.82). Thus, the widespread belief pointed out by D. Bertsekas (2023) above, that discounted MDP can be used as a universal model, is truly questionable.

We explore this idea further in Chapter 4, where we show some examples that demonstrate the critical role and sensitivity of the discount factor γ in IHD-MDP. In that chapter, we will analyze scenarios where the choice of γ fundamentally alters the resulting optimal policy, sometimes leading the agent to prefer suboptimal behaviors that decrease the chance of reaching a goal.

1.2 Can simple versions of SSPs be solved as an Infinite-Horizon Discounted MDP?

For some simple versions of SSP problems, defining an appropriate discount factor that accurately reflects the desired solution is possible but it is a nontrivial challenge. E.g. in a SSP where it is important the goal state be finally reached, when a discount factor is considered, we can have an unexpectedly different result (D. Bertsekas, 2023).


$c(1, a_1, 2) = 0$
$c(2, a_3, g) \gg \epsilon$
1
2
g
$c(2, a_2, 1) = \epsilon$
2
$c(1, a_1, 2) = 0$
1
$c(1, a_3, g) \gg \epsilon$
g
$c(2, a_2, 2) = \epsilon$

(a) *SSP with no dead-end states.*

(b) *SSP with a dead-end state.*

Figure 1.1: *Two SSPs with states **1** (s_0), **2** and **g** (goal state); actions a_1, a_2 and a_3; and cost function $c(s, a_i, s')$ as depicted; adapted from D. Bertsekas (2023).*

Figure 1.1 illustrates two SSPs with three states where g is the goal state. By converting the SSP into an IHD-MDP we show how the chosen discount factor (γ) significantly impacts the solution of the problem. In the SSP depicted in Figure 1.1a, the agent has no transition cost of going from state **1** to state **2** through action a_1 and a small positive cost (ϵ) of going from state **2** to state **1** through action a_2. Also, there is a substantially larger cost of going from state **2** to the goal state **g** through action a_3. We define the ratio between the small cost and the larger cost as $\rho = \frac{c(2,a_2,1)}{c(2,a_3,g)}$. By using a discount factor $\gamma < 1 - \rho$ we obtain a policy that stays in the cycle between state **1** to state **2**, i.e., a policy that never reaches the goal state **g**. Conversely, by using $\gamma > 1 - \rho$, we obtain a policy that will reach the goal state **g**. Figure 1.1b illustrates a SSP with a goal state and a dead-end state. By using a discount factor $\gamma < 1 - \rho$ we obtain a policy that stays in the dead-end state **2**, and by using $\gamma > 1 - \rho$ we obtain a policy that will reach the goal state **g**. Thus, defining an appropriate γ for the converted IHD-MDP is crucial to find a desired policy for the original problem.

Notice that if we apply a Finite-penalty criterion on the SSP with dead-end state of Figure 1.1b, giving a penalty $D > \rho$ to state **2**, the IHD-MDP agent will choose a policy to the goal, regardless the discount factor. However, as mentioned before, adopting a domain independent approach it is not an easy task to estimate D.

1.3 Motivation and Goals

Recall that a key difference between an MDP and an SSP is that the latter explicitly includes a set of goal states to be reached and, instead of maximizing the total expected reward, the agent has to minimize the total expected cost towards a goal state, eventually avoiding dead-end states. Also recall that SSPs implicitly become a multi-objective optimization problem: we want to maximize the probability to reach the goal while minimizing the expected cost. A simple form of SSP assume there exists a complete proper policy, i.e., a policy that reaches the goal with probability 1 from any state (Mausam and Kolobov, 2012). However, many real-world problems break this assumption by having the presence of dead-end states (Kolobov and Weld, 2012) (Fig. 1.1b) and consequently there is no complete proper policy for them.

In an SSP with Avoidable Dead-Ends (SSPADE), a proper policy from an initial state s_0 is guaranteed to exist. However, finding this policy is non-trivial, as classical MDP algorithms like VI or LRTDP fail to converge due to the presence of the dead-ends. In an SSP with Unavoidable Dead-Ends (SSPUDEs for short) this issue is even worse, as no proper policy exists and these classical algorithms will never converge. Among the planning solutions for SSPs and extensions are: (a) VI with Finite Penalty and any Find& Revise algorithm to solve fSSPUDE and SSPADE problems (Kolobov and Weld, 2012); (b) the GPCI planning algorithm, (Teichteil-Königsbuch, 2012), which uses the S^3P optimization criterion; (c) the i-dual planning algorithm (Trevizan *et al.*, 2017), which uses the MCMP optimization criterion; and (d) the GUBS optimization criterion (Freire and Karina Valdivia Delgado, 2017) for SSPUDEs. While the former criteria handle the multi-objective problem lexicographically, GUBS uses a utility-based semantic to find a better trade-off between the objectives.

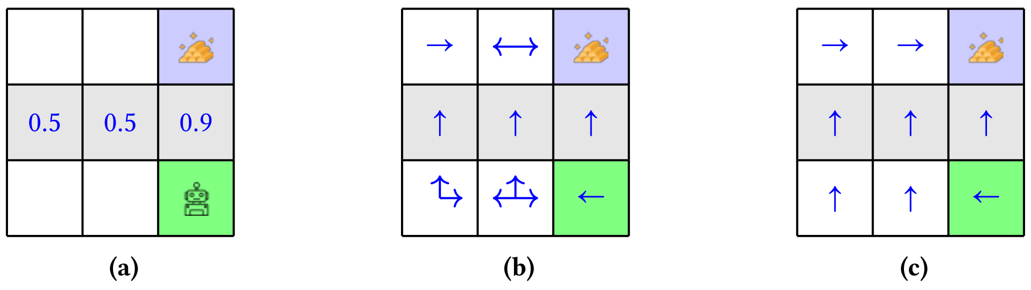


Figure 1.2: *An SSP instance of the Navigation domain: (a) probabilities to reach a dead-end when the agent passes through middle line locations; (b) a MaxProb solution; and (c) a MinCost solution that also satisfies the MaxProb solution.*

The Navigation Domain. This is a benchmark domain for SSPUDE problems, i.e. SSPs with unavoidable dead-ends. Consider the grid world of Fig. 1.2a where an agent must move from an initial state s_0 (green cell) to a goal state s_G (blue cell). The agent can move (at cost 1) up, down, left, and right with probability 1, in almost all locations. When reaching locations in the middle line, with probabilities 0.5, 0.5, and 0.9, respectively, the agent can "break" and no longer be able to move or accomplish its mission towards the goal, which configures a set of dead-end states (one for each middle location). The agent's objective is to find the path that maximizes the probability to the goal with minimum cumulative cost. Since dead-ends can not be avoided in this domain, an optimal policy could make a good commitment between the maximum probability to reach the goal and the minimum cumulative cost.

Fig. 1.2b shows a set of policies that maximize the probability to the goal, called MaxProb policies, without considering the costs; Note that the agent avoids going up in the right column from the initial state since it has the highest chance to break. Fig. 1.2c shows a solution that minimizes costs among all MaxProb policies which seems a better commitment between costs and probability to reach the goal.

1.4 Dissertation goal

The SSP formulation, which naturally models goal-directed behavior involves a multi-objective optimization problem and has been more thoroughly investigated within the planning community. However, it remains relatively underexplored in the RL literature. The aim of this work is to bridge the gap between these two areas offering the opportunity to apply reinforcement learning on more challenging goal oriented problems. Therefore, we draw inspiration from probabilistic planning approaches related to goal-oriented tasks, such as SSPs and their extensions, to integrate their optimization criteria into the Q-learning algorithm. This leads to the development of novel RL solutions for SSPs, particularly SSPADEs and SSPUDEs.

Our main contribution is to propose five new RL algorithms: Q-learning-FP (Q-learning with Finite Penalty), Q-learning-MaxProb (Q-learning for MaxProb), Offline Q-learning-S^{3}P, Offline Q-Learning-MCMP and Q-learning with ϵ-Greedy MaxProb-Exploration.

To evaluate our solutions, we show experiments in the Navigation domain of the International Probabilistic Planning Competition (IPPC) (Bryce and Buffet, 2008), and an extension with dead-end traps. We compare the results with the corresponding exact planning solutions. We also make a discussion about the results of using an IHD-MDP model with different values of discount factor.

1.5 Goals of this work

To summarize, the goals of this work are:

- Describe the challenges of using the IHD-MDP model in reinforcement learning to solve SSPs and variations;
- Propose new RL algorithms to solve SSPs with dead-ends, among them:
  - Q-Learning under the Finite Penalty optimization criterion, called Q-Learning-FP;
  - Q-Learning under the MaxProb optimization criterion, called Q-Learning-MaxProb;
  - Q-Learning under the S3P optimization criterion, called Q-Learning-S^{3}P;
  - Q-Learning under the MCMP optimization criterion, called Q-Learning-MCMP; and
  - Q-learning with a ϵ-greedy MaxProb exploration.
- Present an empirical evaluation of the proposed algorithms in the Navigation domain, and an extension called Navigation with DE-Trap domain, with avoidable and unavoidable dead-ends.
- Compare the proposed algorithms with their corresponding probabilistic planning solution.

1.6 Dissertation Organization

This dissertation is organized as follows:

- **Chapter 2** presents the formalization of Sequential Decision-Making as Markov Decision Processes (MDPs) problems. In addition, we describe classical algorithms to solve these problems.
- **Chapter 3** presents the formalization of Sequential Decision-Making as an Stochastic Shortest Path (SSP) problems and its extensions including avoidable dead-ends (SSPADE) and unavoidable dead-ends (SSPUDE). Moreover, we describe planning criteria to solve SSPs with dead-ends, and we also describe the Probabilistic Planning techniques to solve SSPs and extensions.
- **Chapter 4** presents examples to compare the solutions obtained by each criteria to solve SSPs with dead-ends, such as S^3P, MCMP and the IHD criterion.
- **Chapter 5** presents the Reinforcement Learning (RL) techniques to solve sequential decision-making problems formalized as SSPs. Furthermore, we propose RL algorithms to solve SSPs with dead-ends.
- **Chapter 6** presents the Experimental Analysis of the proposed algorithms.
- **Chapter 7** presents the Conclusions and Future Work.

Chapter 2

Background on Planning for Markov Decision Process

In this chapter, we define the concepts and formalization of Markov Decision Process (MDP) (PUTERMAN, 2014), such as Finite-Horizon MDPs and Infinite-Horizon MDPs. We also describe some classical methods to solve MDPs using Probabilistic Planning (MAUSAM and KOLOBOV, 2012).

2.1 Finite-Horizon MDP

As we briefly introduced in the previous chapter, sequential decision-making problems are defined using a mathematical framework called Markov Decision Process (MDP). In this section, we describe the main concepts of an MDP and its formal definition.

As the name suggests, sequential decision-making is a problem that involves making a sequence of decisions through time. Thus, time is a very important aspect of an MDP, because a decision that you make today can have an impact on another decision in the future. Consequently, if we wish to do a sequence of good decisions, we can not analyze each decision independently, but we need to analyze a sequence of decisions through some sequence of timesteps.

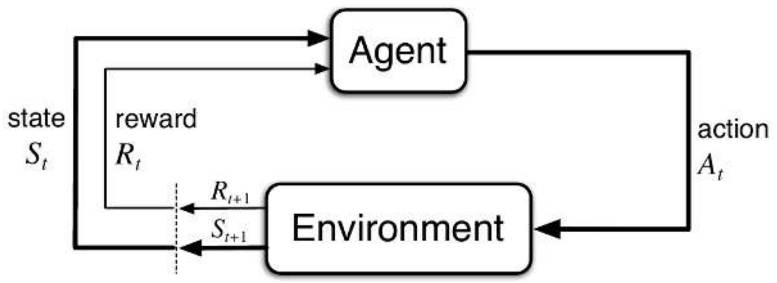


Figure 2.1: *Agent-environment interaction in an MDP (SUTTON and BARTO, 2018).*

Figure 2.1 illustrates a sequential decision-making process where *state* S_t represents what the agent observes from the environment at some time-step t then, using this information, the agent takes an *action* A_t in the environment chosen from a set of available actions; consequently, the agent receives a *reward* signal R_{t+1}. Then, the environment changes according with a *transition probability function* and the agent observes a new state and reward. The cycle repeats for the horizon of H time-steps and this sequence $[(S_0, A_0, R_1), ..., (S_{H-1}, A_{H-1}, R_H)]$ is what we call the process **trajectory** or **episode**. A solution for an MDP is an **optimal policy** that chooses the best action to be executed at each time-step.

In general, MDPs are classified based on the concept of the horizon H. When H is a finite value, it is referred to as a finite-horizon problem ($t = \{0, 1, 2, ..., H-1\}$). Similarly, if H is finite but can vary at each episode, we call it an indefinite horizon problem. In contrast, if $H = \infty$ we call it an infinite-horizon problem ($t = \{0, 1, 2, ..., \infty\}$).

Definition 2.1.1 (Finite-Horizon Markov Decision Process (MDP)). *A Finite-Horizon Markov Decision Process (MDP) is defined by the tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{H})$ *where:*

- $\mathcal{S}$ *is a finite set of states;*
- $\mathcal{A}$ *is a finite set of actions;*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *is the transition probability function;*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is the reward function; and*
- $\mathcal{H} \in \mathbb{N}$ *is a finite horizon of time-steps per episode.*

A policy for a Finite-Horizon Markov Decision Process depends on the current state and time-step, i.e., it is a **non-stationary policy**. In contrast, in the case of infinite and indefinite-horizon, we generally assume that our policies are stationary, because they are not time-dependent policies. Since in this work we are interested on IHD-MDPs and SSPs, we will not discuss solutions for this problem.

2.2 Infinite-Horizon Discounted MDP

Infinite-horizon Discounted MDP is a model broadly used for sequential decision making problems, either in automated probabilistic planning or reinforcement learning.

Definition 2.2.1 (Infinite-horizon Discounted MDP (IHD-MDP)). *An Infinite-horizon Discounted MDP is given by a tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ *where:*

- $\mathcal{S}$ *is a finite set of states;*
- $\mathcal{A}$ *is a finite set of actions;*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *is the transition probability function;*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is the reward function; and*
- γ *is the discount factor,* $0 < \gamma < 1$.

A policy π for an IHD-MDP can be deterministic or stochastic. A **deterministic policy** is defined by a mapping of state s to an action a, i.e.:

$$\pi : \mathcal{S} \rightarrow \mathcal{A} \tag{2.1}$$

A **stochastic policy** is defined by the probability distribution of taking action a given some state s

$$\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A}) \tag{2.2}$$

In order to find a policy that effectively solves an IHD-MDP, we need to assess the quality of a policy, that is, we need a measure of how good a policy is. This measure is called the **value function**, denoted by V^{π}, which estimates the value of a given policy π through the sequence of rewards.

Let G_t be the cumulative discounted reward the agent can receive at time-step t from an episode of an IHD-MDP, i.e.:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.3}$$

G_t is also known as the **discounted return** of an IHD-MDP episode which captures the overall performance of a policy during a single episode and serves as the basis for defining the expected discounted return criterion. This criterion is used to define the value function of a policy π (also called a state-value function), which estimates the expected return from a given state s under a fixed policy π.

Definition 2.2.2 (State-value function of a policy π). *Given an IHD-MDP, a policy π and a set of episodes of infinite time-steps generated with the execution of π, the value of a state s when we follow π is the expected value of G_t (return of the episodes), i.e.:*

$$V^{\pi}(s) = \mathbb{E}^{\pi}\left[G_t \middle| S_t = s\right] = \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right], \forall s \in \mathcal{S}. \tag{2.4}$$

Similarly, we can also estimate the value of a state when we execute an action a and follows a policy π afterwards. This is called the **action-value function**, denoted by $Q(s, a)$.

Definition 2.2.3 (Action-value function of a policy π). *Given an IHD-MDP, a policy π, the action-value function of a policy π at any state s and action a is defined by:*

$$Q^{\pi}(s, a) = \mathbb{E}^{\pi}\left[G_t \middle| S_t = s, A_t = a\right] = \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right], \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.5}$$

The Bellman equation (Puterman, 2014) breaks down the complex task of evaluating long-term rewards into more manageable sub-problems. It expresses the value function in terms of the immediate reward and the value of subsequent states. By iteratively applying this recurrence relation, we can efficiently update our estimates of the value function until they converge. This process not only dramatically reduces the computational burden compared to evaluating all policy combinations, but it also provides a systematic and principled approach to solve complex decision-making problems.

Definition 2.2.4 (Bellman equation).

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}^{\pi}\left[G_t \middle| S_t = s\right] \\ &= \mathbb{E}^{\pi}\left[R_{t+1} + \gamma G_{t+1} \middle| S_t = s\right] \\ &= \sum_{s'} \mathcal{T}(s, \pi(s), s') \left[\mathcal{R}(s, \pi(s)) + \gamma V^{\pi}(s')\right] \end{aligned} \tag{2.6}$$

As we already have a way to assess the value of a policy, we can try to obtain a policy that is optimal, i.e., a policy that maximizes the expected total discounted rewards. In that sense, the optimal policy π^* is a policy π from the set of all policies Π that maximizes the value function V^{π}, formally $V^*(s) \geq V^{\pi}(s), \forall s \in \mathcal{S}, \pi \in \Pi$.

Definition 2.2.5 (Optimal Value function). *The optimal value function (or the optimal state-value function) is defined by:*

$$V^*(s) := \max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S} \tag{2.7}$$

Definition 2.2.6 (Optimal Q function). *The optimal Q-function (or the optimal action-value function) is defined by:*

$$Q^*(s, a) := \max_{\pi} Q^{\pi}(s, a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{2.8}$$

By having the optimal value function, we can derive the optimal policy by choosing the action that has the maximum value in a state.

Definition 2.2.7 (Optimal Policy). *The optimal policy is defined by:*

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S} \tag{2.9}$$

Moreover, it can be shown that an optimal policy always exists, due to the fact that the Bellman equation for the optimal value V^* (or the Bellman optimality equation) has a unique solution.

Definition 2.2.8 (Bellman optimality equation).

$$\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}} Q^{\pi^*}(s, a) \\
&= \max_a \mathbb{E}^{\pi^*}\left[G_t \middle| S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}^{\pi^*}\left[R_{t+1} + \gamma G_{t+1} \middle| S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}\left[R_{t+1} + \gamma V^*(S_{t+1}) \middle| S_t = s, A_t = a\right] \\
&= \max_a \sum_{s'} \mathcal{T}(s, a, s')\left[\mathcal{R}(s, a) + \gamma V^*(s')\right]
\end{aligned} \tag{2.10}$$

The uniqueness of the solution ensures that the process of solving the Bellman optimality equation leads to consistent and well-defined optimal behavior, providing a solid theoretical foundation for algorithms designed to solve MDPs.

2.3 Solving IHD-MDPs

Two classical approaches for solving IHD-MDPs are dynamic programming and linear programming. First, we present Dynamic Programming (DP), where the Bellman optimality equations are used in iterative algorithms. We show how each DP method successively refines value and policy estimates until convergence to the optimal solution. Second, we introduce the Linear Programming (LP) formulation, in which the Bellman conditions are defined as a set of linear constraints and an objective function that minimizes the expected discounted cost (or maximizes reward).

2.3.1 Dynamic Programming Methods

The most common method to solve MDPs is Dynamic Programming (DP). This method is exact and solves the Bellman optimality equation (Equation 2.2.8) iteratively, assuming that we have access to the transition probability function. They are proven to converge to a fixed point following the contraction property (Puterman, 2014).

The primary algorithm for MDPs based on DP is Value Iteration (VI) (Bellman, 1957), which defines an initial arbitrary value for each state using the value function and iteratively updates the value function for all states of the MDP, using the Bellman update operator, i.e.:

$$V_{n+1}(s) \leftarrow \max_{a \in \mathcal{A}} \{\mathcal{R}(s, a) + \gamma \sum_{s' \in S} \mathcal{T}(s, a, s') V_n(s')\}. \tag{2.11}$$

In each iteration n, VI computes a new value for each state by considering the expected return from taking the best possible action in that state, given the current value estimates. This process is repeated until the values converge and, as the number of iterations approaches infinity, VI is guaranteed to converge, under the assumption that the MDP is finite and the discount factor is less than 1.

The convergence of VI is assessed by monitoring the difference between successive value function estimates. Specifically, the algorithm terminates when the maximum norm of the difference between V_{n+1} and V_n, called residual error, is less than a threshold ϵ, that is

$$\|V_{n+1} - V_n\|_\infty \leq \epsilon. \tag{2.12}$$

If the value function converges under this condition, we call that the value function is ϵ-consistent. Once the algorithm has converged, an optimal or ϵ-policy π can be extracted by selecting, for each state, the action that maximizes the expected return according to the computed value function.

Algorithm 1 Value Iteration

```
Require: MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, small threshold $\epsilon$
Ensure: Policy $\pi$
 1: Initialize $V_0(s) \leftarrow 0$, for all $s \in \mathcal{S}$
 2: $n \leftarrow 0$
 3: repeat
 4:     for $s \in S$ do
 5:         $V_{n+1}(s) \leftarrow \max_{a \in A}\{\mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s')V_n(s')\}$
 6:     end for
 7:     $n \leftarrow n + 1$
 8: until $\|V_{n+1} - V_n\|_\infty \leq \epsilon$
 9: for $s \in S$ do
10:     $\pi(s) \in \arg\max_{a \in \mathcal{A}}\{\mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s')V_n(s')\}$
11: end for
12: return $\pi$
```

Algorithm 1 initializes the value of $V(s)$ for all $s \in \mathcal{S}$ and repeat the update rule (Line 5) until the residual error be less or equal to ϵ (Line 8). After convergence we can extract the optimal (greedy) policy (Lines 9 to 11).

Another related method is Policy Iteration (PI) (Howard, 1960), which is an algorithm that solves MDPs iteratively by alternately performing two-steps: policy evaluation and policy improvement. In Step 1, the current policy is evaluated through a value function, using the Bellman equation for the policy. This step involves iterating over all states and computing the expected return for each state, based on the current policy. In Step 2, the policy is updated by choosing, for each state, the action that maximizes the expected return according to the newly evaluated value function. The process repeats until the policy stabilizes, at which point the algorithm has converged to the optimal policy. It is also important to note that PI and VI are very closely related. In fact, VI can be seen as a special case of PI, where the policy evaluation step is stopped after just one update, instead of being run until full convergence.

Many DP algorithms proposed to solve MDPs are extensions of the classical VI and PI methods. These variations aim to improve the efficiency of value function estimation by modifying the way value updates are performed. Two prominent examples are Asynchronous Value Iteration (Asynchronous VI) and Prioritized Value Iteration (Prioritized VI), both of which focus on updating the value function asynchronously. In these methods, each state is updated independently, rather than updating all states simultaneously, as is

Algorithm 2 Policy Iteration

```
Require: MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, small threshold $\epsilon$
Ensure: Policy $\pi$
 1: Initialize arbitrary policy $\pi(s)$ for all $s \in \mathcal{S}$
 2: Initialize $V_0(s) \leftarrow 0$, for all $s \in \mathcal{S}$
 3: $n \leftarrow 0$
 4: repeat                                                    ▷ Policy Evaluation
 5:     repeat
 6:         for $s \in \mathcal{S}$ do
 7:             $V_{n+1}(s) \leftarrow \mathcal{R}(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') V_n(s')$
 8:         end for
 9:         $n \leftarrow n + 1$
10:     until $\|V_{n+1} - V_n\|_\infty \leq \epsilon$
11:     policy_stable ← True                                  ▷ Policy Improvement
12:     for $s \in \mathcal{S}$ do
13:         $a_{\text{old}} \leftarrow \pi(s)$
14:         $\pi(s) \in \arg\max_{a \in \mathcal{A}} \{\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') V_n(s')\}$
15:         if $a_{\text{old}} \neq \pi(s)$ then
16:             policy_stable ← False
17:         end if
18:     end for
19: until policy_stable
20: return $\pi$
```

done in traditional VI. This asynchronous approach allows for more flexibility and can lead to faster convergence in certain settings, especially when states have varying levels of importance or are connected in complex ways.

Furthermore, methods such as Topological Value Iteration (Topological VI) (Dai *et al.*, 2011) take a different approach by representing the MDP as a graph and exploiting the structure of the graph to speed up the process. In this approach, the state space is partitioned into strongly connected components (SCCs), which are subsets of states in which any state can reach any other state within the same component. Using the structure of the graph, Topological VI can update the states within each SCC more efficiently, potentially leading to significant performance improvements. This partitioning helps to identify independent subproblems within the MDP, allowing for localized updates that can speed up convergence and reduce computational overhead. In some cases, such partitioning into SCCs can lead to substantial performance gains, particularly when the state space exhibits a natural graph structure, as it allows for targeted, localized updates that avoid redundant calculations across loosely connected regions of the state space.

2.3.2 Linear Programming Methods

There are alternative methods for solving MDPs, one of the most notable being Linear Programming (LP). The general idea is to formulate a linear program to estimate the value function through a linear objective function subject to linear constraints. In the case of

MDPs, the Bellman optimality conditions can be written as a set of linear inequalities, known as the Bellman optimality constraints, which ensure that the value of each state is at least as large as the expected reward plus the discounted value of the next state for any action. The goal is to find the optimal value function by minimizing a linear objective function, such as the sum of the state values, subject to these constraints. The solution to the LP yields a set of values that correspond to the optimal value function of the MDP.

$$\begin{aligned}
&\text{Variables} \quad V(s),\ \forall s \in \mathcal{S};\\
&\text{Minimize:} \quad \textstyle\sum_s \alpha(s)V(s);\\
&\text{Constraints:} \quad V(s) \geq \mathcal{R}(s,a) + \gamma \textstyle\sum_{s'} \mathcal{T}(s,a,s')V(s')\\
&\forall s \in \mathcal{S},\ \forall a \in \mathcal{A};\\
&\text{where } \alpha(s) > 0 \text{ is the state relevance weight for the state } s, \forall s \in \mathcal{S}.
\end{aligned} \tag{2.13}$$

Although this method guarantees an exact solution, it comes with practical limitations. While the LP approach is polynomial in nature with respect to the number of states and actions, it can become computationally expensive for large-scale problems. Specifically, solving the LP can require considerable memory and processing time as the size of the state and action spaces increases. This issue becomes particularly problematic for MDPs with a large number of states or when the transition probabilities are complex. As the number of variables and constraints grows, LP methods suffer from scalability issues, making them infeasible for high-dimensional problems where state and action spaces are large, such as in real-world applications involving large-scale systems or continuous state spaces.

However, one positive aspect of LP methods is their ability to naturally express constraints. This feature is particularly useful in solving Constrained Markov Decision Processes (CMDPs) (Altman, 1999), where additional constraints on the system, such as resource limits or safety requirements, must be incorporated into the decision-making process.

Chapter 3

Background on Planning for Stochastic Shortest Paths

As mentioned before, there is a class of goal-oriented MDP problems that generalizes the Finite-horizon and Infinite-horizon MDP. This class is the Stochastic Shortest Paths (SSP) which focus on obtaining policies that guarantee to reach the goal while minimizing the cumulative undiscounted costs.

In this chapter, we formalize Stochastic Shortest Paths (SSP) problems (D. P. Bertsekas and J. N. Tsitsiklis, 1991; D. P. Bertsekas, J. N. Tsitsiklis, and J. Tsitsiklis, 1996) and their extensions on the presence of avoidable and unavoidable dead-ends, called SSPADE and SSPUDE, respectively (Kolobov and Weld, 2012). In addition, we describe the main known methods to solve SSPUDEs using specialized solutions proposed for probabilistic planning, that are the iSSPUDE (Mausam and Kolobov, 2012), S^3P (Teichteil-Königsbuch, 2012) and MCMP (Trevizan *et al.*, 2017) optimality criteria.

3.1 Stochastic Shortest Path without Dead-Ends

In SSPs, a goal reachability guarantee can be formalized as the existence of a policy under which the goal state is reached with probability 1, independently of the initial state s_0. These policies are called *proper policies*.

Definition 3.1.1 (Proper Policy). *A policy π is called* ***proper*** *if it reaches the goal with probability 1, $\forall s \in \mathcal{S}$. A policy that is not proper is called* ***improper*** *(D. Bertsekas, 2022).*

Furthermore, SSPs are characterized by an indefinite horizon, meaning that while we assume the agent will eventually reach the goal, we do not know exactly when this will occur. Unlike finite-horizon MDPs, where the number of decision steps is fixed in advance, or infinite-horizon MDPs, where we assume the number of timesteps as infinite, SSPs without dead-ends operate under the premise that the decision process terminates upon reaching the goal. However, the number of steps required to reach the goal is not known a priori and may vary depending on the environment dynamics and the policy being followed.

Another important aspect is that in an SSP we do not have the notion of a discount, as in the Infinite-horizon MDP. Thus, in an SSP, we assume that the discount factor is equal to 1, i.e. $\gamma = 1$, reflecting the fact that all future costs are treated equally until the goal is reached. Additionally, we assume that any policy that fails to reach the goal incurs an infinite cost, that is, the values of improper policies are infinite. This assumption naturally induces a preference for proper policies, which guarantee goal attainment, and discourages the selection of improper policies.

Definition 3.1.2 (Stochastic Shortest Path). *A Stochastic Shortest Path (SSP) is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G})$ where:*

- *$\mathcal{S}$ is a finite set of states;*
- *$\mathcal{A}$ is a finite set of states actions;*
- *$\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function;*
- *$\mathcal{C} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{>0}$ is the cost function, that is strictly positive, except in the transition to the goal state;*
- *s_0 is the initial state;*
- *$\mathcal{G} \subseteq \mathcal{S}$ is the set of goal states, each goal state $s_g \in \mathcal{G}$ has the following transition probabilities $\mathcal{T}(s_g, a, s_g) = 1, \mathcal{T}(s_g, a, s') = 0, \forall a \in \mathcal{A}, \forall s' \neq s_g$; and the cost function $\mathcal{C}(s_g, a, s_g) = 0$.*

In addition, an SSP makes the following assumptions:

Assumption 1. *There exists at least one proper policy.*

Assumption 2. *The value of improper policies has infinite cost.*

Following the assumption that $\gamma = 1$ in an SSP problem, we extend the definitions introduced in IHD-MDP (Section 2.2)) to evaluate policies based on their expected cumulative cost. Specifically, for a given policy π, we define the cumulative expected cost as $G_t = \sum_{k=0}^{\infty} C_{k+t+1}$. Note that in the SSP of Definition 3.1.2, there are no dead-end states, all episodes will eventually reach the goal in a finite number of steps and G_t will converge to a finite value.

Definition 3.1.3 (State-value function of a policy π). *Given an SSP and a policy π, we assess the expected return for a state using the state-value function*

$$V^{\pi}(s) = \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty} C_{t+k+1} \middle| S_t = s\right], \forall s \in \mathcal{S}. \tag{3.1}$$

Similarly, we can measure the expected return for a pair of a state s and an action a, following a policy π, using the Action-value function (or a Q-function).

Definition 3.1.4 (Action-value function of a policy π).

$$Q^{\pi}(s, a) = \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty} C_{t+k+1} \middle| S_t = s, A_t = a\right], \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{3.2}$$

Being $V^*(s)$ the optimal cost-to-go, i.e, the optimal accumulated expected cost from s and following an optimal policy, the solution of an SSP can be specified by a unique fixed-point solution for the following set of equations, known as Bellman equations (Trevizan *et al.*, 2017) for SSPs without dead-ends:

$$V^*(s) = \begin{cases} 0, & \text{if } s \in \mathcal{G}, \\ \min_{a \in \mathcal{A}} \mathcal{C}(s,a) + \sum_{s' \in S} \mathcal{T}(s,a,s') V^*(s'), & \text{otherwise.} \end{cases} \tag{3.3}$$

Note that Equation 3.3 does not include a discount factor and, due to the absorbing goal states with cost 0, the horizon can be seen as (inde)finite. Any greedy policy w.r.t. to the optimal value function $V^*(s)$ of Equation 3.3 is an optimal policy.

3.2 Stochastic Shortest Path with Dead-Ends

What happens if we allow dead-end states and use an algorithm such as VI or PI as we have previously defined?

Definition 3.2.1 (Dead-end state). *A dead-end state s is a state in which under any policy the probability of reaching the goal is zero. We call the set of dead-ends of a given sequential decision making problem as S_{DE}.*

It is easy to see that $V^*(s)$ for a dead-end state s, the Equation 3.3 is not well-defined and it diverges to infinity, because the dead-end state have infinite cost. Therefore, the algorithm will never converge.

One way to incorporate dead-ends into SSPs while preserving the properties of existing solution techniques (e.g. VI or PI) for these problems is to assume that entering a dead-end, although highly undesirable, has an extra finite cost. To implement the finite-cost assumption, we could assign a positive penalty D for visiting a dead-end and no other change for other states. The semantics of this SSP would be that the agent pays D when encountering a dead-end, and the process stops. However, this modification to SSPs cannot be directly operationalized, since the set of dead-ends is not known a priori and needs to be inferred while planning/learning.

Therefore, we may change the semantics of the finite-penalty model as follows. Whenever the agent reaches any state with the expected cost of reaching a goal equaling D or greater, the agent simply pays the penalty D and "gives up", i.e., the process stops. The benefit of putting a bound on all state values is that the value function of a state under any policy becomes bounded. Therefore, we avoid the difficulty of explicitly detecting dead-ends, either a priori or while planning, by implicitly assuming that dead-end states will converge to a value of D.

$$V^*(s) = \begin{cases} 0, & \text{if } s \in \mathcal{G}, \\ D, & \text{if } s \in S_{DE}, \\ \min\{D, \min_{a \in \mathcal{A}} \mathcal{C}(s,a) + \sum_{s' \in S} \mathcal{T}(s,a,s') V^*(s')\} & \text{otherwise.} \end{cases} \tag{3.4}$$

This notion of using a finite penalty for the dead-ends and giving a bound to $V^*(s)$ is formalized in Equation 3.4 and will be described in more detail later in this chapter.

It is important to note that the penalty D needs to be chosen carefully, because we may incorrectly consider a "good state" as a dead-end. For example, suppose that we set $D = 10$, and we have a state s that reaches the goal G, but with a high expected cost $V(s) = 50$. In this case, since $V(s) > D$, applying Equation 3.4 would result in limiting the value of $V(s)$ to D, thus, preventing the finding of a policy that reaches the goal state.

3.2.1 Stochastic Shortest Path with Avoidable Dead-Ends (SSPADE)

If the SSP assumptions from Definition 3.1.2 do not hold, i.e., the space has dead-end states, the solution of Equation 3.3 can fail. In the case of avoidable dead-ends (called SSPADE), we can find a solution if there is a proper policy only w.r.t. the initial state s_0, not everywhere (Kolobov, M. Mausam, *et al.*, 2011; Geffner and Bonet, 2013).

Definition 3.2.2 (Stochastic Shortest Path with Avoidable Dead-Ends (SSPADE)). *Stochastic Shortest Path with Avoidable Dead-Ends (SSPADE) is defined by the tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G})$ *where:*

- $\mathcal{S}$ *is a finite set of states;*
- $\mathcal{A}$ *is a finite set of states actions;*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *is the transition probability function;*
- $\mathcal{C} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{>0}$ *is the cost function, that is strictly positive, except in the transition to the goal state;*
- s_0 *is the initial state; and*
- $\mathcal{G} \subseteq \mathcal{S}$ *is the set of goal states.*

An SSPADE also makes the following assumption:

Assumption 3. *There exists at least one proper policy starting from the initial state* s_0 *and the value of improper policies has infinite cost.*

The objective in SSPADE is to determine a proper policy π^* that avoids the dead-end states s_{DE} from s_0. This policy needs to ensure that the goal state is reached with probability 1, while minimizing the expected cumulative cost.

Thus, similarly to an SSP without dead-ends, a solution of a SSPADE can be specified by a unique fixed-point of Equation 3.3 and should return a policy rooted in s_0 that is capable of reaching the goal with probability 1 while avoiding all dead-ends with minimum expected accumulated cost. This is because Assumption 3 ensures that all improper policies have infinite cost. However, we cannot use VI or PI for SSPADEs, as we have discussed above, i.e., these algorithms will never converge for the set of dead-states or states that can inevitable can lead to them. One solution is to apply the idea of penalty and use Equations 3.4 assuming we know the set of dead-end states and we have an appropriate value for D. Kolobov and Weld (2012) show that in practice, if we do not know how to detect dead-end states, finding a policy that satisfies only the 1st and 3rd equations of the Equations 3.4, will result in an optimal (proper) policy for a SSPADE, as we discuss in more details in Section 3.3.1.

3.2.2 Stochastic Shortest Path with Unavoidable Dead-Ends (SSPUDE)

In an *SSPUDE*, since Assumptions 1, 2 and 3 do not hold, there exists a set of unavoidable dead-end states $S_{DE} \subset S \setminus G$. In that case, all solutions have a positive probability of going into a dead-end.

Definition 3.2.3 (Stochastic Shortest Path with Unavoidable Dead-Ends (SSPUDE)). *Stochastic Shortest Path with Unavoidable Dead-Ends (SSPUDE) is defined by the tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G}, \mathcal{D})$ *where:*

- $\mathcal{S}$ *is a finite set of states;*
- $\mathcal{A}$ *is a finite set of actions;*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ *is the transition probability function;*
- $\mathcal{C} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}_{>0}$ *is the cost function, that is strictly positive, except in the transition to the goal state;*
- s_0 *is the initial state;*
- $\mathcal{G} \subseteq \mathcal{S}$ *is the set of goal states;*
- $D \in \mathbb{R}^+ \cup \{+\infty\}$ *is the penalty.*

Assumption 4. *For every improper policy π, for every $s \in \mathcal{S}$ where π is improper, the value of policy π at s is infinite.*

If $D < \infty$, we call it Finite-Penalty SSPUDE (fSSPUDE) and we can use the solution of Equations 3.4.

If $D = \infty$, we call it Infinite-Penalty SSPUDE (iSSPUDE).

In an *SSPUDE*, unlike in SSPADEs, no policy π can prevent reaching certain dead-end states from any initial state s_0 due to the stochastic nature of the transitions. Thus, the goal of solving an *SSPUDE* is to find a policy π that minimizes the expected cumulative cost to reach a goal state $s_g \in \mathcal{G}$, while maximizing the probability of reaching a goal state from the initial state s_0, even though some paths inevitably lead to dead-ends in S_{DE}.

3.3 Solving SSPs with Dead-Ends

Unlike *SSPADE*s, no policy $\pi : S \rightarrow A$ can prevent reaching certain states in S_{DE} from s_0. Thus, we need a new definition of proper policy for SSPUDEs, which is a policy that has positive probability to reach the goal while an improper policy has probability 0 to reach the goal. Thus, the challenge of solving an *SSPUDE* is to find a policy π that minimizes the expected cumulative cost while ensuring that the probability of reaching a goal state $s_g \in G$ from s_0 is maximized, even when some paths inevitably lead to dead-ends.

To address the problems introduced by SSPUDEs, several authors of the probabilistic planning area proposed different optimization criteria (Kolobov and Weld, 2012; Teichteil-Königsbuch, 2012; Trevizan *et al.*, 2017; Freire and Karina Valdivia Delgado, 2017). By using these criteria, the Bellman equation is well-defined and we can obtain optimal policies that reaches the goal. Also, these criteria are able to solve SSPUDEs even if the set of dead-end states is unknown.

3.3.1 Finite Penalty Criterion

As we briefly discussed before, the Finite Penalty criterion (Kolobov and Weld, 2012), assumes that going into a dead-end, despite being undesirable, has a finite cost. This modification guarantees that the value function will be bounded in any policy.

Definition 3.3.1 (Finite Penalty Criterion). *Given an SSPUDE $\mathcal{M}' = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$ and a dead-end penalty $D \in \mathbb{R}^*_+$, called fSSPUDE model, find a greedy policy π^{FP} for the unique fixed-point solution of:*

$$V^*(s) = \begin{cases} 0, & \textit{if } s \in \mathcal{G}, \\ \min\{D, \min_{a \in A(s)} \mathcal{C}(s,a) + \sum_{s' \in S} \mathcal{T}(s,a,s')V^*(s')\}, & \textit{otherwise.} \end{cases} \tag{3.5}$$

The Equation 3.5 has a subtle difference in relation to Equation 3.4, as in many applications, we do not know the set of dead-ends beforehand, thus, we cannot apply the former equation. In practice, to solve problems using the Finite-Penalty criterion, we use the Equation 3.4 which does not depend on the set of dead-ends. Additionally, we can optionally infer the dead-end states implicitly during planning/learning.

In addition, note that Equation 3.5 is equivalent of transforming the original SSP $\mathcal{M}$ into a new SSP $\mathcal{M}' = \langle S, A', T', C', s_0, G \rangle$ where, $\forall s \in \mathcal{S}$, $A'(s) = A(s) \cup \{\text{give-up}\}$, $C'(s, \text{give-up}) = D$, $C'(s,a) = C(s,a)\ \forall a \in A(s)$, and $\mathcal{T}(s, \text{give-up}, s_g) = 1$ for any $s_g \in G$. With this new SSP, M' has no dead-ends and can be solved using any probabilistic planning algorithm for SSPs (Trevizan *et al.*, 2017).

Despite the useful properties of guaranteeing that the value function is bounded, one main limitation of the Finite-Penalty criterion for SSPUDEs, is that a state s can be viewed as a dead-end if it has an expected cost of reaching the goal greater than the penalty D (Trevizan *et al.*, 2017). This can be a significant limitation as it can lead to suboptimal policies.

To overcome the limitation of defining a correct penalty value, one alternative is to directly maximize the probability of reaching the goal, ignoring the expected cost, a criterion known as *MaxProb*. Note that since there are several policies that can go with maximum probability to the goal, we can later choose the ones with minimal expected cumulated cost over those MaxProb solutions, as we will discuss in Sections 3.3.2 and 3.3.3.

3.3.2 MAXPROB Criterion

Definition 3.3.2 (MAXPROB Criterion). *Given a SSPUDE $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, C, s_0, \mathcal{G} \rangle$, the objective of the MAXPROB criterion is to find a policy π^{MP} that, when followed, maximizes the probability of reaching a goal state $s_g \in G$.*

To derive a MAXPROB problem from a SSPUDE we modify the cost function C to be a reward function $\mathcal{R}$ (rewarding states for reaching the goal) obeying two conditions:

$$\mathcal{R}(s,a) = \begin{cases} 0, & \forall s \in \mathcal{S} \setminus \mathcal{G}, \forall a \in \mathcal{A} \\ 1, & \forall s \in \mathcal{G}, \forall a \in \mathcal{A} \end{cases} \tag{3.6}$$

Since MaxProb is now reward-based, an optimal solution is to maximize the cumulative expected reward (i.e. probabilities) of reaching any goal state (Kolobov and Weld, 2012). This allows to compute the Bellman equation for the optimum probability-value function $P^*(s), \forall s \in \mathcal{S}$, as follows:

$$P^*(s) = \begin{cases} 1, & \forall s \in \mathcal{G}, \\ \max_{a \in \mathcal{A}} \sum_{s' \in S} \mathcal{T}(s,a,s') P^*(s'), & \forall s \notin \mathcal{G}, \end{cases} \tag{3.7}$$

and, $\pi^{MP}(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in S} \mathcal{T}(s,a,s') P^*(s')$.

In addition, note that an optimal policy that satisfies Equation 3.7 may be a non-deterministic policy, since more than one policy can reach the goal with the same probability. In this case, as we have mentioned before, we can have multiple fixed-point solutions, thus, another criteria propose to, not only maximizing the probability, but also considering the costs.

MaxProb belong to a broader class of SSPs, called Generalized SSP (Kolobov and Weld, 2012), that relax the SSP assumptions. In particular they do not require the existence of a complete proper policy, neither that all improper policies have infinite cost. In fact, Equation 3.7 have several non-optimal fixed-point solutions. Thus, to solve MaxProb we need to define a special VI method, called VI^{MP} that can be initialized with an arbitrary heuristic function, but instead of the Bellman backup operator it uses its generalized version that we call *Bellman backup with Escaping Traps* (BET) (Kolobov and Weld, 2012). For MaxProb the Bellman backup can have many fixed points. So to check whether P^* is optimal, BET applies the trap elimination operator to it, which involves constructing the transition graph that uses actions of all policies greedy w.r.t. P^* and eliminate traps.

Definition 3.3.3 (Trap (Kolobov, M. Mausam, *et al.*, 2011)). *A* ***trap*** *is a maximal strongly connected component (SCC) $C = (\mathcal{S}_C, \mathcal{A}_C)$ of the greedy graph with respect the probability-value function P, called G_P, with the following properties:*

- *For all $g \in G$, $g \notin \mathcal{S}_C$.*
- *G_P has no edges (s_i, s_j) such that $s_i \in \mathcal{S}_C$ and $s_j \notin \mathcal{S}_C$. In particular, there is no path in G_P from any state $s \in \mathcal{S}_C$ to a goal $g \in G$.*

Informally, a trap is just a strongly connected component of the greedy graph G_P with no outgoing edges in G_P and no goal states. A permanent trap consists of a set of states from that does not have any outgoing edges in G_P. Therefore, no policy can ever reach a goal from any of the states in a permanent trap, i.e. all states in permanent traps are dead ends. For states in transient traps, a proper policy may exist, but it can not be greedy w.r.t. the current probabilistic-value function P.

Solutions for MaxProb are heuristic search algorithms (Trevizan *et al.*, 2017) such as FRET (Kolobov, M. Mausam, *et al.*, 2011) and FRET-π (Steinmetz *et al.*, 2016). Both these algorithms iterate between: (i) finding a fixed-point solution for P^* using any optimal heuristic search algorithm for SSPs; and (ii) eliminate traps from the greedy policies w.r.t. P^*.

3.3.3 S^{3}P Criterion

Given an iSSPUDE, the Stochastic Safest and Shortest Path Criteion (S^{3}P) (Teichteil-Königsbuch, 2012) uses the result of solving the MaxProb problem derived from iSSPUDE (Kolobov and Weld, 2012), and apply a second optimization step considering the cost of the original iSSPUDE.

To analyze the criterion more precisely, we first start by defining the behavior of this criterion through the notion of traces. A trace T is a sequence of transitions of the form $T = (T^1, T^2, \dots, T^{|T|+1})$, where each T^i represents a visited state s at the timestep i, and the transition from T^i to T^{i+1} is determined by a policy π. At each state T^i, the policy selects an action $\pi(T^i)$, which leads probabilistically to the next state T^{i+1} according to the transition model $P(T^{i+1} \mid T^i, \pi(T^i))$. In the case of SSPUDEs, these traces can be either finite or infinite. In the finite case, the final state s is a goal state ($s \in G$), and in the infinite case it goes into a dead-end state ($s \in S_{DE}$) (Trevizan *et al.*, 2017).

Thus, we can derive the probability of observing a finite trace under some policy π by the following Equation:

$$P(T) = \prod_{i=1}^{|T|} P(T^{i+1} \mid T^i, \pi(T^i)) \tag{3.8}$$

Similarly, we can obtain the cost of a trace by the sum of the costs taken at each state:

$$C(T) = \sum_{i=1}^{|T|} C(T^i \mid \pi(T^i)), \tag{3.9}$$

As the costs are all positive, a policy that causes the agent to enter an infinite trace has infinite cost. Thus, as these traces never reaches the goal state, we can say that these traces are infinite and its cost diverges, i.e., $C(T) = \infty$.

To better understand the implications of different types of traces, we can define $T_{\pi,s}$ as the set of all possible traces generated by executing a policy π starting from a state s. As we mentioned, $T_{\pi,s}$ may be infinite, particularly in environments that contain cycles or nonterminal dead-end states. In such cases, it is helpful to analyze the behavior of the policy by partitioning $T_{\pi,s}$ into two subsets.

Let $T^G_{\pi,s}$ denote the subset of traces that eventually reach a goal state. These traces are always finite by definition, since they terminate once a goal is reached. On the other hand, let $T^{DE}_{\pi,s}$ represent the set of traces that fail to reach a goal. These include traces that either end in a terminal dead-end or continue indefinitely due to nonterminal dead-end states.

This decomposition of $T_{\pi,s}$ into $T^G_{\pi,s}$ and $T^{DE}_{\pi,s}$ provides a useful framework to understand how a policy behaves in terms of its successful and unsuccessful trajectories, and the difference between each dual-optimization criterion.

The S^{3}P criterion (Teichteil-Königsbuch, 2012) involves deriving an optimal policy in two steps: (Step-1) identifying policies that maximizes the probability of reaching the goal, i.e., finding a policy that satisfies $P^*(s), \forall s \in \mathcal{S}$ using the Equation 3.7 (Definition 3.3.2); (Step-2) deriving a min-cost policy considering only the paths leading to the goal with maximum probability. Thus, we can now define the S^{3}P Criterion on Definition 3.3.4. We highlight that the S^{3}P Criterion only considers successful traces that reach the goal.

Definition 3.3.4 (S^{3}P Criterion).

$$\pi = \arg\min_{\pi \in \Pi^{MP}} \mathbb{E}[C(T) | T \in T^G]$$

Thus, in this criterion, the set of actions applicable in a given state is restricted to those from a MaxProb policy, denoted by $\pi^{MP}(s)$. The value function $V^*(s)$ is then defined in Equation 3.10 by a conditional Bellman equation (adapted from Teichteil-Königsbuch (2012)), which calculates the expected cost given that the goal is successfully reached.

The value is defined as $V^*(s) = 0$ if the probability of reaching the goal is zero $P^*(s) = 0$, otherwise:

$$V^*(s) = \min_{a \in \pi^{MP}(s)} \frac{\sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') P^*(s') \left[\mathcal{C}(s,a) + V^*(s') \right]}{P^*(s)} \tag{3.10}$$

This criterion improves the MaxProb, because it tries to maximize the probability of reaching the goal, but it does not ignore the costs. Thus, in situations where multiple policies achieve the maximum probability of reaching the goal, this criterion enables us to differentiate among them by selecting the policy that results in the lowest expected cost. In this way, it provides a more balanced and realistic evaluation of policy performance.

However, one drawback of S^{3}P is that it overlooks the costs of dead-end states. It calculates the expected cost by considering only the states that successfully reach the goal, thus, it excludes the contribution of trajectories that terminate in dead-ends. As a result, the value function can underestimate the true cost of following a given policy, especially in environments where dead-ends are unavoidable. An alternative criterion that handles this limitation is the Min-Cost given Max-Prob (MCMP) that we will discuss in the next section.

3.3.4 The MCMP Criterion

The Min-Cost given Max-Prob (MCMP) criterion (Trevizan *et al.*, 2017) is another approach, similar to S^{3}P criterion, that is also designed to address SSPs with dead-ends by obtaining a policy that does not only minimizes cost but also maximizes the probability of reaching the goal. This criterion uses a Linear Programming (LP) formulation, to solve a dual optimization problem using two related LPs: LP 1 for minimizing cost and LP 2 for maximizing goal reaching probability.

To make sure that even policies that goes into a infinite trace are finite, the MCMP criterion applies a trace truncation function defined below:

Definition 3.3.5 (Trace truncation function). *Let $\psi : T \mapsto T$ be the function:*

$$\psi(s_i s_{i+1} \cdots) = \begin{cases} s_i & if |T| = 1 \; or \; P(T^G_{s_i}) = 0 \\ s_i \; \psi(s_{i+1} \cdots) & otherwise \end{cases}$$

This trace truncation identify the dead-end states that have zero-probability of reaching the goal ($P(T^G_{s_i}) = 0$), and truncate the trace in the first encountered dead-end state. Then, given an infinite trace $T \in T^{DE}_{\pi,s}$ the result of $\psi(T)$ is always finite. In the case of traces that are already finite, such as traces that eventually reaches the goal, the trace truncation function returns the finite trace itself, i.e, given a finite trace $T \in T^G_{\pi,s}$, $\psi(T) = T$.

Now, we can define the MCMP Criterion, where we try to obtain a policy that minimizes the costs of these traces, but under policies that reaches the goal with maximum probability ($\pi \in \Pi^{MP}$).

Definition 3.3.6 (MCMP Criterion).

$$\pi = \arg \min_{\pi \in \Pi^{MP}} \mathbb{E}[C(\psi(T))]$$

To obtain these policies, this criterion optimizes the occupation measure or state-visitation frequencies through the following Linear Programming formulation:

Definition 3.3.7 (Min-Cost LP).

$$min_x \sum_{s \in \mathcal{S}, a \in \mathcal{A}} x_{s,a} \mathcal{C}(s,a) \qquad \textit{s.t. (C1)–(C6)} \quad \textit{(LP 1)}$$

$$x_{s,a} \geq 0 \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}(s) \quad \textit{(C1)}$$

$$in(s) = \sum_{s' \in \mathcal{S}, a \in \mathcal{A}(s')} x_{s',a} \mathcal{T}(s, a, s') \qquad \forall s \in \mathcal{S} \quad \textit{(C2)}$$

$$out(s) = \sum_{a \in \mathcal{A}(s)} x_{s,a} \qquad \forall s \in \mathcal{S} \setminus \mathcal{G} \quad \textit{(C3)}$$

$$out(s) - in(s) \leq 0 \qquad \forall s \in \mathcal{S} \setminus (\mathcal{G} \cup \{s_0\}) \quad \textit{(C4)}$$

$$out(s_0) - in(s_0) \leq 1 \qquad \textit{(C5)}$$

$$\sum_{s_g \in \mathcal{G}} in(s_g) = p^{\max} \qquad \textit{(C6)}$$

Definition 3.3.8 (Max-Prob LP).

$$max_x \sum_{s_g \in \mathcal{G}} in(s_g) \qquad \textit{s.t. (C1)–(C3), (C7)–(C8)} \quad \textit{(LP 2)}$$

$$out(s) - in(s) = 0 \qquad \forall s \in \mathcal{S} \setminus (\mathcal{G} \cup \{s_0\}) \quad \textit{(C7)}$$

$$out(s_0) - in(s_0) = 1 \qquad \textit{(C8)}$$

Compared to methods such as Finite-Penalty, the MCMP criterion eliminates the drawback in which states are classified as dead ends solely based on their expected cost. In Finite-Penalty approaches, any state whose expected cost of reaching the goal exceeds the predefined penalty D is treated as a dead end, regardless of whether the probability of reaching the goal is still positive. This can lead to misclassifications that improperly prune viable states and degrade the quality of the resulting policy. MCMP avoids this issue by focusing primarily on maximizing the probability of reaching the goal before considering cost, ensuring that states are evaluated based on their goal reaching probability rather than an arbitrary cost threshold.

In addition, as discussed before, the MCMP criterion improves over S^3P by offering a more accurate evaluation of the expected cost in SSP problems with dead ends. The S^3P criterion ranks policies based on expected cost but only considers traces that successfully reach the goal while ignoring those that end in dead ends. This means that when following an S^3P optimal policy in practice, unavoidable dead ends can still be reached and their associated costs are incurred even though they were not reflected in the computed optimal value. As a result, the expected cost experienced when executing an S^3P policy might differ from the cost estimated during planning.

In contrast, MCMP accounts for the full set of possible traces from the initial state, including those leading to dead ends. By first maximizing the probability of reaching the goal and then minimizing the expected cost among the best policies, MCMP ensures that the computed value truly matches what would happen during execution.

3.3.5 Goals with Utility-Based Semantic (GUBS) Criterion

Another criterion is the Goals with Utility-Based Semantic (GUBS) (Freire and Karina Valdivia Delgado, 2017), which allows the decision-maker to define their preferences regarding the trade-off between the probability of reaching a goal and the cost required to achieve it. This is accomplished through a utility function that combines these two objectives.

The utility function is defined by the following equation:

Definition 3.3.9 (GUBS Utility function).

$$U(C_T, B_T) = u(C_T) + K_g \mathbb{1}_{B_t=g} \tag{3.11}$$

Where C_T is the total accumulated cost, B_T indicates whether the goal was reached. Furthermore, we can define the value of a policy π under the GUBS model as the following expected utility:

$$V^\pi = \mathbb{E}[U(C_T, B_T) \mid \pi, s_0] \tag{3.12}$$

Unlike lexicographic criteria, which evaluate objectives in a strict order (maximizing goal probability and then minimizing the costs), GUBS combines these two aspects into a unified scalar through utility-based scalarization (Vamplew *et al.*, 2011). This enables a continuous and flexible representation of preferences, where both goal achievement and cost can be weighed simultaneously according to the user preferences.

By choosing some utility functions such as the exponential (Freire, Karina Valdivia Delgado, and Reis, 2019) and setting the reward K_g for reaching the goal, users can the balance between reaching the goal and cost minimization. This flexibility is particularly valuable in applications where accepting a small probability of failure might be justified by a large reduction in expected cost.

It is important to note that the policies obtained through the GUBS model are non-markovian. This happens because the criteria augments the state space with information about the accumulated cost until each timestep. As a result, the optimal action at a given state depend not only on the current state but also on the history of costs. This differs from the previous criteria that are generally stationary and markovian.

3.4 Solving SSPs, SSPADEs and SSPUDEs

In this section, we discuss solution methods for different classes of SSP problems, including standard SSPs, SSPADEs, and SSPUDEs. We focus on three main approaches: Heuristic Search Methods, Monte-Carlo Tree-Search Planning, and Symbolic Methods. In addition, we describe the fundamental ideas behind each method, discuss their adaptations to handle dead-ends, and highlight their strengths and limitations across the different problem settings.

3.4.1 General Efficient Methods

Basically, the most efficient planning methods to solve MDPs and SSPs are heuristic search, symbolic dynamic programming and Monte Carlo tree-search.

Heuristic Search for SSPs

Kolobov, M. Mausam, *et al.* (2011) states that: *Because VI (PI) stores and updates the value function for the entire S, it can be slow and lack memory for solving even relatively small SSPs. However, thanks to Bellman update convergence to V^* on SSPs (without dead-ends) independently of initialization, VI has given rise to several much more efficient algorithms for this class of problem. An assumption they make is* that the initial state s_0 is known in advance, and one is interested in a policy originating in s_0 only. With this small caveat, it was shown that if the initialization function V_0 (called a heuristic) is admissible, i.e. satisfies $V_0 \leq V^*$ under element-wise comparison, then one can compute V^* for the states relevant to reaching a goal from s_0 without updating or even memoizing values for many of the other states. The Find-and-Revise (F&R) framework (Bonet and Geffner, 2003) generalizes this idea by comprising the algorithms that start with an admissible V_0 and iteratively build the graph of the policy greedy w.r.t. the current V (i.e., the policy derived from V via extracting the greedy action from Bellman equation, finding those of the graph whose values haven't converged and updating them using Bellman update. Since an admissible heuristic, computable on the fly, makes some states look bad a priori and the policy is greedy, they may never end up in the policy graph, making F&R algorithms both speedy and frugal in memory use. Nonetheless, the policies F&R yields under an admissible heuristic are provably optimal upon convergence.

Heuristic search methods focus on solving SSPs by exploring states guided by a heuristic, which helps to direct the search process more efficiently than classical synchronous DP methods. The key advantage of heuristic search over classical DP algorithms, such VI, is that DP updates the values of all states in the state space while heuristic search methods only update the values of the states that are actually visited during the search process, making them more computationally efficient. We call this type of heuristic search as asynchronous DP.

Within this category, several prominent algorithms are commonly used. AO^* (Nilsson, 1982) is a notable heuristic search algorithm that extends the classical A^* algorithm to solve MDPs in acyclic domains. AO^* performs a best-first search to find an optimal solution by propagating values backward from the goal to the initial state, making it particularly suited for problems where the state space is directed and does not contain loops or cycles. Another key method is LAO* (Hansen and Zilberstein, 2001), which is built around AO* but is designed to handle MDPs with cycles or loops in the state space. LAO* incorporates a more sophisticated search strategy that can dynamically adjust its exploration, allowing it to effectively find solutions even when the state space contains feedback loops or cyclic dependencies, a common challenge in many real-world problems.

Real-Time Dynamic Programming (RTDP) (Barto *et al.*, 1995) is another important heuristic search method to solve MDPs and SSPs that combines value iteration with trajectory simulations, called "trials". RTDP uses an admissible heuristic to initialize the

Q-values and then simulates trajectories from the current state to the goal state. As it traverses the state space, RTDP updates the value function based on the actual trajectory it follows, allowing it to efficiently focus on the most relevant parts of the state space, improving performance over standard DP methods in large or complex SSPs.

Find-and-Revise (F&R) (Bonet and Geffner, 2003) is an important framework used to generalize the heuristic search algorithms such as LAO*, ILAO, RTDP and LRTDP, as well FRET and FRET-π. FRET (Find, Revise, Eliminate Traps), an extension of F&R, is capable of efficiently solving MaxProb, with the help of an admissible heuristic. It does so by not only improving the value function with Bellman backups but also by escaping this operator's suboptimal fixed points in a novel way that preserves the value function's admissibility Kolobov, M. Mausam, *et al.*, 2011.

FRET-π (Steinmetz *et al.*, 2016) is a variant of FRET that instead of considering the greedy graph in respect to the value function, it consider only the actions selected in the current greedy policy. The gains of FRET-π over FRET are mostly in terms of compute, because in FRET the greedy graph is built on almost the entire reachable state space, making it a more extensive solution to remove traps. Conversely, in FRET-π, we can eliminate traps only related to the current greedy policy, then, we can find optimal policies while maintaining traps in regions that we are not interested.

Monte-Carlo Tree-Search Planning

Monte-Carlo Planning methods are based on running simulations from the current state to guide the decision-making process. These methods expand a search tree by exploring the next possible states that can be reached from a given action. After expanding the tree to a certain depth or time limit, they update the value function based on the results of the simulations. By simulating multiple possible future trajectories, Monte-Carlo Planning enables the identification of optimal policies without needing to model the entire environment explicitly.

Among the most notable methods in this category is Upper Confidence Bounds applied to Trees (UCT) (Kocsis and Szepesvári, 2006), which has been successfully applied to complex problems such as the game of Go (Gelly and D. Silver, 2011). UCT uses a form of balanced exploration and exploitation by selecting actions that maximize an upper confidence bound on the expected reward. This approach has been particularly effective in domains that require strategic decision-making under uncertainty, where the search space is vast and difficult to explore exhaustively.

Another significant method is Probabilistic Planning Based on UCT (PROST) (Keller and Eyerich, 2012). Building on UCT, PROST enhances Monte-Carlo Planning by incorporating techniques to detect dead-ends and goal states, which helps to focus the search on promising areas of the state space. Additionally, PROST modifies the Q-value initialization and prunes irrelevant actions to improve computational efficiency. It also leverages the RDDL (Relational Dynamic Influence Diagram Language) structure for more efficient planning. PROST has been successfully applied in a variety of planning domains and was the top-performing planner at the International Planning Competition (IPPC) 2011 (Sanner and Yoon, 2011).

Symbolic Dynamic Programming for planning

In general, these methods focus on solving a compact representation of the MDP, known as factored MDPs. Factored MDPs offer significant advantages by reducing memory requirements, improving computational efficiency, and enabling the solution of previously intractable problems. By exploiting the structure of the MDP, these methods allow for more efficient representation and processing of the value function.

Typically, symbolic methods are used to represent and solve factored MDPs. These methods rely on specialized data structures such as Algebraic Decision Diagrams (ADDs) (Bahar *et al.*, 1993), Binary Decision Diagrams (BDDs) (Bryant, 1986), or Extended Algebraic Decision Diagrams (XADDs) (Sanner, Karina Valdivia Delgado, *et al.*, 2011), which provide a compact way to estimate the value function, allowing efficient manipulation and evaluation of large MDPs and SSPs (Karina V. Delgado *et al.*, 2016).

A notable symbolic method for solving factored MDPs is Stochastic Planning using Decision Diagrams (SPUDD) (Hoey *et al.*, 1999). SPUDD extends the Value Iteration (VI) algorithm by using ADDs to represent the value function. This enables the algorithm to handle large state spaces in a memory-efficient manner. By compactly representing the value function, SPUDD can efficiently compute optimal policies for problems with a large number of states.

Similarly, Structured Value Iteration (SVI) (Boutilier *et al.*, 2000) combines the VI algorithm with a decision tree representation of the value function. In this approach, the algorithm performs state aggregation, selecting a subset of features necessary to estimate the value of a state. Over time, the decision tree evolves, and the algorithm focuses on the most relevant aspects of the MDP. This process leads to a more compact and efficient representation of the value function, improving the algorithm's ability to solve the problem.

Another approach, sRTDP (Feng *et al.*, 2002), adapts the Real-Time Dynamic Programming (RTDP) (Barto *et al.*, 1995) algorithm by incorporating ADDs for value function representation. By combining RTDP's trajectory-based updates with the compactness of ADDs, sRTDP can efficiently solve large MDPs, especially when the problem includes a significant amount of stochasticity or uncertain outcomes. In addition, there is Symbolic Bounded RTDP (sBRTDP) (Karina Valdivia Delgado *et al.*, 2010) which is a variant of sRTDP that add a upper and lower bound to the value function and is more adequate in scenarios that the transition probability matrix is dense, in contrast to the classical RTDP.

3.4.2 Planning algorithms for IHD-MDP, SSP, SSPADE and SSPUDE

IHD-MDP

To solve an Infinite Horizon Discounted MDP (Definition 2.2.1), we can use classical solutions such as DP (VI, PI, TVI), LP, Monte-Carlo Tree-Search Planning (PROST). The main challenge of this solutions is how to choose an appropriate discount factor to guarantee that the value function is bounded and we can adjust our preference over short-term and long-term rewards/costs.

SSP without dead-ends

To solve an SSP without dead-ends according with Definition 3.1.2, we are interested in policies that are guaranteed to arrive the goal state with minimum cost. To solve problems formulated as an SSP, we can use clasical DP algorithms such as VI and PI, Heuristic Search methods such as RTDP, LRTDP, LAO*, ILAO*, and Monte-Carlo Tree Search methods such as UCT and PROST. However, most of these algorithms assume that the SSP does not contain any dead-end states, and if there exists a dead-end state, they may break.

SSPADE

In the case of a SSPADE, we do have dead-ends, but they are avoidable. Thus, there is a proper policy that can reach the goal by avoiding the dead-ends, but, even classical DP algorithms such as VI diverge in SSPADEs. This happens because DP algorithms evaluate the values of all states, and as the dead-end state does not reach the goal, the value of a dead-end state is infinite, then, the Bellman equation becomes unbounded.

Nevertheless, we can modify the Bellman update of the VI algorithm to truncate the value function using the Finite Penalty criterion. By using this criterion, the maximum value of the value function will be the penalty D, and even as the value of the dead-end state goes to infinity, the truncated value function of the dead-end will be set to D. In consequence, the Bellman equation will be bounded and the algorithm will converge.

Similarly, not every heuristic search algorithm that solves an SSP is able to solve SSPADEs. These include LAO*, which uses PI to evaluate the policy, and the algorithm never terminates the evaluation phase because the dead-end states have infinite value. The same issue arises with RTDP and LRTDP, once they enter a dead-end state, they stay there forever, and consequently, the algorithm does not converge.

On the other hand, we can still solve SSPADEs using heuristic search methods such as Improved LAO* (ILAO*), without any modification. ILAO* works as a "focused VI" algorithm because instead of updating all states, it only updates the states that are in the best partial solution graph (Hansen and Zilberstein, 2001). The main difference between ILAO* and LAO*, is that ILAO* limits the number of iterations of the VI/PI steps, and while dead-end states have infinite cost, the number of iterations limits the value of dead-end states. Thus, as dead-end states accumulate more cost than the other states, they will be removed from the solution graph, and the solution will be a policy that reaches the goal from the initial state s_0.

Furthermore, despite not working "out of the box", it is important to note that LAO*, RTDP, and LRTDP can be modified to solve SSPADEs. The main idea is to modify the termination condition, by introducing a fixed number of iterations or limiting the number of timesteps (Mausam and Kolobov, 2012).

fSSPUDE

In the case of SSPUDE, the issue is even worse, because all policies that reach the goal have a positive probability of going into a dead-end state. Thus, as in SSPADEs, some classical DP algorithms such as VI don't work in an SSPUDE because the value of dead-end states is infinite, following the definition of SSPs that improper policies have infinite cost. Similarly, we can assume that while reaching a dead-end is not ideal, it carries a finite price. We call this type of SSPUDE as Finite-Penalty SSPUDE (fSSPUDE). In these problems, we can modify the VI to use the Finite Penalty criterion through a penalty D, which sets a maximum value to the dead-end state (Mausam and Kolobov, 2012).

iSSPUDE

In iSSPUDEs, the issue is even worse, because even modifying the classical VI with the Finite-Penalty criterion isn't sufficient, because the penalty of going into a dead-end is infinite. In that case, the original optimization problem of minimizing the costs to reach the goal becomes ill-defined. This problem affects all previous algorithms that can solve SSP and SSPADE, and to handle this problem, we need to solve it in two steps: 1. obtain a policy that maximizes the probability of reaching the goal, and 2. minimize the costs among the policies that maximize the goal-probability. Using this new formulation, we can solve iSSPUDEs using algorithms such as Infinite-Penalty Value Iteration (IVI), Staged Heuristic Search (SHS) (Kolobov and Weld, 2012), Goal-Probability and Cost-Iteration (GPCI) (Teichteil-Königsbuch, 2012), and Minimum Cost and Maximum Probability (MCMP) (Trevizan *et al.*, 2017). As the first step involves solving a MaxProb problem, it may encounter zero-reward cycles that forms a trap. Thus, these solutions rely on the FRET or FRET-π framework to remove traps.

Model	**Assumptions**	**Classical Solutions**	**Output**
Infinite Horizon Discounted MDP	Infinite number of time-steps to go; discount factor $\gamma < 1$ (to ensure convergence).	DP (VI, PI), LP, Heuristic Search (RTDP, LRTDP, ILAO*), Monte-Carlo Tree-Search Planning.	Stationary policy that optimizes the expected discounted total reward/cost over an infinite horizon.
SSP	Indefinite horizon; Existence of at least one proper policy; The value of improper policies is infinite; Goal state is absorbing with zero cost; Costs are strict positive, except on the goal state.	DP (VI, PI), LP, Heuristic Search (RTDP, LRTDP, ILAO*), Monte-Carlo Tree-Search Planning (PROST).	Stationary policy that minimizes the expected total cost towards a goal state.
SSP with Avoidable Dead-Ends (SSPADE)	Existence of avoidable Dead-ends states; The goal state can be reachable from the initial state s_0; The value of improper policy has infinite cost.	Heuristic Search (ILAO*), DP (Finite-penalty VI).	Stationary policy that minimizes the expected total cost with probability 1 towards the goal, rooted in the initial state s_0.
SSP with Unavoidable Dead-Ends and Finite-Penalty (fSSPUDE)	Dead-ends are unavoidable, and even the policies that reaches the goal have a non-zero probability to reach a dead-end; expected cost may be infinite.	Heuristic Search, Modified Dynamic Programming (Finite penalty criterion), MAXPROB, Dual optimization criterion (S^{3}P, MCMP).	Stationary policy that minimizes the expected total over trajectories with maximum probability towards the goal state, rooted in the initial state s_0.
Maximum Probability MDP (MAXPROB)	Sparse-reward problem, all rewards are zero, except on the goal state that is 1. It is not an SSP because improper policies does not accumulate infinite cost. Can contain zero-reward cycles called dead-end traps.	Heuristic Search and Dynamic Programming (VI_{MP}) using FRET or FRET-π.	Stationary policy with maximum probability towards the goal state, rooted in s_0.
SSP with Unavoidable Dead-Ends and Infinite-Penalty (iSSPUDE)	Dead-ends are unavoidable, and even the policies that reaches the goal have a non-zero probability to reach a dead-end; expected cost may be infinite.	Heuristic Search, Modified Dynamic Programming (Finite penalty criterion), MAXPROB, Dual optimization criterion (S^{3}P, MCMP).	Stationary policy that minimizes the expected total over trajectories with maximum probability towards the goal state, rooted in the initial state s_0.

Table 3.1: *Summary of assumptions and planning solutions for sequential decision making models.*

Chapter 4

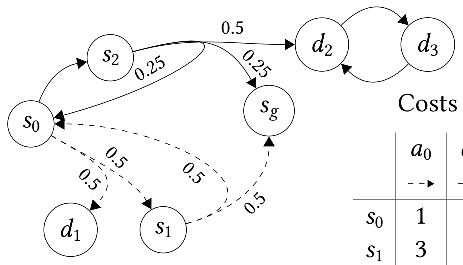
Comparing the dual optimization criteria S^{3}P, MCMP with the IHD-MDP criterion

In this chapter we extend the example of the dual optimization criteria, S^{3}P and MCMP, introduced by Trevizan *et al.*, 2017 to include the IHD-MDP criterion in this illustration. The idea is to show that by: (i) considering only traces that reach the goal ignoring the traces that reaches dead ends; (ii) considering traces that reach the goal and the the traces that stop in the first dead-end; and (iii) considering all traces, including the ones that enter in dead-end traps scan lead to unexpected optimal policies.

4.1 An SSP example with two MaxProb policies

To show how the two dual optimization criteria for SSPUDEs, S^{3}P and MCMP, can end up choosing different optimal policies, Trevizan *et al.*, 2017 defined an SSP example with two policies, π_0 and π_1, depicted in Fig. 4.1. Notice that in the figure, the different type of arrows, dashed line and solid line, are used to represent policies π_0 and π_1, respectively; and the table on the right defines the SSP cost function $C(S, A)$.

Example 1. *Consider the SSP in Fig. 4.1. There are two possible policies in this SSP: π_0 that always applies action a_0 and π_1 that always applies action a_1.*



Costs

	a_0	a_1
	$\dashrightarrow$	$\rightarrow$
s_0	1	1
s_1	3	–
s_2	–	2

Figure 4.1: *Example of SSP with two policies. The goal is $G = \{s_g\}$. The initial state is s_0 and states d_1, d_2 and d_3 are dead ends. (extracted from (Trevizan et al., 2017)).*

In the original work (Trevizan *et al.*, 2017) this example did not include the IHD criterion. However, to show how it can be compared with the dual opitimization criteria, S^{3}P and MCMP, we use this example to include the IHD criterion analysis. Note that following the policy π_0, each of the three criteria consider the traces bellow:

- S^3P criterion: $s_0 s_1 s_g$ and $s_0 s_1 s_0 s_1 s_0 \ldots$
- MCMP criterion: $s_0 d_1$, $s_0 s_1 s_g$ and $s_0 s_1 s_0 s_1 s_0 \ldots$
- IHD criterion: $s_0 d_1$, $s_0 s_1 s_g$ and $s_0 s_1 s_0 s_1 s_0 \ldots$

and when following the policy π_1, the three criterion considers the traces:

- S^3P criterion: $s_0 s_2 s_g$ and $s_0 s_2 s_0 s_2 s_0 \ldots$
- MCMP criterion: $s_0 s_2 s_g$, $s_0 s_2 s_0 s_2 s_0 \ldots$ and $s_0 s_2 d_2$
- IHD criterion: $s_0 s_2 s_g$, $s_0 s_2 s_0 s_2 s_0 \ldots$ and $s_0 s_2 d_2 d_3 d_2 d_3 \ldots$

Note that for the policy π_0 both criteria MCMP and IHD consider the same traces; However for policy π_1, the IHD criterion considers a different trace by following the trap $d2d3d2d3 \ldots$ (infinite loop), which must be acumulate costs with the given discount.

Since S^{3}P and MCMP rely on the MaxProb criteria, we start by computing the probability of reaching the goal from the initial state s_0.

4.1.1 Computing $P^*(s_0)$

Recall that the probability of reaching the goal state s_g from the initial state s_0, following any policy π, is denoted by $P^\pi(s_0)$. This probability can be computed from the set of all traces of π from s_0 that reach s_g, denoted by $P(T^G_{s_0,\pi})$. Thus, we can compute the expected probability to the goal, as follows:

$$P^\pi(s_0) = P(T^G_{s_0,\pi}) = \sum_{T^G_{\pi,s_0}} \prod_{i=1}^{|T|} \mathcal{T}(s_i, \pi(s_i), s_{i+1}). \tag{4.1}$$

In the example of Figure 4.1, Equation 4.1 can be used to compute the probability to the goal from s_0 when following policy π_0, resulting in $P^{\pi_0}(s_0) = 1/3$ and when following policy π_1, resulting in $P^{\pi_1}(s_0) = 1/3$, showing that both policies have the same probability to the goal.

4.1.2 Computing $V^{\pi_0}(s_0)$ and $V^{\pi_1}(s_0)$ with the S^{3}P criterion

Under the S^{3}P criterion, only traces that reach the goal state are considered. Specifically, when following policy π_0, the relevant traces are those of the form $(s_0 s_1)^+ s_g$, representing repeated visits between states s_0 and s_1 before eventually reaching the goal s_g. Similarly, for policy π_1, only traces of the form $(s_0 s_2)^+ s_g$ are taken into account. The S^{3}P value function satisfies the following recursive equation (Trevizan *et al.*, 2017):

$$V^{\pi}_{S^3P}(s_0) = \frac{\sum_{s' \in \mathcal{S}} \mathcal{T}(s_0, \pi(s_0), s')\, P(T^G_{\pi,s'}) \left[C(s_0, \pi(s_0)) + V^{\pi}_{S^3P}(s')\right]}{P(T^G_{\pi,s_0})} \tag{4.2}$$

Here, $\mathcal{T}(s, a, s')$ is the transition probability from state s to s' under action a, $C(s, a)$ is the immediate cost, and $P(T^G_{\pi,s})$ denotes the probability that a trace starting from state s under policy π reaches the goal. Thus, the expected costs for policies π_0 and π_1 are computed as follows:

$$V^{\pi_0}_{S^3P}(s_0) = \frac{\sum_{i=0}^{\infty}(0.5 \cdot 0.5)^i (1+3) \cdot i}{p^{\max}} = \boxed{\frac{16}{3}}$$

$$V^{\pi_1}_{S^3P}(s_0) = \frac{\sum_{i=0}^{\infty}(1.0 \cdot 0.25)^i (1+2) \cdot i}{p^{\max}} = \boxed{4}$$

4.1.3 Computing $V^{\pi_0}(s_0)$ and $V^{\pi_1}(s_0)$ with the MCMP criterion

Under the MCMP criterion, all traces are finite because, even in the presence of dead ends, it applies a truncation function ψ that terminates each trace T at the first occurrence of a dead end. As a result, the trace generated from the initial state s_0 under policy π_0, denoted T_{π_0,s_0}, is finite; The same holds for the trace generated under policy π_1, T_{π_1,s_0}, which may enter the cycle $d_2 d_3 d_2 \dots$. By applying the truncation function, this infinite sequence is reduced to a finite one: $\psi(s_0 \dots s_2 d_2 d_3 d_2 \dots) = s_0 \dots s_2 d_2$. Another important point is that the MCMP criterion relies on a dual linear programming (LP) formulation, which estimates the value function using the state visitation frequencies, also known as occupancy measures (x_s). Therefore, the expected cost under a given policy π is then computed as:

$$V^{\pi}_{\mathrm{MCMP}}(s_0) = \sum_{s \in \mathcal{S}} x_s C(s, \pi(s)),$$

where x_s denotes the expected number of times state s is visited, and $C(s, a)$ is the immediate cost of taking action a in state s. Thus, the expected costs of π_0 and π_1 under the MCMP criterion can be computed as follows:

$$V^{\pi_0}_{MCMP}(s_0) = \frac{4}{3} + 3 \cdot \frac{2}{3} = \boxed{\frac{10}{3}}$$
$$V^{\pi_1}_{MCMP}(s_0) = \frac{4}{3}(1+2) = \boxed{4}$$

4.1.4 Computing $V^{\pi_0}(s_0)$ and $V^{\pi_1}(s_0)$ with the IHD criterion

Under the IHD criterion, the value function satisfies the Bellman equation, which recursively defines the expected cost of a state under a given policy. Specifically, for a discount factor $\gamma \in [0,1)$ and a fixed policy π, the value function $V^\pi(s)$ represents the expected cumulative cost starting from state s and is given by:

$$V^\pi(s) = C(s,\pi(s)) + \gamma \sum_{s'} \mathcal{T}(s,\pi(s),s')V^\pi(s'),$$

where $C(s,a)$ denotes the immediate cost of taking action a in state s, and $\mathcal{T}(s,a,s')$ is the probability of transitioning to state s' after taking action a in state s. This formulation ensures that future costs are geometrically discounted, giving more weight to immediate outcomes and guaranteeing convergence of the value function even in the presence of infinite trajectories. Thus, the expected costs of π_0 and π_1 under the IHD criterion, considering $V(d_1) = V(d_2) = V(d_3) = 0$ (i.e. any action applied to those dead-ends have 0 costs) can be computed as follows:

$$V^{\pi_0}_{IHD_{\gamma=0.99}}(s_0) = \frac{1+1.5\gamma}{(1-0.25\gamma^2)} = \frac{1+1.5\cdot 0.99}{(1-0.25\cdot 0.99^2)} = \boxed{3.29}$$
$$V^{\pi_1}_{IHD_{\gamma=0.99}}(s_0) = \frac{1+2\gamma}{(1-0.25\gamma^2)} = \frac{1+2\cdot 0.99}{(1-0.25\cdot 0.99^2)} = \boxed{3.95}$$

4.1.5 Comparison of the three criteria for Example 1

	S^3P	MCMP	$IHD_{\gamma=0.99}$	
	$V^\pi(s_0)$	$V^\pi(s_0)$	$V^\pi(s_0)$	$P^\pi(s_0)$
π_0	5.33	**3.33**	**3.29**	**1/3**
π_1	**4.00**	4.00	3.95	**1/3**

Table 4.1: *Value function, $V^\pi(s_0)$, and probability to goal, $P^*(s_0)$, computed for the policies π_0 and π_1, considering the three optimization criterion S^3P, MCMP and IHD.*

Table 4.1 summarizes the computation of Value function ($V^\pi(s_0)$) and max probability to goal $P^*(s_0)$ for s_0 computed for the policies π_0 and π_1, considering the three optimization criterion S³P, MCMP and IHD with $\gamma = 0.9$. Notice that both policies have the same MaxProb value for s_0. As a result, while S³P chooses π_1 as an optimal policy, MCMP and

IHD choose π_0, however, they compute different values for $V^{\pi}(s_0)$ due to the discount factor of $\gamma = 0.99$.

Furthermore, each criteria offers a different perspective on evaluating policies in the presence of dead ends. The S^{3}P criterion focuses exclusively on goal reaching trajectories, discarding any trace that fails to reach the goal. In contrast, the MCMP criterion considers all traces but applies a truncation function that terminates execution upon encountering a dead end, ensuring all traces are finite. The IHD criterion, based on total discounted cost, also includes all traces, successful or not, but reduces the impact of future costs using a discount factor γ. Although MCMP and IHD differ in technical formulation, they may exhibit similar behavior. An intuitive interpretation of the discount factor γ in the IHD criterion is that it introduces an implicit probability of termination at each time step. Specifically, γ can be seen as the probability of continuing to the next step, while $1 - \gamma$ represents the probability of stopping. Thus, the IHD criterion truncates the infinite horizon, making it similar to the truncation function of MCMP.

In the next section, we relax the previous assumption of assigning zero cost to dead ends under the IHD criterion. Specifically, we introduce positive costs for the dead-end cycle $d2d3d2\ldots$ to obtain a more accurate measure of the IHD criterion under dead end traps.

If we modify the Example 1, to add cost 2 for the actions applied in dead-ends d_2 and d_3, i.e, the action a_1 now has cost 2 when applied to d_2 and d_3, the IHD would still choose policy π_0. This is shown with the following computation:

$$V^{\pi_0}_{IHD_{\gamma=0.99}}(s_0) = \frac{1 + 1.5\gamma + 0.5\gamma \cdot d}{(1 - 0.25\gamma^2)} = \frac{1 + 1.5 \cdot 0.99 + 0.5 \cdot 0.99 \cdot 0}{(1 - 0.25 \cdot 0.99^2)} = \boxed{3.29}$$

$$V^{\pi_1}_{IHD_{\gamma=0.99}}(s_0) = \frac{1 + 2\gamma + 0.5\gamma^2(\frac{d}{(1-\gamma)})}{(1 - 0.25\gamma^2)} = \frac{1 + 2 \cdot 0.99 + 0.5 \cdot 0.99^2(\frac{2}{(1-0.99)})}{(1 - 0.25 \cdot 0.99^2)} = \boxed{133.76}$$

	S^3P	MCMP	$IHD_{\gamma=0.99}$	
	$V^*(s_0)$	$V^*(s_0)$	$V^*(s_0)$	$P^{\pi}(s_0)$
π_0	5.33	**3.33**	**3.29**	**1/3**
π_1	**4.00**	4.00	133.76	**1/3**

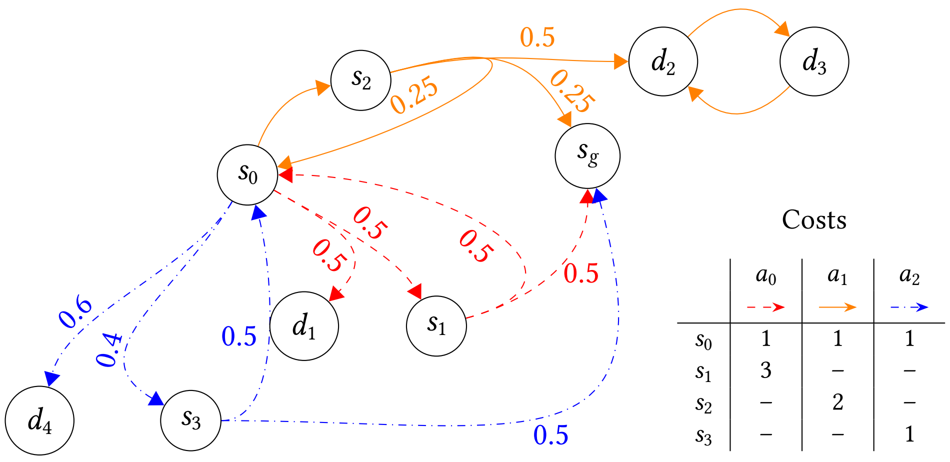
Table 4.2: *Updated values for Example 1 after adding cost 2 to the action a_1 when applied to dead-ends d_2 and d_3.*

Table 4.2 updates the computed values, showing that IHD still chooses policy π_0, even when we add cost to dead-ends. Note that this is done based on a significantly overestimated cost for π_1. The computed value of $V^{\pi_1}(s_0)$ increases to 133.76. This inflation is caused by the discounted sum when a policy can lead to a costly, absorbing dead-end. The term $(\frac{d}{1-\gamma})$ in the value function becomes very large as γ approaches 1, representing the accumulated discounted cost of being trapped in the dead-end d_2.

4.2 The SSP example with an extra policy π_2

In the Example 2 we add a new policy π_2 that has less cost to reach the goal, but also has less probability to reach the goal when we compare with policies (π_0 and π_1).

Example 2. *Consider the SSP in Fig. 4.2. There are three possible policies in this SSP: π_0 that always applies a_0, π_1 that always applies a_1 and π_2 that always applies a_2.*



Costs

	a_0	a_1	a_2
s_0	1	1	1
s_1	3	–	–
s_2	–	2	–
s_3	–	–	1

Figure 4.2: *Example of SSP with three policies, adapted from (Trevizan et al., 2017) by including an extra action a_2 and policy π_2.*

Let us analyze the behavior of the three different optimization criteria on Example 4.2, The policy π_2, which consists of always choosing action a_2, is intentionally designed to present a distinct trade-off. It offers a path to the goal with less cost compared to the other available policies, π_0 and π_1. However, this cost efficiency comes at the expense of the goal-probability, as the probability of successfully reaching the goal under policy π_2 is lower than the probabilities offered by both π_0 and π_1. This creates a clear dilemma for the decision-making process and provides a critical test case for comparing how different criteria behaves in this scenario.

Recalling that the probability to reach the goal must be computed from the set of all traces of π_2 from s_0 that reach s_g:

$$P^{\pi_2}(s_0) = P(T^G_{s_0,\pi_2}) = \sum_{T \in T^G_{\pi_2,s_0}} \prod_{i=1}^{|T|} \mathcal{T}(s_i, \pi(s_i), s_{i+1}) \tag{4.3}$$

$$P(T^G_{s_0,\pi_2}) = \mathcal{T}(s_0, a_2, s_3)[(\mathcal{T}(s_3, a_2, s_g) + \mathcal{T}(s_2, a_2, s_0)P(T^G_{s_0,\pi_2}))]$$

$$P(T^G_{s_0,\pi_2}) = 0.4(0.5 + 0.5P(T^G_{s_0,\pi_2}))$$
$$P(T^G_{s_0,\pi_2}) = \frac{0.2}{0.8} = \boxed{\frac{1}{4}}$$

Since $P^{\pi_2}(s) = 1/4$ which is less than $P^{\pi_0}(s)$ and $P^{\pi_1}(s)$, π_2 is not in the set of MaxProb policies Π^{MP} and therefore would not be selected by the dual optimization criteria. However, the IHD criterion will select policy π_2 since it has the lowest value from s_0:

$$V^{\pi_2}_{IHD_{\gamma=0.99}}(s_0) = \frac{1 + 0.4\gamma}{(1 - 0.2\gamma^2)} = \frac{1 + 0.4 \cdot 0.99}{(1 - 0.2 \cdot 0.99^2)} = \boxed{1.74}$$

	S^3P	MCMP	IHD	
	$V^*(s_0)$	$V^*(s_0)$	$V^*(s_0)$	$P^*(s_0)$
π_0	5.33	**3.33**	3.29	**1/3**
π_1	**4.00**	4.00	3.95	**1/3**
π_2	–	–	**1.74**	**1/4**

Table 4.3: *Optimal value function for s_0 ($V^*(s_0)$) computed for policies π_0, π_1 and π_2, considering the three optimization criterion S^3P, MCMP and IHD.*

Table 4.3 summarizes the analysis of Example 2, presenting the optimal value function $V^*(s_0)$ computed for the three policies (π_0, π_1, and π_2) under the S^{3}P, MCMP, and IHD criteria. Additionally, the table reports the maximum probability of reaching the goal $P^{\pi}(s_0)$ for each policy. We observe that policies π_0 and π_1 share the same maximum success probability of $P^*(s_0) = \frac{1}{3}$, whereas π_2 has a lower success probability of $\frac{1}{4}$, indicating that it is not a solution for the dual optimization criterion.

In contrast, the IHD criterion evaluates all policies regardless of their success probabilities. It selects π_2 as optimal, since it achieves the lowest expected cost (1.74) among all candidates. However, this comes at the expense of a reduced probability of success. This highlights a key limitation of the IHD criterion in the context of SSPs, where the primary objective is often to reach the goal reliably. Thus, IHD may produce solutions that are cost-efficient in expectation, but undesirable when the probability of reaching the goal is critical.

Chapter 5

Reinforcement Learning for Stochastic Shortest Paths with Dead-Ends

This chapter presents the main contribution of this work: the use of reinforcement learning (Sutton and Barto, 2018) for solving SSPs, SSPADEs and SSPUDEs, inspired by the results from automated planning research.

5.1 The Q-Learning algorithm: a brief review

In this section, we give a brief description of the main Reinforcement Learning concepts and the Q-Learning algorithm (Watkins and Dayan, 1992).

In contrast to planning algorithms, where the transition model is assumed to be known, RL does not assume knowledge about the transition, thus, it relies on learning from real-world experiences or simulations of the environment.

One of the main algorithms of RL is Q-learning and, in general, it is commonly applied on IHD-MDP problems, i.e. it assumes an IHD-MDP model. Q-Learning is a model-free, off-policy reinforcement learning algorithm used to *find the optimal action-selection policy for an IHD-MDP that maximizes the expected cumulative discounted reward.* The core idea of Q-Learning is to iteratively update the Q-values, which represent the expected cumulative reward for taking a particular action a in a specific state s, and following the optimal policy thereafter in an infinite discounted horizon. These Q-values are updated using the regular Bellman equation, with the update rule as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]. \tag{5.1}$$

The main ingredients of the regular Q-learning update rule are: α is the learning rate, γ is the discount factor, and r is the immediate reward received after taking action a in state s and transitioning to the next state s'. This update is performed iteratively as the agent explores the environment, gradually refining its Q-values over the complete space of pairs (s,a), until convergence to optimal values and consequently optimal policy.

Algorithm 3 Q-Learning-IHD Algorithm

```
Require: IHD-MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, epsilon $\epsilon$, learning rate $\alpha$, discount factor $\gamma$
Ensure: $\pi(s) \approx \pi^*(s)$,    $Q^\pi(s,a) \approx Q^*(s,a)$ for all $s \in S$ and $a \in A$
 1: Initialize $Q(s,a) \leftarrow 0$, for all $s \in S$ and $a \in A$
 2: $t \leftarrow 0$
 3: repeat
 4:     $s \leftarrow s_0$                                    ▷ Initial state
 5:     $h \leftarrow 0$
 6:     repeat
 7:         Choose action $a$ in $s$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
 8:         Take action $a$, observe reward $r$ and next state $s'$
 9:         $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \max_{a'} Q(s',a') - Q(s,a)]$
10:         $s \leftarrow s'$                             ▷ Transition to next state
11:         $t \leftarrow t + 1$
12:         $h \leftarrow h + 1$
13:     until $s$ is terminal or $h \geq$ max horizon
14: until $t \geq$ max timesteps
15: $\pi(s) \leftarrow \arg\max_a Q(s,a)$, for all $s \in \mathcal{S}$
16: return $\pi, Q$
```

Exploration versus Exploitation. One of the key features of Q-Learning-IHD (Algorithm 3) is its off-policy nature, meaning the agent learns the optimal policy independently of the policy it is currently following (Sutton and Barto, 2018) on the generation of episodes. This is achieved through the use of the greedy action selection mechanism during updates, where the algorithm always updates based on the maximum possible Q-value for the next state, regardless of the current exploration strategy. Typically, an ϵ-greedy strategy is employed, where the agent **explores** the environment randomly with probability ϵ and **exploits** the learned Q-values with probability $1 - \epsilon$. Over time, if we use an ϵ-greedy decay parameter, the exploration decreases and the agent converges towards the greedy-optimal policy. This idea aligns with the concept of Greedy in the Limit with Infinite Exploration (GLIE) (Singh *et al.*, 2000; Russell *et al.*, 2010). Despite its simplicity, Q-Learning has proven to be a powerful tool for reinforcement learning, capable of solving a wide range of problems without requiring a model of the environment.

Q-Learning is guaranteed to converge to the optimal policy under certain conditions, specifically when each state-action pair is visited infinitely often (J. N. Tsitsiklis, 1994). This convergence is assured even when the agent follows a suboptimal policy during the learning process, as long as the exploration ensures sufficient state-action pair visits. As the Q-values are updated through repeated iterations, the algorithm asymptotically converges to the optimal Q-values, from which the optimal policy can be derived. However, practical convergence may be slower in environments with large state spaces or high noise, where fine-tuning of the learning rate and exploration parameters is required for optimal performance.

Although ϵ-greedy is one of the most common ways to explore in Q-learning, it is not the only method. A variety of alternative exploration methods (Thrun, 1992; Amin *et al.*,

2021) have been proposed to both accelerate learning and provide theoretical guarantees of convergence to the optimal policy. These guarantees are often formalized through the concept of sample complexity (Kakade, 2003), which measures the amount of experience an algorithm requires to successfully solve a task. Similarly, algorithms can be analyzed under the Provably Approximately Correct in Markov Decision Processes (PAC-MDP) framework (Alexander L Strehl *et al.*, 2006; Alexander L. Strehl *et al.*, 2009), which aims to prove that an algorithm makes only a polynomial number of suboptimal decisions with high probability.

Online versus Offline learning. An important distinction is that Q-learning can be either used online or offline. In the case of online, the algorithm interacts with the environment or a simulator to obtain experiences. Despite being off-policy, the actions that Q-learning chooses influences new samples that are collected during the learning process. In contrast, in offline settings, we start with a set of experiences that were previously collected with a policy that may be unknown to the agent (Wiering and Otterlo, 2012; Levine *et al.*, 2020). Then, by sampling these experiences, the algorithm learns the state-action values only with respect to this set of experiences. This idea is called experience-replay (Lin, 1992) and is used in different online or offline algorithms, such as Deep Q-Networks (DQN) Mnih *et al.*, 2015. We can use the replay buffer to sample past experiences (online or offline) to estimate Q(s,a).

5.2 Reinforcement Learning for Stochastic Shortest Paths: a study based on probabilistic planning

As we have discussed before, most of the RL literature assumes an IHD-MDP model, while SSPs have received comparatively little attention from the RL community. A common assumption is that IHD-MDP can model any type of sequential decision-making problem. However, as noted by D. Bertsekas, 2023 and discussed in Section 1.2, this assumption is questionable for the following reasons:

- The discount factor is generally unknown in advance, and its value can significantly affect the nature of the optimal policy;
- Using the IHD-MDP as a way to describe an SSP problem can be problematic. In particular, by assuming an IHD-MDP, the agent may either prefer to remain in cycles indefinitely instead of reaching the goal, which fundamentally changes the problem being solved; or choose policies with the shortest path without to prioritize the probability to the goal, which is not a safe choice.

As showed in Chapter 3, SSPs have received significant attention in the field of automated probabilistic planning. This highlights a notable gap: while SSPs are well understood in probabilistic planning with known models, their potential in reinforcement learning has not been fully explored. Then, to start this exploration, we focus our investigation into using Q-learning to solve SSPs, SSPADEs and SSPUDEs.

5.2.1 RL for SSPs without Dead-Ends

Let us start considering an SSP as in Definition 3.1.2. Unlike traditional RL settings where we aim to maximize cumulative expected reward, SSPs are cost-based. Therefore, the objective of RL for SSPs becomes to *minimize the expected total cost to reach the goal state.*

In the context of **SSPs without dead-ends**, Q-learning converges if we make the common assumptions of an SSP, such as there is at least one proper policy and that improper policies have infinite cost. When these conditions are satisfied, Q-learning can be applied directly to solve SSPs (J. N. Tsitsiklis, 1994). Therefore, by introducing goal states, guaranteeing there are no dead-ends, the Q-learning update rule is adapted accordingly and defined as follows:

Q-Learning update rule for SSPs without dead-ends (or SSPADEs)

$$Q(s,a) \leftarrow \begin{cases} 0, & \text{if } s \in \mathcal{G}, \\ Q(s,a) + \alpha \left[c + \min_{a'} Q(s',a') - Q(s,a)\right], & \text{otherwise.} \end{cases} \tag{5.2}$$

Note that Equation 5.2 differs from the update rule 5.1 as follows: (i) does not include a discount factor; (ii) operates with costs, instead of rewards; (iii) minimizes the expected cumulative cost, instead of maximizing the expected cumulative reward; and (iv) initializes the action-state value function of goal states with 0. It is easy to prove that, with the update 5.2, Q-learning will converge to an approximated optimal policy for a given SSPs of Definition 3.1.2, with no need of including a discount factor.

However, in many practical scenarios, the SSP assumptions from Definition 3.1.2 may be violated. One such case is when dead-end states are present, and it is impossible to reach the goal with probability 1 neither all improper policies have an infinite cost. In these situations, the problem become more complex because we no longer have guarantees that all exploration paths will eventually reach the goal neither have a minimal cost. As a result, RL convergence to an optimal or even proper policy becomes more difficult, and the learning process may require modifications to remain effective.

5.2.2 RL for SSPs with Avoidable Dead-Ends

Let us consider Definition 3.2.2 for the case of SSPs with avoidable dead-ends (SSPADEs) where *a proper policy exists that avoids all dead-ends when starting from the initial state s_0 and all improper policies have an infinite cost.* In this case, Q-learning with the update rule 5.2 remains applicable without any modification.

The presence of dead-end states introduces difficulties during exploration in RL, as the agent goes into a dead-end state and it stays there until the episode is finished. This issue can be mitigated if the agent collects a sufficiently large and diverse set of experiences. Trajectories that goes into nonterminal dead-ends can accumulate costs, which gradually increase the Q-values of actions leading to those states. As a result, the agent learns to prefer actions that lead toward the goal. Thus, with sufficient exploration and appropriate learning rate schedules, Q-learning using the update rule 5.2 converges to an optimal proper policy rooted on s_0, which avoids dead-ends.

In this case, the update rule 5.2 will return a policy rooted in s_0 that is capable of reaching the goal with probability 1 while avoiding all dead-ends with approximated minimum expected accumulated cost.

We can also try to convert an SSPADE of Definition 3.2.2 into an infinite-horizon discounted MDP but rooted in s_0. If Assumption 3 holds, i.e. there is a policy that reaches the goal with probability 1 from s_0, and if we choose an appropriate discount factor for it, the optimal policy for Equation 3.3 will also be the solution for the converted infinite-horizon discounted MDP. However, since the optimal policy for SSPADEs cannot reach dead-end states from s_0, the task of choosing the discount factor is more challenging than in a SSP of Definition 3.1.2. Thus, the advantage of using the update rule 5.2 for SSPs without dead-ends and SSPADEs is that it does not include the discount factor γ and still guarantees convergence.

5.2.3 RL for SSPs with Unavoidable Dead-Ends

Let us consider Definition 3.2.3. In an *SSPUDE*, since Assumptions 1, 2 and 3 do not hold, there exists a set of unavoidable dead-end states $S_{DE} \subset S \setminus G$. Unlike *SSPADE*s, no policy $\pi : S \rightarrow A$ can prevent reaching certain states in S_{DE} from s_0. Thus, a proper policy for SSPUDEs has positive probability to reach the goal while an improper policy has probability 0 to reach the goal. The challenge of solving an *SSPUDE* is to find a policy π that minimizes the expected cumulative cost while ensuring that the probability of reaching a goal state $s_g \in G$ from s_0 is maximized, even when some paths inevitably lead to dead-ends.

To address the problems introduced by unavoidable dead-end states, several authors of probabilistic planning proposed different optimization criteria. Also, these criteria are able to solve SSPUDEs even if the set of dead-end states is unknown. In this section we discuss how to use RL to solve SSPUDEs using the finite-penalty criterion (fSSPUDE), the MaxProb criterion and the lexicographic criteria: S^{3}P and the MCMP.

Q-learning for fSSPUDE

Given an SSPUDE according with Definition 3.2.3, the probabilistic planning criterion called Finite Penalty (Kolobov and Weld, 2012), assumes that going into a dead-end, despite undesirable, has a finite cost. This modification guarantees that the value function will be bounded in any policy. In RL, a finite penalty can be assigned to dead-ends during the learning process. Thus, we can adapt the Q-Learning update rule for SSPUDE as:

Q-Learning update rule for SSPUDE with the Finite-Penalty Criterion

$$Q^{FP}(s,a) \leftarrow \begin{cases} 0, & \text{if } s \in \mathcal{G}, \\ \min\{D, Q^{FP}(s,a) + \alpha[C + \min_{a'} Q^{FP}(s',a') - Q^{FP}(s,a)]\}, & \text{otherwise.} \end{cases} \tag{5.3}$$

where D is a penalty value. This ensures that Q-Learning can handle cases where dead-ends are unavoidable by applying a penalty D to any state leading to them. Note that D

defines a bound over the action value function $Q^{FP}(s, a)$, thus, the Bellman equation is guaranteed to be bounded and it does not diverge.

Although very similar to the Finite Penalty Criterion for planning (Equation 3.5), the nature of this update rule is also close to Q-learning rule, which includes a temporal difference, optimizes policies (state-action value) instead of state values and includes a learning ratio α. We call this update-rule as **Q-learning-FP**.

Notice that, as in probabilistic planning (Trevizan *et al.*, 2017), solving an SSPUDE using Q-learning with the update rule 5.3 is equivalent of solving the SSP $\mathcal{M}' = \langle S, A', C', s_0, G\rangle$ where, $\forall s \in S$, $A'(s) = A(s) \cup \{\text{give-up}\}$, $C'(s, \text{give-up}) = D$, $C'(s, a, s') = C(s, a)\ \forall a \in A(s)$, and $T(s, \text{give-up}, s_g) = 1$ for any $s_g \in G$. With this new SSP, S' has no dead-ends and can be solved using any Regular (undiscounted) Q-learning algorithm for SSPs.

The Algorithm 4, called Q-learning-FP, can solve an SSPUDE that satisfies Assumption 4 by using a finite penalty D as input. We initialize the algorithm with all action-state values equal to 0 and use a sufficiently small learning rate to increase the smoothness of convergence. It is important to note that the penalty D is a parameter that needs to be tuned because it can limit the value of non-dead end states. Notice also that the penalty D is not explicitly associated with the dead-end states, but it is considered as a threshold for the Q function.

Algorithm 4 Q-Learning-FP (Finite Penalty)

```
Require: SSPUDE $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{C}, s_0, \mathcal{G})$, penalty $D$, epsilon $\epsilon$, epsilon decay $\delta_\epsilon$, learning
    rate $\alpha$
Ensure: Policy $\pi^{FP}$
 1: Initialize $Q^{FP}(s, a) \leftarrow 0$, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$
 2: $t \leftarrow 0$
 3: repeat
 4:     $s \leftarrow s_0$                                             ▷ New episode
 5:     $h \leftarrow 0$
 6:     repeat
 7:         Choose action $a$ in $s$ using policy derived from $Q^{FP}$ (e.g., $\epsilon$-greedy)
 8:         Take action $a$, observe cost $c$ and next state $s'$
 9:         $Q^{FP}(s, a) \leftarrow \min\left(D, Q^{FP}(s, a) + \alpha\left[c + \min_{a'} Q^{FP}(s', a') - Q^{FP}(s, a)\right]\right)$
10:         $s \leftarrow s'$
11:         $t \leftarrow t + 1$
12:         $h \leftarrow h + 1$
13:     until $s \in \mathcal{G}$ or $h \geq$ max horizon
14: until $t \geq$ max timesteps
15: return $\pi^{FP}(s) = \arg\min_{a \in A} Q^{FP}(s, a)$ for all $s \in \mathcal{S}$
```

In the Algorithm 4, Q-learning-FP, we initialize the function Q^{FP} with 0 for all (s, a) pairs and run a loop through a maximum number of timesteps in Line 3 to 13. Inside this loop, we start each episode in the initial state and reset the actual horizon in Lines 4-5. In the inner loop from Line 6 to 12, we chose an action a following an e-greedy policy on Line

7. Then, on Line 8, we apply this action in the environment, observe an immediate cost c and the next state s'. In the Line 9, we update the Q-value function with the modified Q-value function using the Finite-Penalty Criteria, as defined on Equation 5.3. In Line 15 the algorithm returns the greedy optimal policy.

Q-learning for iSSPUDE

As we have discussed in Section 3.3.2, the main limitation of the Finite-Penalty criterion for SSPUDEs, is that a state s can be viewed as a dead-end if it has an expected cost of reaching the goal greater than the penalty D (Trevizan *et al.*, 2017). Although for some problems, it might be easy to derive a good value for D, in domain independent reinforcement learning solution this is not allowed (i.e., in RL we make the assumption that we do not use knowledge about the problem being solved to derive D).

To handle these limitations, we can seek an RL alternative that directly maximize the probability of reaching the goal, ignoring the expected cost. This criterion is known as *Q-learning-MaxProb*.

Q-learning-MaxProb

Given an SSPUDE $\mathcal{M}$ according with Definition 3.2.3, we compile it as a Q-learning MaxProb MDP $\mathcal{M}'$, according with Def. 3.3.2 whose objective is to generate a set of policies that maximizes the probability to the goal. Therefore, a direct transformation into a Q-learning algorithm for MaxProb requires the following update rule:

Q-Learning MaxProb update rule

$$Q^{\mathrm{MP}}(s,a) \leftarrow \begin{cases} 1, & \text{if } s \in \mathcal{G}, \\ Q^{\mathrm{MP}}(s,a) + \alpha\left[r + \max_{a'} Q^{\mathrm{MP}}(s',a') - Q^{\mathrm{MP}}(s,a)\right], & \text{otherwise.} \end{cases} \tag{5.4}$$

In MAXPROB problems, while the optimization objective appears straightforward: finding a policy that maximizes the probability of reaching a goal state, it introduces several challenges.

The ***first challenge*** is that MaxProb is a sparse-reward MDP, since the agent only receives a reward upon reaching the goal state, it is often difficult to obtain meaningful learning signals using RL algorithms. To mitigate this, we introduce an Intrinsic Motivation (IM) term to encourage more effective exploration in the absence of frequent external rewards. This technique is heavily used in sparse-reward problems (Barto, 2013; Aubret *et al.*, 2019) and can help the agent to learn more effectively.

We define the intrinsic reward at state s as follows:

$$r_{int}(s) = \frac{N(s)}{t} \tag{5.5}$$

This term is weighted by a parameter $\beta \in [0,1]$, where $N(s)$ is the number of times state

s is visited and t is the number of total timesteps that the agent sampled. It corresponds to the percentage of times the agent visited a state in relation to all visits. As we add a negative term in the β parameter, this will penalizes the agent of going into regions that he visited many times. Thus, to learn the MaxProb action value function, called $Q^{\text{MP}} \in [0,1]$, using the intrinsic reward, we perform the update rule defined in Equation 5.6. Note that when $\beta = 0$ we have the original MaxProb formulation without intrinsic motivation.

Q-learning-MaxProb update rule with Intrinsic Motivation

$$Q^{\text{MP}}(s,a) \leftarrow \begin{cases} 1, & \text{if } s \in \mathcal{G}, \\ \max\left(0,\ Q^{\text{MP}}(s,a) + \alpha\left[(r - \beta \cdot r_{\text{int}}) + \max_{a'} Q^{\text{MP}}(s',a') - Q^{\text{MP}}(s,a)\right]\right), & \text{otherwise.} \end{cases} \tag{5.6}$$

Algorithm 5 Q-Learning-MaxProb (with Intrinsic Motivation and Elliminate Traps)

```
Require: MaxProb $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, s_0, \mathcal{G})$, epsilon $\epsilon$, epsilon decay $\delta_\epsilon$, learning rate $\alpha$,
    intrinsic motivation weight $\beta$
Ensure: Action-state values $Q^{MP}$ representing the maximum probability to reach the goal
 1: $Q^{MP}(s,a) = 0, \forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$
 2: $Q^{MP}(g,a) = 1, \forall g \in \mathcal{G}$ and $\forall a \in \mathcal{A}$
 3: $N(s,a) = 0, \forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$
 4: t = 1
 5: repeat
 6:     // New episode
 7:     $h = 0$
 8:     $s = s_0$
 9:     repeat
10:         Choose action $a$ from state $s$ using policy derived from $Q^{MP}$ (e.g., $\epsilon$-greedy)
11:         Take action $a$, observe reward $r$ and next state $s'$
12:         $N(s,a) = N(s,a) + 1$
13:         $r_{int}(s) = \frac{N(s)}{t}$                    ▷ compute the intrinsic reward
14:         $Q^{\text{MP}}(s,a) \leftarrow \max\left(0, Q^{\text{MP}}(s,a) + \alpha\left[(r - \beta \times r_{int}) + \max_{a'} Q^{\text{MP}}(s',a') - Q^{\text{MP}}(s,a)\right]\right)$
15:         $s \leftarrow s'$
16:         $h \leftarrow h + 1$
17:         $t \leftarrow t + 1$
18:     until $s$ is terminal or h $\leq$ max horizon
19:     Eliminate-Traps{$\mathcal{M}$, $Q^{\text{MP}}$}                    ▷ Eliminate Traps
20: until t $\leq$ max timesteps return $Q^{MP}$
```

The ***second challenge*** when solving MaxProb is that not all fixed point solution of the update rules 5.4 and 5.6 are optimal. As we discussed in Section 3.3.2, we need to eliminate traps during the learning process. One of the main issues is the presence of zero-reward cycles, sequences of actions and transitions where the agent can remain indefinitely without making any progress toward the goal. These cycles often do not decrease the value function under the Bellman backup as even infinite cycles have zero-reward. This is the case of some cycles in MaxProb. As a result, multiple fixed points may emerge, some of which correspond to policies that do not maximize the goal-reaching probability.

As we discussed in Section 3.3.2, to overcome this, we must combine the Bellman update with a trap elimination mechanism (Bellman Backup with Escaping Traps - BET) (Kolobov, M. Mausam, *et al.*, 2011; Kolobov, 2013), that filters out these undesirable behaviors. The key idea is to periodically analyze the transitions induced by the current greedy policy, i.e., the policy that selects actions maximizing the current action value estimates, and construct a greedy transition graph. Within this graph, we can find traps using efficient algorithms such as Tarjan's algorithm that identify strongly connected components (SCCs). Components that contain zero-reward loops and have no transitions leading to the goal indicate parts of the policy where the agent could be stuck indefinitely. Then, we can alter the action value function to penalize these states, such as putting infinite cost at the pair of state and actions, i.e. ($Q(s, a) = \infty$). By doing that, we eliminate the action and avoid that the agent goes into these cycles.

A ***third challenge*** when solving a MaxProb problem is due to the approximation nature of Q-learning. When we do the temporal-difference, we are approximating the optimal state-action values, but the uncertainty around this estimate is related to the number of experiences so far. Furthermore, Q-learning may suffer from the maximization bias (H. Hasselt, 2010; Sutton and Barto, 2018). In the case of MaxProb, we have zero-reward cycles that is analogous to the worst types of MDP with recurrent connections that significantly affects the maxization bias. In addition, these MDPs may require an exponential number of experiences to converge (H. P. v. Hasselt, 2011).

Offline Q-learning-S^{3}P

This method solves an SSPUDE using a lexicographic approach: finding first a set of MaxProb policies and then compute the one with the minimal cost. Thus, given an SSPUDE $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{C}, s_0, \mathcal{G})$, as in Definition 3.2.3, after transforming $\mathcal{M}$ into a MaxProb problem S^{3}P starts by finding the set of MaxProb policies Π^{MP} using Algorithm 5. Next, we propose a Q-learning algorithm that uses Π^{MP} to sample the episodes that end-up in some goal state. We call this algorithm Q-learning-S^{3}P.

Following Definition 3.3.4, to find policies that satisfy the S^{3}P optimization criterion for SSPUDEs, we need to collect experiences from episodes that correspond to traces that terminate in goal states. To ensure that we need to apply an offline Q-learning strategy. Thus Algorithm 6 describes a pseudo-code of applying Q-learning following the S^{3}P optimization criterion that collects episodes, and learns an optimal policy, using an offline approach. Note that this algorithm requires as input the set of MaxProb policies, Π^{MP}. The offline Q-learning–S^{3}P starts collecting a set of experiences from episodes that terminates in goal states using the set of MaxProb policies Π^{MP} (Lines 2-11). After that, the algorithm learns the state-action values $Q(s, a)$ that minimizes the cumulative expected cost only with respect to this set of experiences (Lines 12-18) that end-up in a goal state. This idea is called experience-replay (Lin, 1992) and is used in different online or offline algorithms, such as Deep Q-Networks (DQN) Mnih *et al.*, 2015.

Algorithm 6 Offline Q-learning-S^{3}P

```
Require: SSPUDE $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{C}, s_0, \mathcal{G})$
Require: $Q^{\text{MP}}$ that is the solution of a MaxProb derived from $\mathcal{M}$
Require: Offline max timesteps $T$, learning rate $\alpha$, discount factor $\gamma$
Ensure: Action-state values $Q(s, a)$ representing the minimum cost to reach the goal
    Step 1: Compute Max-Prob policies from $Q^{MP}$
 1:     $\Pi^{\text{MP}} \leftarrow \left\{\pi \,:\, Q^{\text{MP}}(s, \pi(s)) = \max_{a \in \mathcal{A}} Q^{\text{MP}}(s, a),\ \forall s \in \mathcal{S}\right\}$
    Step 2: Generate traces $T \in T^G$ and add the corresponding episodes into $\mathcal{B}$
 2: Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$
 3: while $|\mathcal{B}| < T$ do
 4:     Sample policy $\pi \sim$ Uniform($\Pi^{\text{MP}}$)
 5:     Generate episodes $\tau = (s_0, a_0, r_0, s_1, \dots, s_n)$ using $\pi$ on SSPUDE $\mathcal{M}$
 6:     if $s_n \in \mathcal{G}$ then                                      ▷ successful episode
 7:         for $t = 0$ to $n - 1$ do
 8:             Add transition $(s_t, a_t, r_t, s_{t+1})$ to buffer $\mathcal{B}$
 9:         end for
10:     end if
11: end while
    Step 3: Offline Q-learning on replay buffer $\mathcal{B}$
12: $Q(s, a) = 1000$ (pessimistic value), $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$
13: $Q(g, a) = 0$, $\forall g \in \mathcal{G}$ and $\forall a \in \mathcal{A}$
14: for $t = 1$ to $T$ do
15:     for all $(s, a, r, s') \in \mathcal{B}$ do
16:         $y \leftarrow r + \gamma \cdot \min_{a'} Q(s', a')$
17:         $Q(s, a) \leftarrow Q(s, a) + \alpha \cdot (y - Q(s, a))$
18:     end for
19: end for
20: return $Q(s, a)$
```

Offline Q-learning-MCMP

As in the Offline Q-learning-S^{3}P algorithm, Offline Q-learning-MCMP follows Definition 3.3.6 and also requires the solution of the MaxProb problem to generate episodes that correspond to traces that either end-up in some goal state or in some dead-end state. We also define an offline Q-learning-MCMP version since we need to use experiences from traces generated by the MaxProb policies Π^{MP} that guarantee to either terminate in goal states or dead-end states. Thus, given an SSPUDE $\mathcal{M} = (\mathcal{S}, \mathcal{A}, C, s_0, \mathcal{G})$, after transforming $\mathcal{M}$ into a MaxProb problem Offline-Q-learning-MCMP algorithm starts by finding the set of MaxProb policies Π^{MP} (Line 1). Lines 2-11 the algorithm generate traces that go into the replay buffer that either terminate in a goal state or a dead-end state (i.e. a state s_n that has probability 0 to reach the goal) (Line7). After the generation of a replay buffer $\mathcal{B}$, the algorithm computes the optimal Q(s,a) function that minimizes the cumulative expected cost.

Algorithm 7 Offline Q-learning-MCMP

```
Require: SSPUDE $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \mathcal{C}, s_0, \mathcal{G})$
Require: $Q^{\mathrm{MP}}$ that is the solution of a MaxProb derived from $\mathcal{M}$
Require: Offline max timesteps $T$, learning rate $\alpha$, discount factor $\gamma$
Ensure: Action-state values $Q(s, a)$ representing the minimum cost to reach the goal
    Step 1: Compute Max-Prob policies $Q^{MP}$
 1:     $$\Pi^{\mathrm{MP}} \leftarrow \left\{\pi \,:\, Q^{\mathrm{MP}}(s, \pi(s)) = \max_{a \in \mathcal{A}} Q^{\mathrm{MP}}(s, a),\ \forall s \in \mathcal{S}\right\}$$
    Step 2: Generate traces $T \in \{T^G, T^{DE}\}$ and add the corresponding episodes into $\mathcal{B}$
 2: Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$
 3: while $|\mathcal{B}| < T$ do
 4:     Sample policy $\pi \sim \mathrm{Uniform}(\Pi^{\mathrm{MP}})$
 5:     Generate episodes $\tau = (s_0, a_0, r_0, s_1, \dots, s_n)$ using $\pi$ on SSPUDE $\mathcal{M}$
 6:     if $s_n \in \mathcal{G}$ or $P^*(s_n) = 0$ then          ▷ if $T^G$ or $T^{DE}$, add experiences into $\mathcal{B}$
 7:         for $t = 0$ to $n - 1$ do
 8:             Add transition $(s_t, a_t, r_t, s_{t+1})$ to replay buffer $\mathcal{B}$
 9:         end for
10:     end if
11: end while
    Step 3: Offline Q-learning on replay buffer $\mathcal{B}$
12: $Q(s, a) = 1000$ (pessimistic value), $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$
13: $Q(g, a) = 0$, $\forall g \in \mathcal{G}$ and $\forall a \in \mathcal{A}$
14: for $t = 1$ to $T$ do
15:     for all $(s, a, r, s') \in \mathcal{B}$ do
16:         $y \leftarrow r + \gamma \cdot \min a' Q(s', a')$
17:         $Q(s, a) \leftarrow Q(s, a) + \alpha \cdot (y - Q(s, a))$
18:     end for
19: end for
20: return $Q(s, a)$
```

Note that the offline version of the Q-learning with the S3P and MCMP optimization criterion do not relay on any strategy of exploration/exploitation.

ϵ-Greedy MaxProb-Exploration

As an alternative to the offline approaches like Q-learning-S^{3}P and -MCMP, we introduce an online exploration strategy for solving the cost-minimization phase of the dual-optimization problem. This strategy leverages the pre-computed set of MaxProb policies, Π^{MP}, but leverages it in a fundamentally different way: to guide exploration during live interaction with the environment rather than to generate a static dataset.

We term this strategy **ϵ-greedy MaxProb-Exploration**. The core principle is to constrain the agent's exploration space. At any given state s, the ϵ-greedy action selection mechanism is restricted to choosing actions only from the set of MaxProb policies, $\Pi^{\text{MP}}(s)$. This ensures that the agent's exploratory actions still preserve the maximum possible probability of reaching the goal.

This online formulation presents some advantages over the offline methods discussed previously. Unlike learning from a fixed replay buffer $\mathcal{B}$, the agent can visit and learn about any state reachable under a MaxProb policy. Furthermore, it performs online updates from all types of trajectories as they occur.

5.3 Summary

A key theoretical strength of Q-learning is its convergence guarantees under certain conditions. It is proven to converge with probability one in IHD-MDPs, as well as in SSP problems where all policies are proper, i.e., policies that reach the goal state with probability one from any initial state. Moreover, even in SSPs that include improper policies, Q-learning still converges (J. N. Tsitsiklis, 1994), provided the Q-values remain nonnegative possibly going to infinite or bounded (Yu and D. P. Bertsekas, 2013; D. Bertsekas, 2019).

Therefore, regular Q-learning can be applied without modification to solve both traditional SSPs and SSPADEs, where at least one proper policy exists. However, in SSPUDEs, where no policy can guarantee reaching the goal from all states, Q-learning fails to satisfy the required assumptions for convergence and may produce unstable or suboptimal solutions.

To address this, we modify Q-learning by adapting some probabilistic planning criteria that are better suited for environments with unavoidable dead-end states. Specifically, we adapt Q-learning using the Finite Penalty, MaxProb, S^3P and MCMP criteria. These resulted in the following modified Q-learning algorithms: Q-learning with Finite Penalty (Algorithm 4), Q-learning with Maximum Probability (Algorithm 5), and Offline Q-learning S^3P (Algorithm 6) and Offline Q-learning MCMP (Algorithm 7).

5.4 Related Work

Goal-oriented Reinforcement Learning (GORL): Our work aims to solve SSP problems with dead-ends, but some authors focused on using RL to solve general SSP by introducing algorithms such as UC-SSP (Tarbouriech *et al.*, 2020). In Tarbouriech, 2022 the author argues that algorithms commonly used for MDPs, such as UCRL2 (Jaksch *et al.*, 2010), could also be used to solve SSP problems without dead-ends. Furthermore, our work focuses on solutions on discrete state space, however, recent work (Min *et al.*, 2022) proposes solutions that use function approximation to solve SSPs without dead-ends. These solutions are even extended to Bayesian settings, as in the PSRL-SSP algorithm (Jafarnia-Jahromi *et al.*, 2023).

Safe Reinforcement Learning (Safe RL): Reaching goals with maximum probability could be framed as a problem similar to Safe RL (Garcia and Fernandez, 2015; Amodei *et al.*, 2016; Fatemi and Sharma, 2019; Jansen *et al.*, 2020; Thiago D. Simão *et al.*, 2021; T. D. Simão, 2023; Gu *et al.*, 2023), since while the agent seeks to reach the goal, he also should try to find paths to the goal that are safe, i.e., have maximum probability to goal.

Goal-conditioned Reinforcement Learning (GCRL): In this problem (Liu *et al.*, 2022) there are different types of goal states. Therefore, a state or action value function should be conditioned to each of these types resulting in several optimal policies. Our approach involves finding a policy that reaches any state from a set of goal states, without having to condition the value function.

Multi-objective Reinforcement Learning (MORL): Another interesting approach is MORL (Roijers *et al.*, 2013; Hayes, Rădulescu, Bargiacchi, Källström, *et al.*, 2022; Hayes, Rădulescu, Bargiacchi, Kallstrom, *et al.*, 2023) that focus on obtaining Pareto optimal policies. In general, MORL problems involves MDPs with several conflicting reward functions optimizations. Considering our case, we could frame our problem of maximizing the goal-probability function while minimizing the costs as a multi-objective problem.

Chapter 6

Empirical Analysis

The aim of this chapter is to present an empirical analysis of our proposed RL algorithms modifications for solving SSP problems with avoidable and unavoidable dead-ends, i.e. SSPADE and SSPUDE problems. We investigate Q-learning in SSPs without dead-ends to baseline performance and assess discount factor impacts (including $\gamma = 1$). Second, examining Q-learning in SSPADEs to compare the sample efficiency of discounted versus undiscounted methods in learning goal-reaching policies.

The core evaluation focuses on our novel algorithms for SSPUDEs: Q-learning-FP, Q-learning-MaxProb, Offline Q-learning-S^{3}P, Offline Q-Learning-MCMP and ϵ-Greedy MaxProb-Exploration. We measure their effectiveness by tracking $V(s_0)$ convergence and goal success rates. Crucially, we validate these model-free approaches by benchmarking their results against exact solutions from corresponding model-based planners (VI, VI-FP, GPCI and MCMP).

6.1 Experimental Setup

In our experimental analysis we solve problems of the original Navigation domain, which was briefly introduced in Section 1.3. The original Navigation domain involves a stochastic environment, meaning that the actions may not always lead to the agent's desired outcome. Instead, there is a probability distribution over the action possible outcomes, introducing uncertainty and randomness in the sequential decision making. Each action incurs a unitary cost and the objective of the agent is to find the optimal policy that minimizes the expected cumulative cost towards the goal. We also solve problems of a modified version of the Navigation domain, called *Navigation with DE-Trap Domain*, described next.

The original Navigation Domain. This benchmark domain was originally proposed for SSPUDE problems (Sanner and Yoon, 2011) (Andrés *et al.*, 2018), i.e. SSPs with unavoidable dead-ends using the Probabilistic Planning Domain Definition Language (PPDDL). In a grid world $n \times m$ an agent must move from an initial location to a goal location. The agent can move (at cost 1) up, down, left, or right with probability 1, in almost all locations. When moving to a location in some internal line (except the bottom and top lines), there is a probability $p > 0$ the agent can "break", and no longer be able to move from that location and accomplish its mission towards the goal location, which configures a terminal dead-end state (one for each middle location). The agent's objective is to find the path that maximizes the probability to the goal with minimum expected cumulative cost. An optimal policy in this domain should make some commitment between the maximum probability to reach the goal and the minimum cumulative cost. To represent an SSPADE we can just define one or more middle locations with $p = 0$.

The Navigation with DE-Trap Domain. In this domain we modify the original Navigation domain for SSPUDE problems with nonterminal dead-end states where the applicable actions have positive costs. In this way we guarantee that improper policies have infinite cost (Assumption 4). Thus, the interpretation is that, instead of having a probability the agent will break and stop when reaching locations in the middle line, the agent will only be able to move left or right; The actions up and down can no longer be applied which implies the agent will not be able to accomplish its mission towards the goal, configuring a set of nonterminal dead-end states. Furthermore, the original domain have different probabilities in each column, thus, we parametrize our environment to support different columns with the same probabilities.

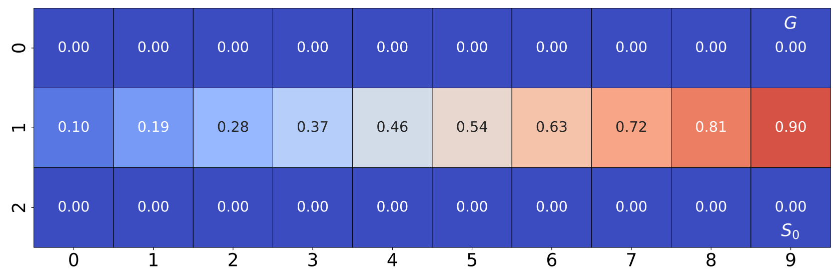


Figure 6.1: *Navigation-05 instance with* 3×10 *cells with unavoidable dead-ends. In the original domain, when the agent moves to one cell in the middle line it can have success with probabilility* $1 - p$ *or break with probability* p*, and no longer be able to move. The same instance can be generated in the Navigation with DE-Trap however, in the case the agent reaches a dead-end cell, it can still move left or right in the middle line configuring a nonterminal dead-end.*

Figure 6.1 shows an instance (Navigation-05) of the Navigation domain with 3 lines and 10 columns. The initial state is located in the bottom right corner; and the goal state is in the top right corner. Each cell has an associated probability p of breaking, and the background color shows the magnitude of this value which varies from 0.10 to 0.90. Note that this instance is an SSPUDE, because even using a policy that increases the probability to reach the goal, the agent may get stuck into a dead-end at least with probability of 10% (left column). In the original Navigation domain, a dead-end is terminal: when reached the agent stays there forever. In the Navigation with DE-Trap domain a dead-end is nonterminal: when reached by the agent it can still move to right or left staying in that middle line forever, not being able to reach the goal. The agent's objective is to find the path that minimizes the total cost while maximizing the probability of reaching the goal state from s_0. If we change the probability p of any middle line cell to 0, all dead-ends

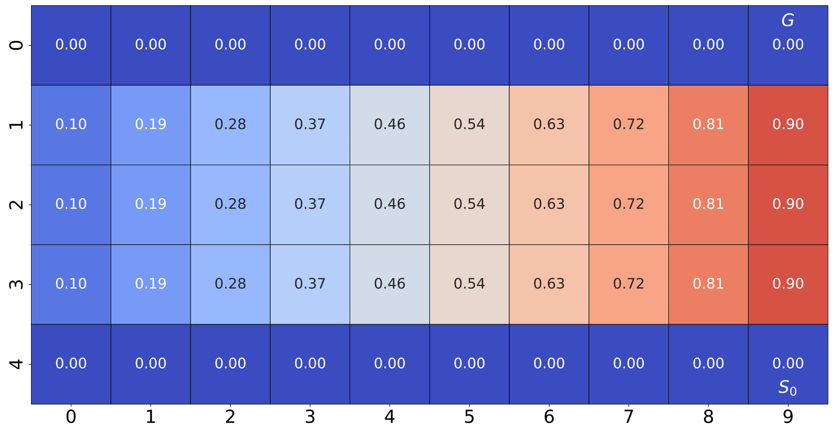


Figure 6.2: *Navigation-07 instance with* 5 × 10 *cells with unavoidable dead-ends. In this instance, when the agent tries to reach the goal on Line 0, it has to reach Lines 3, 2 and 1, all with some probability of success (break) that is computed as a joint probability.*

can be avoidable and the problem became an SSPADE.

Figure 6.2 shows the Navigation-07 instance defined on a 5 × 10 grid with unavoidable dead-end states. If the agent successfully reaches a cell in Line 3, it may then attempt to transition to Line 2 (through the same column). The probability of reaching Line 2 without breaking is $(1 - p)^2$. To reach Line 1 without breaking moving through the same column, the agent must succeed in three consecutive transitions: into Line 3, Line 2, and finally into Line 1. Thus, the probability of reaching Line 1 without breaking is $(1 - p)^3$. Notice that when the agent breaks it stays in the same cell if we are in original Navigation domain, or stay in the same line (Lines 3, 2 or 1) forever.

The environment simulator for the Navigation and Navigation with DE-Trap domains was created using OpenAI Gym/Gymnasium (Brockman *et al.*, 2016) (Towers *et al.*, 2024), based on the description of the original Navigation domain of IPPC'08 (International Probabilistic Planning Competition 2008) (Bryce and Buffet, 2008) used in PDDLGym (T. Silver and Chitnis, 2020). Table 6.1 shows the size of 10 instances from IPPC'08 analyzed in this work.

6.2 Experiment I: Using Reinforcement Learning to solve SSPADEs

As discussed in Section 3.2.1, in the case of an SSPADE, we can find a solution if there is a proper policy only w.r.t. the initial state s_0, not everywhere (Kolobov, M. Mausam, *et al.*, 2011) (Geffner and Bonet, 2013) and also, despite the presence of dead-ends, if the Assumption 3 of Definition 3.2.2 holds, i.e. all improper policies have an infinite cost.

In this experiment we solve the Navigation-05 instance transformed into an SSPADE (first column with probability 0 to break, i.e. $p = 0$) in the two Navigation domains using

Instance	# lines	# columns	# risky lines
Navigation-01	3	4	1
Navigation-02	3	5	1
Navigation-03	4	5	2
Navigation-04	6	5	4
Navigation-05	3	10	1
Navigation-06	4	10	2
Navigation-07	5	10	3
Navigation-08	3	20	1
Navigation-09	4	20	2
Navigation-10	5	20	3

Table 6.1: *Size of instances of the Navigation Domain; the risky lines are the ones containing cells with probability $p > 0$ to break.*

Equation 5.2. We only sampled experiences throughout episodes starting at s_0 that either terminate at the goal state or stay in the DE trap forever. Notice that the shortest path with probability 1 to the goal has 20 steps.

Table 6.6 shows the hyperparameter values used in this experiment. We show results of solving the Navigation-05 instance as an SSPADE in the two domains: (a) with terminal and (b) nonterminal dead-end states.

Parameter	Description	Value
T_{max}	Maximum number of timesteps	100k
T_{episode}	Maximum timesteps per episode	200
α	Learning rate	0.10
γ	Discount factor	[0.90, 0.95, 0.99, 1.00]
ϵ	Initial exploration rate	1.00
ϵ_{min}	Minimum exploration rate	0.05
δ_ϵ	Exploration rate decay	0.995

Table 6.2: *Q-learning Hyperparameters for Experiment I.*

Results in the Navigation with DE-Trap domain. Figure 6.3 left shows the result of applying Regular Q-learning to solve the Navigation-05 instance transformed into an SSPADE. In this domain we notice that, as expected, when using discount factor 1 or very close to 1. i.e. $\gamma = 0.99$ (red and green curves, respectively), the regular Q-learning algorithm converges to the optimal policy that is able to deviate of all dead-end states and achieve the goal state with probability 1 (success rate very close to 100%). This was expected because, in this domain, dead-ends are nonterminal and accumulating costs when the agent considered large enough horizons. When using Q-learning with $\gamma = 0.90$ and 0.95 (blue and orange curves, respectively) the learned policies can not achieve 100% success rate, even though all dead-ends could be avoidable. This is because the effective horizons considered by these discount factors are approximately 10 and 20 steps, respectively, which do not accumulate all necessary costs to avoid the nonterminal dead ends and therefore forces the agent to make suboptimal choices. Furthermore, the undiscounted solution ($\gamma = 1$) is more sample-efficient than any of the discounted ones. The undiscounted Q-learning agent achieved a 100% success rate after approximately 30,000 timesteps. In comparison, the most effective discounted configuration, with a discount factor of $\gamma = 0.99$, required around 35,000 timesteps to reach the same level of performance.

Results in the original Navigation domain. Figure 6.3 right shows the result of applying Regular Q-learning to solve the Navigation-05 instance transformed into an SSPADE in the original Navigation domain, i.e. where dead-ends are terminal states. In this case, since in this domain improper policies don't have infinite cost violating Assumption 4 i.e., dead-ends do not accumulate costs, regardless of the discount factor, Q-learning will choose the shortest path, despite it having the lowest probability of success. Notice that the success rate of all those policies are 10% which indicates they move to the goal throughout the last column (Column 9), i.e. with the lowest probability to the goal.

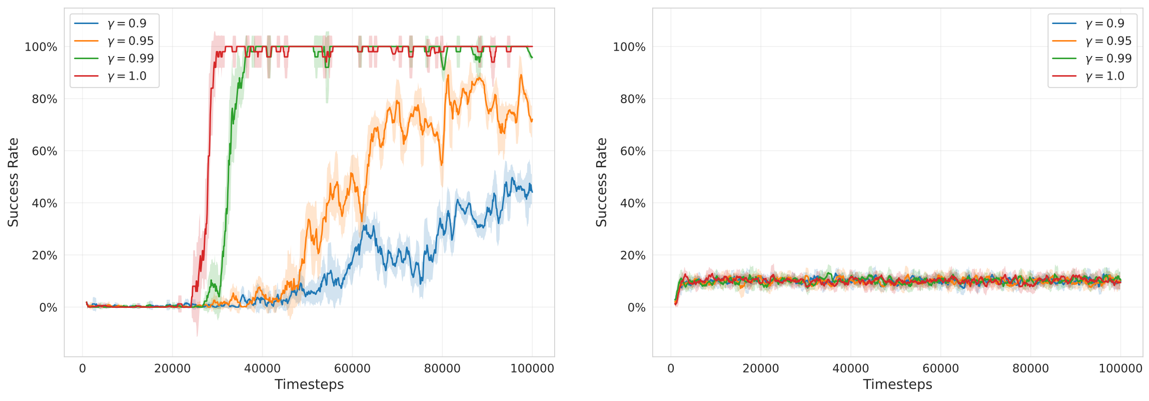


Figure 6.3: *Using Q-learning to solve a SSPADE in the Navigation-05 instance with nonterminal dead-ends (left) and terminal dead-ends (right).*

6.3 Experiment II: Using Reinforcement Learning to solve SSPUDEs with the Finite Penalty criterion

In Chapter 5, Section 5.2.3 we discussed the finite penalty criterion to solve SSPUDEs adapted to reinforcement learning and we defined the Algorithm 4, Q-learning-FP. In this experiment, we first validate the soundness Q-learning-FP showing that it converges to exact values computed by the corresponding planning algorithm for the 10 instances of the Navigation with DE-Trap domain. Then we show on the Navigation-05 instance how varying the penalty D can lead the agent to choose the optimal policy that maximizes the probability to the goal with the minimum cost, which is the expected trade-off known from the probabilistic planning research result.

Notice that we run experiments using the Navigation with DE-Trap domain but we could have the same results with the original Navigation domain (with terminal dead-end states) adding the "give-up" action with cost D (Section 5.2.3). Since in many domains we don't know the dead-end states beforehand, both solutions do not require dead-end identification.

Parameter	Description	Value
$T_{\max}$	Maximum number of timesteps	1M
T_{episode}	Maximum timesteps per episode	200
α	Learning rate	0.10
D	Finite Penalty	[0, 5, 15, 20, 25, ..., 100]
ϵ	Exploration rate	0.10

Table 6.3: *Q-learning Hyperparameters for the Experiment II.*

6.3.1 Soundness of Q-learning-FP

Figure 6.4 shows the convergence of the value function for the initial state, i.e. $V(s_0)$, updated during the execution of Algorithm 4 for the 10 instances of the Navigation with DE-Trap domain with penalty $D = 100$. For all instances $V(s_0)$ computed by Q-learning-FP (solid blue line) successfully converges to the exact value computed by the Value Iteration planning algorithm with the Finite Penalty criterion, called, VI-FP (red dashed line).

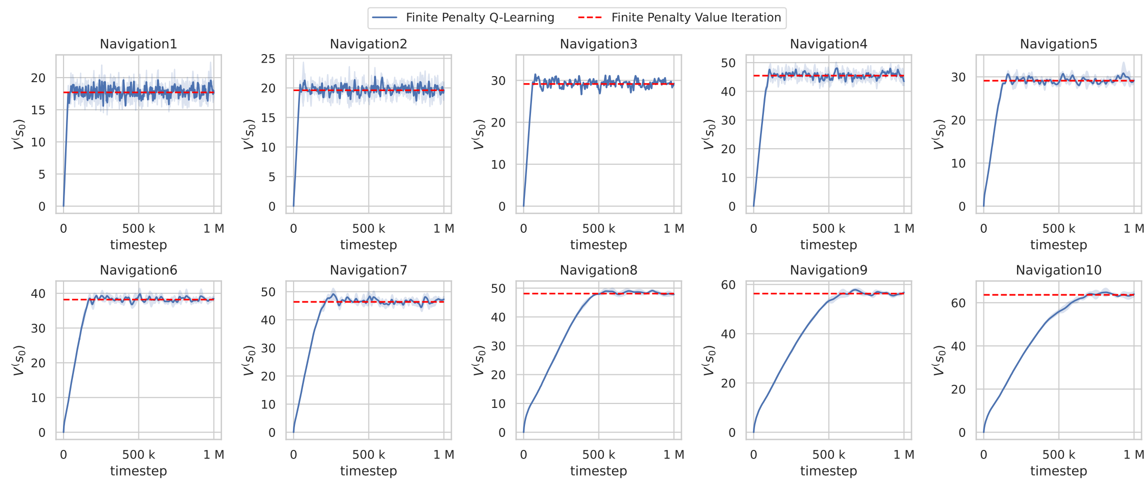
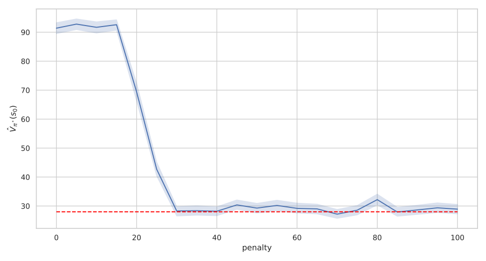


Figure 6.4: *Convergence of $V(s_0)$ per timestep computed by our proposed algorithm Q-learning-FP (solid blue line), in the Navigation with DE-Trap domain, with finite penalty $D = 100$; dashed red line is the exact solution obtained by the planner VI-FP using the same penalty.*

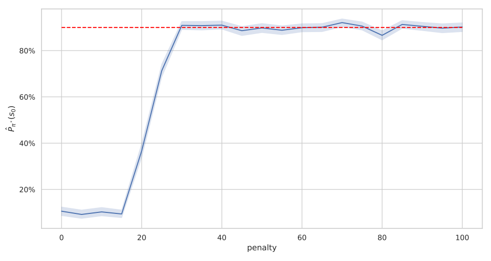
6.3.2 Varying the finite penalty D

This experiment investigates the effect of varying the penalty D on the learned policy π^* in the Navigation-05 instance of the Navigation with DE-Trap domain. Figure 6.5 presents the estimated value function $\hat{V}_\pi(s_0)$ (left) and the probability of reaching the goal $\hat{P}_\pi(s_0)$ (right) as a function of D, where $D \in [0, 100]$. The results, computed over 500k timesteps with an evaluation horizon of 100, reveal a clear transition in policy behavior. When the penalty is low, the agent prioritizes cost minimization over avoiding dead-ends, leading to higher estimated values but lower success probabilities. As D increases beyond a critical threshold (approximately D=20), the agent shifts toward a more conservative strategy, significantly improving goal-reaching probability while lowering the estimated value. The rightmost plot shows that for sufficiently high penalties, $\hat{P}_\pi(s_0)$ stabilizes near

100%, aligning with the optimal safe policy. These findings highlight the crucial role of the penalty parameter in balancing risk and cost efficiency in SSPUDEs.



(a) *$\hat{V}_{\pi^*}(s_0)$ for 1k episodes of horizon size 100.*



(b) *$\hat{P}_{\pi^*}(s_0)$ for 1k episodes of horizon size 100.*

Figure 6.5: *Q-learning-FP for SSPUDE in the Navigation-05: varying the penalty. Computation of $\hat{V}_{\pi^*}(s_0)$ and $\hat{P}_{\pi^*}(s_0)$ estimation of the optimal policy π^* learned using Q-learning-FP with penalty $D \in [0, 100]$ in 500k timesteps using an horizon of size 100.*

To generate the results shown in Figure 6.5, we conducted a systematic evaluation of the learned policy for different values of the penalty parameter D, ranging from 0 to 100 in increments of 5, that is, $D \in \{0, 5, 10, \dots, 100\}$. For each value of D, the agent was trained for a total of 500,000 interaction steps. After training, the resulting policy was fixed and evaluated across multiple independent episodes to estimate two key metrics: the expected cost-to-go from the initial state, $\hat{V}_\pi(s_0)$, and the probability of successfully reaching the goal, $\hat{P}_\pi(s_0)$.

Each evaluation episode was constrained to a maximum of 100 steps, enforcing a practical horizon within which the agent had to reach the goal. The metric $\hat{V}_\pi(s_0)$ represents the average number of steps required to reach the goal from the initial state s_0, but is computed only over those episodes in which the agent actually succeeded in reaching the goal. This provides a measure of the efficiency of the policy, conditional on success. On the other hand, $\hat{P}_\pi(s_0)$ denotes the empirical success rate, calculated as the proportion of evaluation episodes in which the agent reached the goal within the 100-step limit. This serves as an indicator of the policy's robustness in avoiding dead-ends.

By varying the penalty parameter D and observing the resulting changes in $\hat{V}_\pi(s_0)$ and $\hat{P}_\pi(s_0)$, we gain insight into how the agent's behavior adapts to the presence of dead-ends. Low penalty values encourage risk-taking behaviors aimed at minimizing path cost, often at the expense of safety, while higher penalties gradually shift the agent's preference toward safer, more conservative trajectories. This experimental setup reveals the critical role of the penalty in shaping the trade-off between cost efficiency and risk aversion in the SSPUDE setting.

6.4 Experiment III: Using Reinforcement Learning to solve SSPUDEs with the MaxProb criterion

Figure 6.6 shows the convergence of the MaxProb value function in s_0, i.e. $V^{\mathrm{MP}}(s_0)$, along 1M timesteps using our proposed Q-learning-MaxProb algorithm with Intrinsic Mo-

Parameter	Description	Value
T_{max}	Maximum number of timesteps	[1M, 3M]
T_{episode}	Maximum timesteps per episode	100
α	Learning rate	0.01
ϵ	Exploration rate	0.10
β	Intrinsic motivation weight	0.10

Table 6.4: *Q-learning Hyperparameters for the Experiment III.*

tivation and Eliminate Traps (Algorithm 5) on the 10 SSPUDE instances of the Navigation domain. The red dotted line is the exact planning solution obtained by the VI-MaxProb planning algorithm, a value iteration algorithm modified with the Bellman backup update rule of MaxProb formulation (Equation 3.7) and the Eliminate Trap procedure with the improvement of intrinsic motivation.

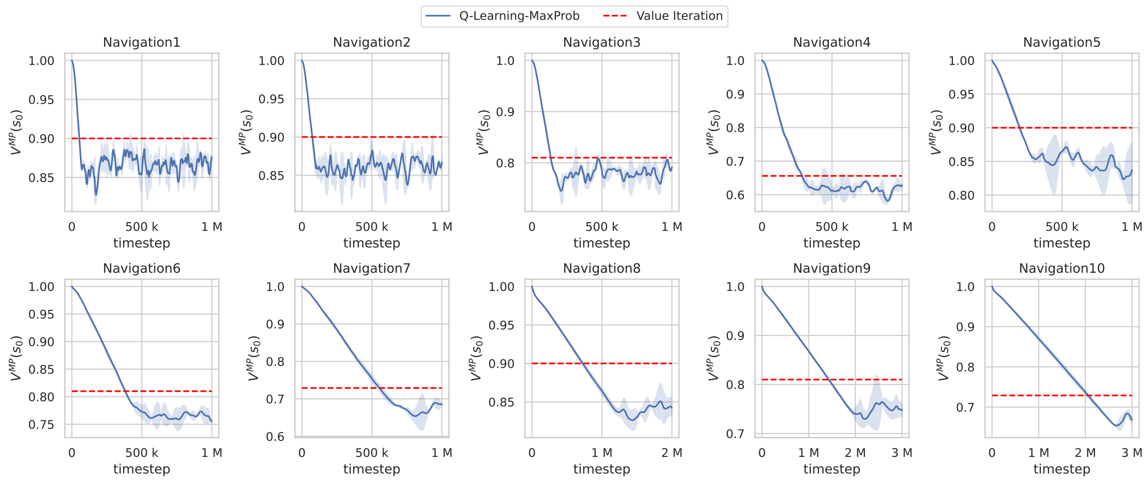


Figure 6.6: *Results of running the algorithm Q-learning-MaxProb with Intrinsic Motivation and Eliminate Traps on 10 instances of the original Navigation domain. Convergence of $V^{MP}(s_0)$ along 1M timesteps. The red dotted line is the exact solution of MaxProb for each instance computed using an exact VI-MaxProb planning solution.*

The results on Figure 6.6 show a convergence error up to 0.05, which is a good approximation to the exact value. Thus the set of MaxProb policies extracted from the learned MaxProb function, Q^{MP}, also considered an error up to 0.05.

6.5 Experiment IV: Using Offline Reinforcement Learning to solve SSPUDEs with the criteria: S^{3}P and MCMP

In this experiment we evaluate our proposed algorithms: the Offline-Q-learning-S^{3}P (Algorithm 6) and Offline-Q-learning-MCMP (Algorithm 7), to solve the Navigation-03

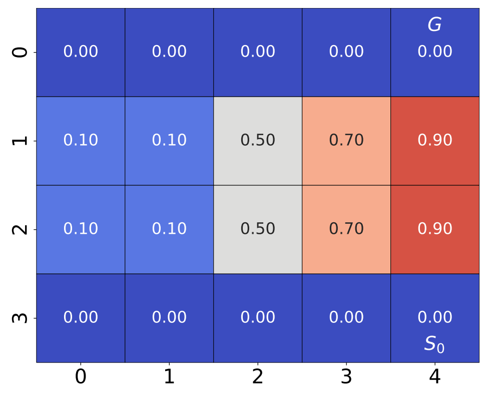


Figure 6.7: *Navigation-03 instance with the first and second column has the same minimum probabilities ($p = 0.10$) to move to dead-end states when crossing the 2 internal lines. Thus, $P^*(s_0) = 0.81 = (1-p) * (1-p)$.*

instance depicted in Figure 6.7. Notice that, in this instance, there are two lines with probabilities 0.10 to reach a dead-end and two columns with the same probabilities ($p = 0.10$). Thus, the maximum probability to reach the goal from s_0 is $P^*(s_0) = (1-p) * (1-p) = 0.81$. Table 6.5 outlines the hyperparameter values used in this experiment.

Parameter	Description	Value
$T_{\max}$	Maximum number of timesteps	1M
$\|\mathcal{B}\|$	Size of the Replay Buffer $\mathcal{B}$	100K
α	Learning rate	0.10
Q_0	Initial Q-value	20.0

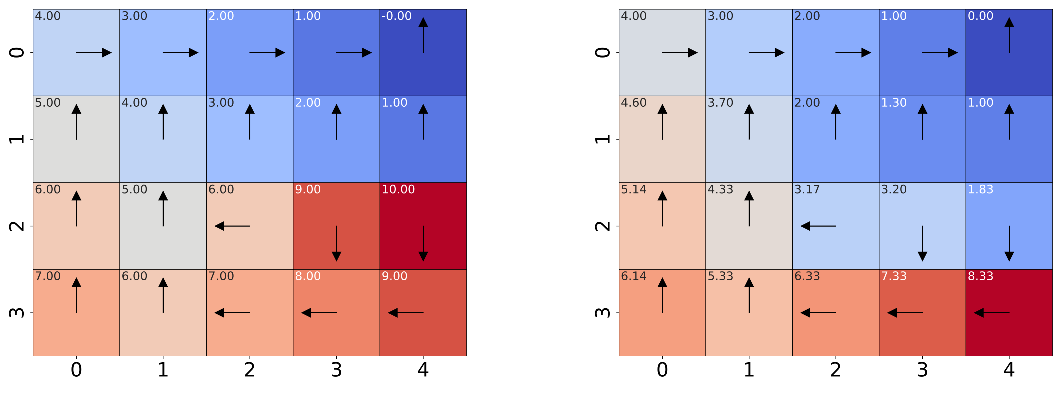
Table 6.5: *Q-learning Hyperparameters for the Experiment IV.*

We compare the results with the exact solutions computed by using our implementation of the corresponding planning solutions using the same optimization criteria:

- The GPCI (S^{3}P) planning implementation: we used the scientific computing package for Python called Numpy (van der Walt *et al.*, 2011) to implement GPCI. In the first step, we computed the MaxProb as defined in Equation 3.7. Then, in the second step, we compute the goal-cost function as defined in Equation 3.10.
- The MCMP planning implementation: we used a Linear Programming toolkit for Python called PuLP (Mitchell *et al.*, 2011) to specify the two linear programs defined in Section 3.3.4 and used the CBC linear solver (Forrest and Lougee-Heimer, 2005).

Figure 6.8 shows the exact policies and $V^*(s)$ optimal value function computed by the GPCI and the MCMP planners when solving the Navigation-03 instance. Notice that although they returned the same policies, the $V^*(s)$ values are different: while S^{3}P computes the exact optimal cost to reach the goal, the MCMP computes values that include the costs to dead-ends.

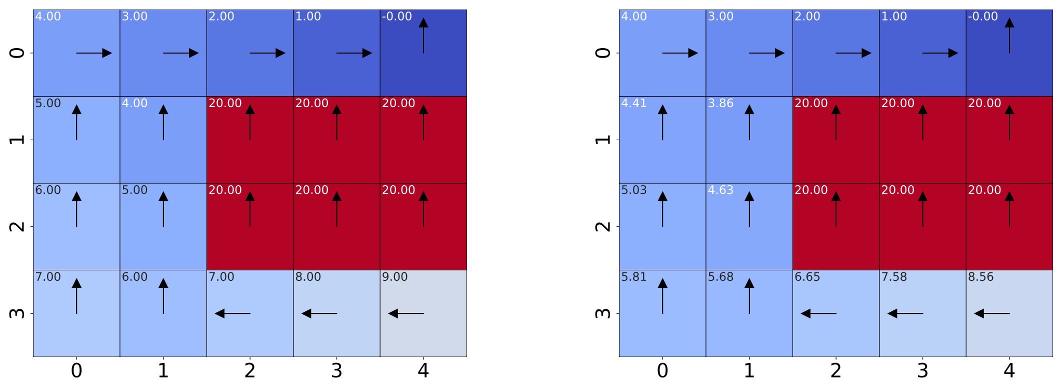
To run our Q-learning algorithms we first sample the complete episodes from s_0 using



(a) *GPCI (S^3P) optimal policy.* **(b)** *MCMP optimal policy.*

Figure 6.8: *Exact policy and $Q^*(s, a)$ function computed by our implemented version of the GPCI planner, which uses the S^3P criterion and the MCMP planner, which uses the MCMP SSPUDE criterion.*

the set of MaxProb policies computed in the Experiment III. In the case of S^{3}P, we consider only traces that terminate on the goal, and in the case of MCMP, we consider traces that terminates either on the goal or on a dead-end state. Despite the presence of multiple MaxProb policies that successfully reach the goal, both algorithms were able to learn the minimum-cost path among them.



(a) *Offline Q-learning-S^3P.* **(b)** *Offline Q-learning-MCMP.*

Figure 6.9: *Policy and Value function computed by the Q-learning-S^3P and the Q-learning-MCMP algorithms.*

Figure 6.9 shows the converged value function and policies computed by our proposed algorithms. Both are able to estimate the correct value function in the initial state. Note that the red cells are were not included by the traces sampled with MaxProb policies and therefore they were not updated since they kept their pessimist initial values. In the case of Q-learning-S^{3}P, it learned the exact same $V^*(s_0)$ as S^{3}P planning solution for the blue cells, while Q-learning-MCMP, computed approximated values when compared with the $V(s_0)$ computed with the MCMP planner due to the traces that possibly lead to a dead-end in the two risky lines.

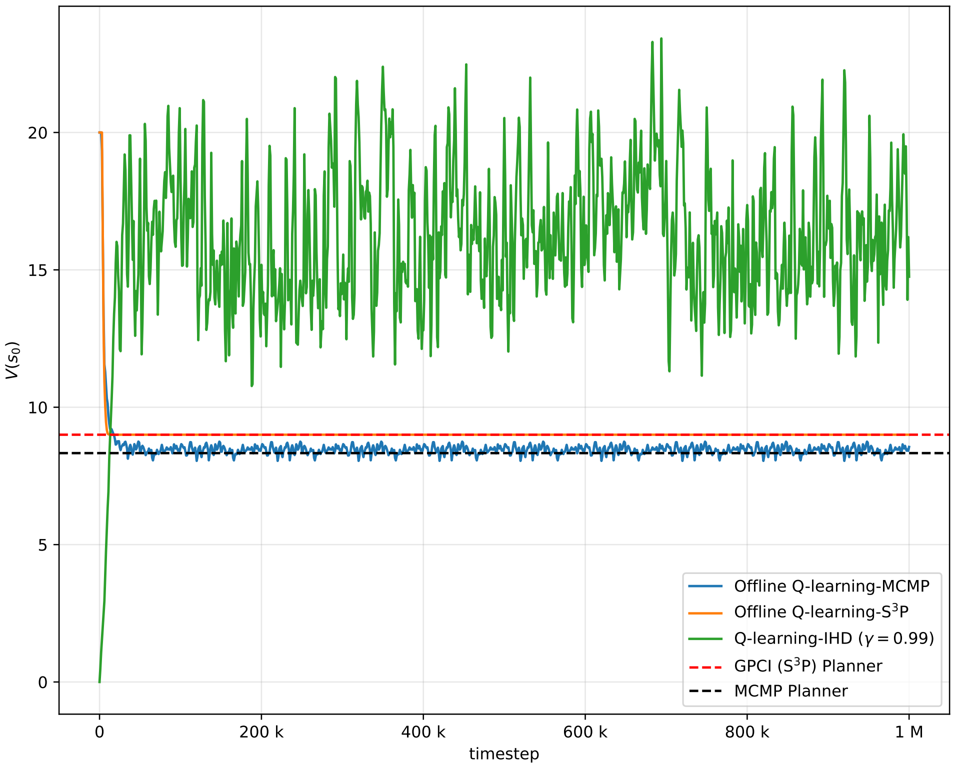


Figure 6.10: *$V(s_0)$ convergence on Navigation-03 of Figure 6.7 using our proposed Q-learning algorithms, the GPCI and MCMP planner, and Q-learning-IHD with $\gamma = 0.99$.*

Figure 6.10 shows the $V(s_0)$ convergence on Navigation-03 of Figure 6.7 using our proposed Q-learning algorithms, the GPCI and MCMP planner, and Q-learning-IHD with $\gamma = 0.99$ (Algorithm 3). The dashed lines (red and black) show the exact $V(s_0)$ computed by the S^{3}P and MCMP planners, respectively. The solid lines (blue, orange, green), are the values obtained for $V(s_0)$ by the Offline Q-Learning-MCMP, Offline Q-Learning-S^{3}P and Online Q-learning-IHD with $\gamma = 0.99$. Notice that Q-Learning-MCMP presents a variance due to the sample of traces that goes to dead-ends, although it converges to a value around the $V(s_0) = 8.33$ computed by the MCMP planner. When looking to the behavior of the Q-learning-S^{3}P, it convergences perfectly to the exact $V(s_0) = 9$ computed by the S^{3}P planner. However, when using Q-learning-IHD (i.e. using the regular Q-learning algorithm for Infinite Horizon Discounted MDP) with $\gamma = 0.99$, the convergence of $V(s_0)$ exhibits an unstable learning behavior. The value function estimates are different from the proposed criteria and the variance is very high, finding an approximated value for $V(s_0) = 16.4$. This corroborates with our discussion on Chapter 4.

Figure 6.11 compares the success rate of the policies learned with the Offline Q-learning-S^{3}P and Offline Q-learning-MCMP algorithms with the policy learned by the Q-learning-IHD with $\gamma = 0.99$. Notice that the success rate of our propose Offline Q-learning algo-

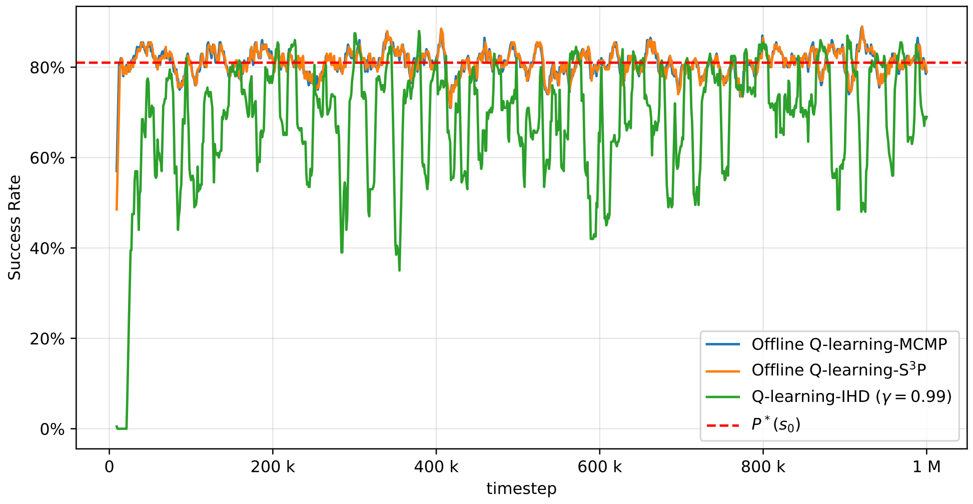


Figure 6.11: *Success Rate of the policies learned with the Offline Q-learning algorithms compared with the policy learned with the Q-learning-IHD for the Navigation-03 instance whose $P^{*}(s_0) = 81\%$.*

rithms are very close to $P^{*}(s_0) = 0.81$, while the success rate of the policy learned with the regular (discounted) Q-learning-IHD algorithm stays around 70%.

This experiment illustrates that the standard discounted criterion may have some limitations to solve problems such as SSPs with dead-ends, even being very powerful and useful to solve general discounted MDPs. It also corroborates with the statements denoted by D. Bertsekas, 2023, that is questionable that discounted MDPs are a universal framework, even using a discount factor near to 1.

Furthermore, one interpretation is that the discounted criterion consider significantly more traces that goes into dead-ends, introducing more uncertainty and noise into the state-action value function estimate.

Thus, our proposed methods using RL tailored to SSPs are useful and can bring a new avenue to connect planning criteria that are focused to efficiently solve SSPs, specially in the presence of dead-ends.

6.6 Experiment V: Regular Online Q-learning with ϵ-Greedy MaxProb-Exploration

Figure 6.12 shows the convergence of $V(s_0)$ on 10 SSPUDE instances of the Navigation with DE-Trap domain computed by an online Q-learning algorithm without discount, i.e. with $\gamma = 1$. In this version of online Q-learning, instead of applying a regular ϵ-greedy exploration strategy on all actions of each instance, we perform an ϵ-greedy exploration only over the set of MaxProb actions Π^{MP}, i.e. we perform an **ϵ-greedy MaxProb-Exploration**. That means we now consider only the MaxProb actions allowing visiting any reachable state while in the previously proposed offline algorithms (Experiment IV) we only visited and updates states present in the buffer $\mathcal{B}$). Also, differently from

Parameter	Description	Value
T_{max}	Maximum number of timesteps	1M
T_{episode}	Maximum timesteps per episode	100
α	Learning rate	0.10
ϵ	MaxProb-Exploration rate	0.10
Q_0	Initial Q-value	0.0

Table 6.6: *Q-learning Hyperparameters for the Experiment V.*

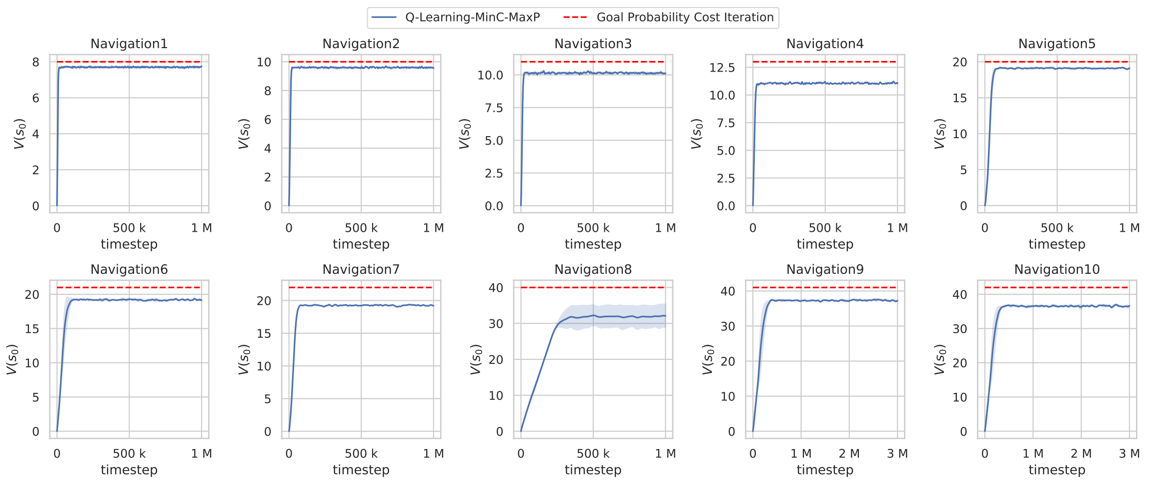


Figure 6.12: *Convergence of $V(s_0)$ in the Navigation with DE-Trap domain, using Q-learning-MinC-MaxP in comparison to the exact planning solution returned by the GPCI planner (*Teichteil-Königsbuch, 2012*).*

Experiment IV with Offline Q-learnig-S^{3}P and -MCMP, in this experiment we make online updates and allow any kind of traces, i.e. T^G, T^{DE} and traces that gets into dead-end traps limited by $T_{episode} = 100$. The results show that we can learn good approximations to the correct values of $V(s_0)$ (see Figure 6.6), when we compare with the results computed by the GPCI planner (Teichteil-Königsbuch, 2012).

Chapter 7

Conclusion

In this work, by connecting the insights from probabilistic planning into reinforcement learning (RL), we propose modifications to the classical Q-learning algorithm to solve Stochastic Shortest Path (SSP) problems, more specifically, SSPs in the presence of avoidable dead-ends (SSPADE) and unavoidable dead-ends (SSPUDE).

Despite the fact that SSPs are a more general class of problems than IHD-MDPs, they remain underexplored in the RL literature. This work demonstrates that studying SSPs in the context of RL enables the development of agents that can efficiently solve complex goal-oriented tasks, even the most complex ones that have the presence of unavoidable dead-end states.

Our proposed solutions handle some common issues that arise when we are trying to use RL to solve SSPs with Dead-Ends, such as Bellman equation divergence, discount factor tuning, penalty tuning, sparse rewards, maximal probability to the goal and dead-end inference (or detection) during the learning process.

In summary, the main contributions of this work are:

- **Analysis of the IHD-MDP model for Goal-Oriented Tasks:** We analyze the IHD-MDP using illustrative examples to show that this formulation can lead to suboptimal policies. We demonstrate cases where, despite the existence of paths with a higher probability of reaching the goal, the discounted objective selects a path with a lower probability of success. This analysis aligns with our experimental results, such as Experiment 6.2 and 6.5, in which we find that this criterion can produce unstable policies with a lower chance of reaching the goal.
- **Ensuring Convergence via Finite Penalty Bounding:** We propose a Q-learning-FP algorithm that integrates the finite penalty planning criterion directly into the Q-learning update rule. This method is designed for fSSPUDEs, i.e., problems where the finite penalty is known. This modification introduces an explicit upper bound on state-action values during learning a solution for SSPUDEs, effectively preventing the Bellman equation divergence caused by infinite costs associated with dead-ends or improper policies. Our Experiment 6.3, shows that this modification enable convergence when solving fSSPUDEs problems when using RL.

- **Learning Maximum Goal Probability Policies with Intrinsic Motivation:** To address scenarios where maximizing the probability of reaching the goal is fundamental, such as iSSPUDE problems, we developed a Q-learning variant, called Q-learning-MaxProb. Since MaxProb is a problem with a sparse reward nature, we enhanced the learning process by incorporating an intrinsic motivation mechanism based on state visitation counts. The Experiment 6.4 provide empirical evidence that this addition facilitates exploration and improves convergence towards policies that maximize the goal-reaching probability.

- **Dual-optimization Q-learning criterion:** We introduced a dual-optimization Offline Q-learning approach based on the S^3P and MCMP planning optimality criteria for iSSPUDE Trevizan *et al.*, 2017. As in probabilistic planning, this approach first employs Q-learning-MaxProb to estimate the maximum goal probability and extract a set of optimal MaxProb policies Π^{MP}. Subsequently, it uses a distinct Offline Q-learning process to generate a replay buffer $\mathcal{B}$ using Π^{MP} policies, to select the policy that minimizes the expected cumulative cost. As shown in Experiment 6.5, this method effectively balances maximizing reachability to the goal state and minimizing costs in environments with unavoidable dead-ends, a solution that can not be found using the classical IHD MDP approach from RL.

- **Using the MaxProb solution in Offline Versus Online Q-learning:** Since we have computed a set of MaxProb policies Π^{MP}, with Algorithm 5, we can either use this set of actions to generate the traces T^G and T^{DE}, as in the Offline Q-learning-S^3P and -MCMP, or we can use Π^{MP} on an Online Q-learning algorithm that performs an ϵ-greedy exploration only over the actions from Π^{MP}, called **ϵ-greedy MaxProb-exploration**. By restricting the agent to actions known to preserve the highest chance of success, this method dramatically improves both the safety and sample efficiency of learning.

- **Empirical Evaluation Against Exact Planning Methods and IHD-MDPs:** We conducted experiments on a SSP benchmark domain for SSPUDEs: the Navigation domain and its extension, called Navigation with DE-Trap that guarantee that all improper policies have infinity cost. These experiments, systematically compared the convergence and performance (in terms of learned value functions, goal achievement probability or success rate) of our proposed Q-learning variants against their exact probabilistic planning solutions counterparts (VI-FP, VI-MaxProb, GPCI and i-dual with MCMP). In addition, we also compared the solutions using regular Q-Learning that solves an IHD-MDP model against our proposed Q-learning algorithms. The results demonstrated that our RL algorithms successfully converge and yield value functions and policies that closely approximate the exact solutions derived from planning. Besides we show that by modeling sequential decision making problems using SSPs, rather than using IHD-MDPs, we can obtain policies that reaches the goal with less sample complexity with a quality that guarantees the maximal probability to the goal. Our final experiment evaluation, Experiment 6.6, shows that we can combine IHD-MDP with SSPUDE when using what we call **ϵ-greedy MaxProb-exploration**, which results with better convergence values and return the optimal policy according with the dual optimization criteria, with advantage of allowing online learning in the cost minimization phase.

7.1 Discussion

This work bridges concepts from Probabilistic Planning and RL to address the challenge of solving SSP problems with dead-ends. We began by highlighting the limitations of the IHD-MDP framework, particularly its sensitivity to the discount factor and its potential failure to either guarantee goal achievement or properly handle dead-end states, as illustrated in our examples (Fig 1.1a). Our core contribution is the adaptation of established planning criteria, such as Finite Penalty, MaxProb, $S^3P$and MCMP, into novel Q-learning variants (Q-learning-FP, Q-learning-MaxProb, Offline Q-learning-S^3P, Offline Q-learning-MCMP and Online Q-learning with ϵ-Greedy MaxProb-Exploration) designed to explicitly handle avoidable (SSPADE) and unavoidable (SSPUDE) dead-ends.

The empirical results presented in Chapter 6 provide compelling evidence for this approach. We confirmed that for SSPs where a proper policy is guaranteed to exist (SSPs and SSPADEs), standard undiscounted Q-learning (Eq 5.2) is not only theoretically sound but also practically effective and more sample efficient than discounted versions.

Furthermore, the use of intrinsic motivation also proved to be effective in helping the convergence for the sparse-reward MaxProb formulation. This suggests that for goal-oriented RL tasks, adopting an SSP perspective and leveraging appropriate planning-inspired criteria can lead to better learning agents.

In conclusion, this research provides strong evidence that the SSP framework, combined with criteria developed in probabilistic planning, offers a powerful and appropriate lens for tackling goal-oriented problems within RL, especially when dead-ends are present.

7.2 Future Work

This work opens up a variety of exciting research opportunities in the intersection of reinforcement learning and probabilistic planning, providing a foundation for future developments that can bridge the gap between some planning and reinforcement learning algorithms.

Several promising avenues include:

- **Using Multi-Objective Reinforcement Learning (MORL) to solve SSPs with Dead-ends**: Our proposed solutions used one (minimizing costs) or two criteria (maximizing probability and minimizing costs) to solve SSPs with Dead-Ends. One potential line of study is to extend this notion by introducing multi-objectives to handle SSPs with Dead-Ends. This can be addressed using MORL methods.

- **Using Deep Reinforcement Learning to solve SSPs with Dead-ends**: In this work we proposed solutions for tabular methods, however, we do think that our research can be extended for more complex scenarios with continuous state and action spaces. To solve these problems, we would like to extend our methods to Deep Reinforcement Learning solutions.

- **Using Model-based Reinforcement Learning (Model-based RL) to solve SSPs with Dead-ends**: Our proposed solutions used model-free approaches to solve SSPs with Dead-Ends, however, we think that using model-based RL methods can be an alternative to plan about different paths. Furthermore, some planning criteria rely directly on the transition probability function, thus, we can use model-based RL methods to estimate them and use it to obtain novel solutions.
- **Incorporate methods to quantify confidence in value functions**: Some proposed solutions use the value function to derive a MaxProb policy, which is then applied in the subsequent MinCost step. Due to the inherent uncertainty in the value function during the MaxProb step, our approach could benefit from methods that provide confidence measures indicating how close the estimated value function is to optimal. Such methods include Interval Estimation (IE) and its model-based variants, Model-Based Interval Estimation (MBIE) and MBIE with Exploration Bonus (MBIE-EB).
- **Using efficient exploration methods to solve SSPs with Dead-ends**: Our proposed solutions were primarily evaluated using undirected exploration techniques, such as ϵ-greedy. However, we believe that adopting the directed exploration strategies commonly employed in PAC-MDP algorithms, which consider factors such as uncertainty, offers a promising direction to improve exploration. These strategies could further enhance our solutions by providing additional guarantees while reducing sample complexity.
- **Exploring another intrinsic motivation techniques to solve MaxProb problems**: Future work could explore other intrinsic motivation techniques that may further accelerate learning, especially in sparse reward environments.
- **Modeling multiple agents to solve SSPs**: A promising direction for future research is to investigate the solutions of SSP in multi-agent settings, where multiple agents must navigate shared environments with dead-ends.

Appendix A

Computing details for Examples

A.1 Computing details for Example 1 without dead-end costs

A.1.1 MCMP

Computing the occupation measure

$$
\begin{aligned}
x_{s_g}^{\pi_0} &= p^{\max} = \boxed{\frac{1}{3}} \\
x_{s_1}^{\pi_0} &= \frac{x_{s_g}^{\pi_1}}{\mathcal{T}(s_1, a_0, s_g)} = \frac{1/3}{0.5} = \boxed{\frac{2}{3}} \\
x_{s_0}^{\pi_0} &= \frac{x_{s_1}^{\pi_1}}{\mathcal{T}(s_0, a_0, s_1)} = \frac{2/3}{0.5} = \boxed{\frac{4}{3}}
\end{aligned}
$$

$$
\begin{aligned}
x_{s_g}^{\pi_1} &= p^{\max} = \boxed{\frac{1}{3}} \\
x_{s_2}^{\pi_1} &= \frac{x_{s_g}^{\pi_1}}{\mathcal{T}(s_2, a_1, s_g)} = \frac{1/3}{0.25} = \boxed{\frac{4}{3}} \\
x_{s_0}^{\pi_1} &= \frac{x_{s_2}^{\pi_1}}{\mathcal{T}(s_0, a_1, s_2)} = \frac{4/3}{1.0} = \boxed{\frac{4}{3}}
\end{aligned}
$$

Computing the value function

$$V^{\pi_0}_{MCMP}(s_0) = \frac{4}{3}\cdot 1 + \frac{2}{3}\cdot 3 = \frac{4}{3} + 3\cdot\frac{2}{3} = \boxed{\frac{10}{3}}$$
$$V^{\pi_1}_{MCMP}(s_0) = \frac{4}{3}\cdot 1 + \frac{4}{3}\cdot 2 = \frac{4}{3}(1+2) = \boxed{4}$$

A.1.2 IHD

The IHD criterion is used in MDPs and if we want to apply it to an SSP with dead-ends we must assume some value for the dead-end states. Let us assume that $V(d_1) = V(d_2) = V(d_3) = 0$.

Computing the policy π_0

Computing $V^{\pi_0}(s_0)$:

$$V^{\pi_0}(s_0) = \mathcal{C}(s_0, a_0) + \gamma(\mathcal{T}(s_0, a_0, s_1)V^{\pi_0}(s_1) + \mathcal{T}(s_0, a_0, d_1)V^{\pi_0}(d_1))$$
$$V^{\pi_0}(s_0) = 1 + \gamma(0.5V^{\pi_0}(s_1) + 0.5\cdot 0)$$
$$V^{\pi_0}(s_0) = 1 + 0.5\gamma V^{\pi_0}(s_1)$$

Computing $V^{\pi_0}(s_1)$:

$$V^{\pi_0}(s_1) = \mathcal{C}(s_1, a_0) + \gamma(\mathcal{T}(s_1, a_0, s_0)V^{\pi_0}(s_0) + \mathcal{T}(s_1, a_0, d)V^{\pi_0}(s_g))$$
$$V^{\pi_0}(s_1) = 3 + \gamma(0.5\cdot V^{\pi_0}(s_0) + 0.5\cdot V^{\pi_0}(s_g))$$
$$V^{\pi_0}(s_1) = 3 + 0.5\gamma V^{\pi_0}(s_0)$$

We do have the following system of equations:

$$V^{\pi_0}(s_0) = 1 + 0.5\gamma V^{\pi_0}(s_1)$$
$$V^{\pi_0}(s_1) = 3 + 0.5\gamma V^{\pi_0}(s_0)$$

We can rename $V^{\pi_0}(s_0)$ to x and $V^{\pi_0}(s_1)$ to y. Thus, we can solve the following system of equations:

$$x = 1 + 0.5\gamma y$$
$$y = 3 + 0.5\gamma x$$

Computing x:

$$\begin{aligned}
x &= 1 + 0.5\gamma y \\
x &= 1 + 0.5\gamma(3 + \gamma 0.5x) \\
x &= 1 + 1.5\gamma + \gamma^2 0.25x \\
x - \gamma^2 0.25x &= 1 + 1.5\gamma \\
x(1 - \gamma^2 0.25) &= 1 + 1.5\gamma \\
x &= \frac{1 + 1.5\gamma}{(1 - 0.25\gamma^2)}
\end{aligned}$$

Computing y:

$$\begin{aligned}
y &= 3 + 0.5\gamma x \\
y &= 3 + 0.5\gamma(1 + 0.5\gamma y) \\
y &= 3 + 0.5\gamma + 0.25\gamma^2 y \\
y - 0.25\gamma^2 y &= 3 + \gamma 0.5 \\
y(1 - 0.25\gamma^2) &= 3 + \gamma 0.5 \\
y &= \frac{3 + \gamma 0.5}{1 - 0.25\gamma^2}
\end{aligned}$$

Then, the values of $V^{\pi_0}(s_0)$ and $V^{\pi_0}(s_1)$ are:

$$V^{\pi_0}(s_0) = \frac{1 + 1.5\gamma}{(1 - 0.25\gamma^2)} \tag{A.1}$$

$$V^{\pi_0}(s_1) = \frac{3 + 0.5\gamma}{1 - 0.25\gamma^2} \tag{A.2}$$

Computing the policy π_1

Computing $V^{\pi_1}(s_0)$:

$$\begin{aligned}
&V^{\pi_1}(s_0) = \mathcal{C}(s_0, a_1) + \gamma(\mathcal{T}(s_0, a_1, s_2)V^{\pi_1}(s_2)) \\
&V^{\pi_1}(s_0) = 1 + \gamma(1 \cdot V^{\pi_1}(s_2)) \\
&V^{\pi_1}(s_0) = 1 + \gamma V^{\pi_1}(s_2)
\end{aligned}$$

Computing $V^{\pi_1}(s_2)$:

$$\begin{aligned}
V^{\pi_1}(s_2) &= \mathcal{C}(s_2, a_1) + \gamma(\mathcal{T}(s_2, a_1, s_0)V^{\pi_1}(s_0) + \mathcal{T}(s_2, a_1, d_2)V^{\pi_1}(d_2) + \mathcal{T}(s_2, a_1, s_g)V^{\pi_1}(s_g))\\
V^{\pi_1}(s_2) &= 2 + \gamma(0.25V^{\pi_1}(s_0) + 0.5V^{\pi_1}(d_2) + 0.25V^{\pi_1}(s_g))\\
V^{\pi_1}(s_2) &= 2 + \gamma(0.25V^{\pi_1}(s_0) + 0.5 \cdot 0 + 0.25 \cdot 0)\\
V^{\pi_1}(s_2) &= 2 + \gamma(0.25V^{\pi_1}(s_0))\\
V^{\pi_1}(s_2) &= 2 + 0.25\gamma V^{\pi_1}(s_0)
\end{aligned}$$

Thus, we do have the following system of equations:

$$\begin{aligned}
V^{\pi_1}(s_0) &= 1 + \gamma V^{\pi_1}(s_2)\\
V^{\pi_1}(s_2) &= 2 + 0.25\gamma V^{\pi_1}(s_0))
\end{aligned}$$

We can rename $V^{\pi_1}(s_0)$ as a and $V^{\pi_1}(s_2)$ as b. Then, we can solve the following system of equations:

$$\begin{aligned}
a &= 1 + \gamma b\\
b &= 2 + 0.25\gamma a
\end{aligned}$$

Computing a:

$$\begin{aligned}
a &= 1 + \gamma b\\
a &= 1 + \gamma(2 + 0.25\gamma a)\\
a &= 1 + 2\gamma + 0.25\gamma^2 a\\
a - 0.25\gamma^2 a &= 1 + 2\gamma\\
a(1 - 0.25\gamma^2) &= 1 + 2\gamma\\
a &= \frac{1 + 2\gamma}{(1 - 0.25\gamma^2)}
\end{aligned}$$

Computing b:

$$\begin{aligned}
b &= 2 + 0.25\gamma a \\
b &= 2 + 0.25\gamma(1 + \gamma b) \\
b &= 2 + 0.25\gamma + 0.25\gamma^2 b \\
b - 0.25\gamma^2 b &= 2 + 0.25\gamma \\
b(1 - 0.25\gamma^2) &= 2 + 0.25\gamma \\
b &= \frac{2 + 0.25\gamma}{(1 - 0.25\gamma^2)}
\end{aligned}$$

Thus, the values of $V^{\pi_1}(s_0)$ and $V^{\pi_1}(s_2)$ are:

$$V^{\pi_1}(s_0) = \frac{1 + 2\gamma}{(1 - 0.25\gamma^2)} \tag{A.3}$$

$$V^{\pi_1}(s_2) = \frac{2 + 0.25\gamma}{(1 - 0.25\gamma^2)} \tag{A.4}$$

A.2 Computing details for Example 1 with positive dead-end costs

A.2.1 MCMP

Computing the occupation measure

$$\begin{aligned}
x_{s_g}^{\pi_0} &= p^{\max} = \boxed{\frac{1}{3}} \\
x_{s_1}^{\pi_0} &= \frac{x_{s_g}^{\pi_1}}{\mathcal{T}(s_1, a_0, s_g)} = \frac{1/3}{0.5} = \boxed{\frac{2}{3}} \\
x_{s_0}^{\pi_0} &= \frac{x_{s_1}^{\pi_1}}{\mathcal{T}(s_0, a_0, s_1)} = \frac{2/3}{0.5} = \boxed{\frac{4}{3}}
\end{aligned}$$

$$\begin{aligned}
x_{s_g}^{\pi_1} &= p^{\max} = \boxed{\frac{1}{3}} \\
x_{s_2}^{\pi_1} &= \frac{x_{s_g}^{\pi_1}}{\mathcal{T}(s_2, a_1, s_g)} = \frac{1/3}{0.25} = \boxed{\frac{4}{3}} \\
x_{s_0}^{\pi_1} &= \frac{x_{s_2}^{\pi_1}}{\mathcal{T}(s_0, a_1, s_2)} = \frac{4/3}{1.0} = \boxed{\frac{4}{3}}
\end{aligned}$$

Computing the value function

$$V^{\pi_0}_{MCMP}(s_0) = \frac{4}{3} \cdot 1 + \frac{2}{3} \cdot 3 = \frac{4}{3} + 3 \cdot \frac{2}{3} = \boxed{\frac{10}{3}}$$
$$V^{\pi_1}_{MCMP}(s_0) = \frac{4}{3} \cdot 1 + \frac{4}{3} \cdot 2 = \frac{4}{3}(1 + 2) = \boxed{4}$$

A.2.2 IHD

Assuming that the dead end cost is d, assuming that $d > 0$:

Computing the policy π_0

Computing $V^{\pi_0}(s_0)$:

$$V^{\pi_0}(s_0) = C(s_0, a_0) + \gamma(\mathcal{T}(s_0, a_0, s_1)V^{\pi_0}(s_1) + \mathcal{T}(s_0, a_0, d_1)V^{\pi_0}(d_1))$$
$$V^{\pi_0}(s_0) = 1 + \gamma(0.5V^{\pi_0}(s_1) + 0.5 \cdot d)$$
$$V^{\pi_0}(s_0) = 1 + 0.5\gamma V^{\pi_0}(s_1) + 0.5\gamma \cdot d$$

Computing $V^{\pi_0}(s_1)$:

$$V^{\pi_0}(s_1) = C(s_1, a_0) + \gamma(\mathcal{T}(s_1, a_0, s_0)V^{\pi_0}(s_0) + \mathcal{T}(s_1, a_0, d)V^{\pi_0}(s_g))$$
$$V^{\pi_0}(s_1) = 3 + \gamma(0.5 \cdot V^{\pi_0}(s_0) + 0.5 \cdot V^{\pi_0}(s_g))$$
$$V^{\pi_0}(s_1) = 3 + 0.5\gamma V^{\pi_0}(s_0)$$

Thus, we do have the following system of equations:

$$V^{\pi_0}(s_0) = 1 + 0.5\gamma V^{\pi_0}(s_1) + 0.5\gamma \cdot d$$
$$V^{\pi_0}(s_1) = 3 + 0.5\gamma V^{\pi_0}(s_0)$$

We can rename $V^{\pi_0}(s_0)$ as x and $V^{\pi_0}(s_1)$ as y. Then, we can solve the following system of equations:

$$x = 1 + 0.5\gamma y + 0.5\gamma \cdot d$$
$$y = 3 + 0.5\gamma x$$

Computing x:

$$\begin{aligned}
x &= 1 + 0.5\gamma y + 0.5\gamma \cdot d \\
x &= 1 + 0.5\gamma(3 + 0.5\gamma x) + 0.5\gamma \cdot d \\
x &= 1 + 0.5 + 1.5\gamma + 0.25\gamma^2 x + 0.5\gamma \cdot d \\
x - 0.25\gamma^2 x &= 1 + 1.5\gamma + 0.5\gamma \cdot d \\
x(1 - 0.25\gamma^2) &= 1 + 1.5\gamma + 0.5\gamma \cdot d \\
x &= \frac{1 + 1.5\gamma + 0.5\gamma \cdot d}{(1 - 0.25\gamma^2)}
\end{aligned}$$

Computing y:

$$\begin{aligned}
y &= 3 + 0.5\gamma x \\
y &= 3 + 0.5\gamma(1 + 0.5\gamma y + 0.5\gamma \cdot d) \\
y &= 3 + 0.5\gamma + 0.25\gamma^2 y + 0.25\gamma^2 \cdot d \\
y - 0.25\gamma^2 y &= 3 + 0.5\gamma + 0.25\gamma^2 \cdot d \\
y(1 - 0.25\gamma^2) &= 3 + 0.5\gamma + 0.25\gamma^2 \cdot d \\
y &= \frac{3 + 0.5\gamma + 0.25\gamma^2 \cdot d}{(1 - 0.25\gamma^2)}
\end{aligned}$$

Thus, the values of $V^{\pi_0}(s_0)$ and $V^{\pi_0}(s_1)$ are:

$$V^{\pi_0}(s_0) = \frac{1 + 1.5\gamma + 0.5\gamma \cdot d}{(1 - 0.25\gamma^2)} \tag{A.5}$$

$$V^{\pi_0}(s_1) = \frac{3 + 0.5\gamma + 0.25\gamma^2 \cdot d}{(1 - 0.25\gamma^2)} \tag{A.6}$$

Computing the policy π_1

Computing $V^{\pi_1}(s_0)$:

$$\begin{aligned}
V^{\pi_1}(s_0) &= \mathcal{C}(s_0, a_1) + \gamma(\mathcal{T}(s_0, a_1, s_2)V^{\pi_1}(s_2)) \\
V^{\pi_1}(s_0) &= 1 + \gamma(1 \cdot V^{\pi_1}(s_2)) \\
V^{\pi_1}(s_0) &= 1 + \gamma V^{\pi_1}(s_2)
\end{aligned}$$

Computing $V^{\pi_1}(s_2)$:

$$\begin{aligned}
&V^{\pi_1}(s_2) = \mathcal{C}(s_2, a_1) + \gamma(\mathcal{T}(s_2, a_1, s_0)V^{\pi_1}(s_0) + \mathcal{T}(s_2, a_1, d_2)V^{\pi_1}(d_2) + \mathcal{T}(s_2, a_1, s_g)V^{\pi_1}(s_g))\\
&V^{\pi_1}(s_2) = 2 + \gamma(0.25V^{\pi_1}(s_0) + 0.5V^{\pi_1}(d_2) + 0.25V^{\pi_1}(s_g))\\
&V^{\pi_1}(s_2) = 2 + \gamma(0.25V^{\pi_1}(s_0) + 0.5V^{\pi_1}(d_2) + 0.25 \cdot 0)\\
&V^{\pi_1}(s_2) = 2 + \gamma(0.25V^{\pi_1}(s_0) + 0.5V^{\pi_1}(d_2))\\
&V^{\pi_1}(s_2) = 2 + 0.25\gamma V^{\pi_1}(s_0) + 0.5\gamma V^{\pi_1}(d_2)
\end{aligned}$$

Computing $V^{\pi_1}(d_2)$:

$$\begin{aligned}
&V^{\pi_1}(d_2) = \mathcal{C}(d_2, a_1) + \gamma(\mathcal{T}(d_2, a_1, d_3)V^{\pi_1}(d_3))\\
&V^{\pi_1}(d_2) = d + \gamma(1.0 \cdot V^{\pi_1}(d_3))\\
&V^{\pi_1}(d_2) = d + \gamma(V^{\pi_1}(d_3))\\
&V^{\pi_1}(d_2) = d + \gamma V^{\pi_1}(d_3)
\end{aligned}$$

Computing $V^{\pi_1}(d_3)$:

$$\begin{aligned}
&V^{\pi_1}(d_3) = \mathcal{C}(d_3, a_1) + \gamma(\mathcal{T}(d_3, a_1, d_2)V^{\pi_1}(d_2))\\
&V^{\pi_1}(d_3) = d + \gamma(1.0 \cdot V^{\pi_1}(d_2))\\
&V^{\pi_1}(d_3) = d + \gamma(V^{\pi_1}(d_2))\\
&V^{\pi_1}(d_3) = d + \gamma V^{\pi_1}(d_2)
\end{aligned}$$

We do have the following system of equations:

$$\begin{aligned}
&V^{\pi_1}(s_0) = 1 + \gamma V^{\pi_1}(s_2)\\
&V^{\pi_1}(s_2) = 2 + 0.25\gamma V^{\pi_1}(s_0) + 0.5\gamma V^{\pi_1}(d_2)\\
&V^{\pi_1}(d_2) = d + \gamma V^{\pi_1}(d_3)\\
&V^{\pi_1}(d_3) = d + \gamma V^{\pi_1}(d_2)
\end{aligned}$$

We rename $V^{\pi_1}(s_0)$ as x, $V^{\pi_1}(s_2)$ as y, $V^{\pi_1}(d_2)$ as z and $V^{\pi_1}(d_3)$ as w. Then, we do have the following system of equations:

$$\begin{aligned}
x &= 1 + \gamma y \\
y &= 2 + 0.25\gamma x + 0.5\gamma z \\
z &= d + \gamma w \\
w &= d + \gamma z
\end{aligned}$$

As z and w are dead ends, we compute z:

$$\begin{aligned}
z &= d + \gamma w \\
z &= d + \gamma(d + \gamma z) \\
z &= d + \gamma d + \gamma^2 z \\
z - \gamma^2 z &= d + \gamma d \\
z(1 - \gamma^2) &= d + \gamma d \\
z(1 - \gamma^2) &= d(1 + \gamma) \\
z &= \frac{d(1 + \gamma)}{(1 - \gamma^2)} \\
z &= \frac{d(1 + \gamma)}{(1 + \gamma)(1 - \gamma)} \\
z &= \frac{d}{(1 - \gamma)}
\end{aligned}$$

Computing w:

$$\begin{aligned}
w &= d + \gamma z \\
w &= d + \gamma(d + \gamma w) \\
w &= d + \gamma d + \gamma^2 w \\
w - \gamma^2 z &= d + \gamma d \\
w(1 - \gamma^2) &= d + \gamma d \\
w(1 - \gamma^2) &= d(1 + \gamma) \\
w &= \frac{d(1 + \gamma)}{(1 - \gamma^2)} \\
w &= \frac{d(1 + \gamma)}{(1 + \gamma)(1 - \gamma)} \\
w &= \frac{d}{(1 - \gamma)}
\end{aligned}$$

Computing x:

$$\begin{aligned}
x &= 1 + \gamma y \\
x &= 1 + \gamma(2 + 0.25\gamma x + 0.5\gamma z) \\
x &= 1 + 2\gamma + 0.25\gamma^2 x + 0.5\gamma^2 z \\
x - 0.25\gamma^2 x &= 1 + 2\gamma + 0.5\gamma^2 z \\
x(1 - 0.25\gamma^2) &= 1 + 2\gamma + 0.5\gamma^2 z \\
x(1 - 0.25\gamma^2) &= 1 + 2\gamma + 0.5\gamma^2(\frac{d}{(1 - \gamma)}) \\
x &= \frac{1 + 2\gamma + 0.5\gamma^2(\frac{d}{(1-\gamma)})}{(1 - 0.25\gamma^2)}
\end{aligned}$$

Computing y:

$$\begin{aligned}
y &= 2 + 0.25\gamma x + 0.5\gamma z \\
y &= 2 + 0.25\gamma(1 + \gamma y) + 0.5\gamma z \\
y &= 2 + 0.25\gamma + 0.25\gamma^2 y + 0.5\gamma z \\
y - 0.25\gamma^2 y &= 2 + 0.25\gamma + 0.5\gamma z \\
y(1 - 0.25\gamma^2) &= 2 + 0.25\gamma + 0.5\gamma z \\
y(1 - 0.25\gamma^2) &= 2 + 0.25\gamma + 0.5\gamma(\frac{d}{(1-\gamma)}) \\
y &= \frac{2 + 0.25\gamma + 0.5\gamma(\frac{d}{(1-\gamma)})}{(1 - 0.25\gamma^2)}
\end{aligned}$$

Thus, the values of $V^{\pi_1}(s_0)$, $V^{\pi_1}(s_2)$, $V^{\pi_1}(d_2)$ and $V^{\pi_1}(d_3)$ are:

$$V^{\pi_1}(s_0) = \frac{1 + 2\gamma + 0.5\gamma^2(\frac{d}{(1-\gamma)})}{(1 - 0.25\gamma^2)} \tag{A.7}$$

$$V^{\pi_1}(s_2) = \frac{2 + 0.25\gamma + 0.5\gamma(\frac{d}{(1-\gamma)})}{(1 - 0.25\gamma^2)} \tag{A.8}$$

$$V^{\pi_1}(d_2) = \frac{d}{(1-\gamma)} \tag{A.9}$$

$$V^{\pi_1}(d_3) = \frac{d}{(1-\gamma)} \tag{A.10}$$

A.3 Computing details for Example 2

A.3.1 MCMP

Computing the occupation measure

$$\begin{aligned}
x^{\pi_0}_{s_g} &= p^{\max} = \boxed{\frac{1}{3}} \\
x^{\pi_0}_{s_1} &= \frac{x^{\pi_1}_{s_g}}{\mathcal{T}(s_1, a_0, s_g)} = \frac{1/3}{0.5} = \boxed{\frac{2}{3}} \\
x^{\pi_0}_{s_0} &= \frac{x^{\pi_1}_{s_1}}{\mathcal{T}(s_0, a_0, s_1)} = \frac{2/3}{0.5} = \boxed{\frac{4}{3}}
\end{aligned}$$

$$
\begin{aligned}
x^{\pi_1}_{s_g} &= p^{\max} = \boxed{\frac{1}{3}} \\
x^{\pi_1}_{s_2} &= \frac{x^{\pi_1}_{s_g}}{\mathcal{T}(s_2, a_1, s_g)} = \frac{1/3}{0.25} = \boxed{\frac{4}{3}} \\
x^{\pi_1}_{s_0} &= \frac{x^{\pi_1}_{s_2}}{\mathcal{T}(s_0, a_1, s_2)} = \frac{4/3}{1.0} = \boxed{\frac{4}{3}}
\end{aligned}
$$

$$
\begin{aligned}
x^{\pi_2}_{s_g} &= p^{\max} = \boxed{\frac{1}{4}} \\
x^{\pi_2}_{s_3} &= \frac{x^{\pi_2}_{s_g}}{\mathcal{T}(s_3, a_2, s_g)} = \frac{1/4}{0.25} = \boxed{1} \\
x^{\pi_2}_{s_0} &= \frac{x^{\pi_2}_{s_2}}{\mathcal{T}(s_0, a_2, s_3)} = \frac{1.0}{0.4} = \boxed{\frac{10}{4}}
\end{aligned}
$$

Computing the value function

$$
\begin{aligned}
V^{\pi_0}_{MCMP}(s_0) &= \frac{4}{3} + 3 \cdot \frac{2}{3} = \frac{10}{3} = \boxed{3.34} \\
V^{\pi_1}_{MCMP}(s_0) &= \frac{4}{3}(1 + 2) = \boxed{4} \\
V^{\pi_2}_{MCMP}(s_0) &= 1 \cdot 1 + \frac{10}{4} \cdot 1 = \frac{10}{4} = \boxed{2.5}
\end{aligned}
$$

A.3.2 IHD

Now, we can compute the value function for the policy π_2.

Computing the policy π_2

Computing $V^{\pi_2}_{\gamma}(s_0)$:

$$
\begin{aligned}
V^{\pi_2}_{\gamma}(s_0) &= C(s_0, a_2) + \gamma(\mathcal{T}(s_0, a_2, d_4) \cdot V(d_4) + \mathcal{T}(s_0, a_2, s_3) \cdot V(s_3)) \\
V^{\pi_2}_{\gamma}(s_0) &= 1 + \gamma(0.6 \cdot 0 + 0.4 \cdot V(s_3)) \\
V^{\pi_2}_{\gamma}(s_0) &= 1 + \gamma(0.4V(s_3)) \\
V^{\pi_2}_{\gamma}(s_0) &= 1 + 0.4\gamma V(s_3)
\end{aligned}
$$

Computing $V^{\pi_2}_{\gamma}(s_3)$:

$$\begin{aligned}
V_\gamma^{\pi_2}(s_3) &= \mathcal{C}(s_3, a_2) + \gamma(\mathcal{T}(s_3, a_2, s_0) \cdot V(s_0) + \mathcal{T}(s_3, a_2, s_g) \cdot V(s_g)) \\
V_\gamma^{\pi_2}(s_3) &= 1 + \gamma(0.5 \cdot V(s_0) + 0.5 \cdot 0) \\
V_\gamma^{\pi_2}(s_3) &= 1 + \gamma(0.5 \cdot V(s_0)) \\
V_\gamma^{\pi_2}(s_3) &= 1 + 0.5\gamma V(s_0)
\end{aligned}$$

We do have the following system of equations:

$$\begin{aligned}
V_\gamma^{\pi_2}(s_0) &= 1 + 0.4\gamma V(s_3) \\
V_\gamma^{\pi_2}(s_3) &= 1 + 0.5\gamma V(s_0)
\end{aligned}$$

We can rename $V_\gamma^{\pi_2}(s_0)$ as x and $V_\gamma^{\pi_2}(s_3)$ as y, then, we can solve the following system of equations:

$$\begin{aligned}
x &= 1 + 0.4\gamma y \\
y &= 1 + 0.5\gamma x
\end{aligned}$$

Computing x:

$$\begin{aligned}
x &= 1 + 0.4\gamma y \\
x &= 1 + 0.4\gamma(1 + 0.5\gamma x) \\
x &= 1 + 0.4\gamma + 0.2\gamma^2 x \\
x - 0.2\gamma^2 x &= 1 + 0.4\gamma \\
x(1 - 0.2\gamma^2) &= 1 + 0.4\gamma \\
x &= \frac{1 + 0.4\gamma}{(1 - 0.2\gamma^2)}
\end{aligned}$$

Computing y:

$$\begin{aligned}
y &= 1 + 0.5\gamma x \\
y &= 1 + 0.5\gamma(1 + 0.4\gamma y) \\
y &= 1 + 0.5\gamma + 0.2\gamma^2 y \\
y - 0.2\gamma^2 y &= 1 + 0.5\gamma \\
y(1 - 0.2\gamma^2) &= 1 + 0.5\gamma \\
y &= \frac{1 + 0.5\gamma}{(1 - 0.2\gamma^2)}
\end{aligned}$$

Thus, the values of $V_\gamma^{\pi_2}(s_0)$ e $V_\gamma^{\pi_2}(s_3)$ are:

$$V_\gamma^{\pi_2}(s_0) = \frac{1 + 0.4\gamma}{(1 - 0.2\gamma^2)} \tag{A.11}$$

$$V_\gamma^{\pi_2}(s_3) = \frac{1 + 0.5\gamma}{(1 - 0.2\gamma^2)} \tag{A.12}$$

References

[Altman 1999] Eitan Altman. *Constrained Markov Decision Processes.* CRC Press, Mar. 1999. isbn: 978-0-8493-0382-1 (cit. on p. 16).

[Amin *et al.* 2021] Susan Amin, Maziar Gomrokchi, Harsh Satija, Herke van Hoof, and Doina Precup. *A Survey of Exploration Methods in Reinforcement Learning.* Sept. 2021. doi: 10.48550/arXiv.2109.00157. arXiv: 2109.00157 [cs] (cit. on p. 44).

[Amodei *et al.* 2016] Dario Amodei *et al. Concrete Problems in AI Safety.* July 2016. arXiv: 1606.06565 [cs] (cit. on p. 55).

[Andrés *et al.* 2018] Ignasi Andrés, Leliane Nunes de Barros, Denis D. Mauá, and Thiago D. Simão. "When a Robot Reaches Out for Human Help". In: *Advances in Artificial Intelligence – IBERAMIA 2018.* Cham: Springer International Publishing, 2018, pp. 277–289. isbn: 978-3-030-03928-8. doi: 10.1007/978-3-030-03928-8_23 (cit. on p. 57).

[Aubret *et al.* 2019] Arthur Aubret, Laetitia Matignon, and Salima Hassas. *A Survey on Intrinsic Motivation in Reinforcement Learning.* Nov. 2019. doi: 10.48550/arXiv.1908.06976. arXiv: 1908.06976 [cs] (cit. on p. 49).

[Bahar *et al.* 1993] R.I. Bahar *et al.* "Algebraic decision diagrams and their applications". In: *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD).* Nov. 1993, pp. 188–191. doi: 10.1109/ICCAD.1993.580054 (cit. on p. 31).

[Barto 2013] Andrew G. Barto. "Intrinsic Motivation and Reinforcement Learning". In: *Intrinsically Motivated Learning in Natural and Artificial Systems.* Ed. by Gianluca Baldassarre and Marco Mirolli. Berlin, Heidelberg: Springer, 2013, pp. 17–47. isbn: 978-3-642-32375-1. doi: 10.1007/978-3-642-32375-1_2 (cit. on p. 49).

[Barto *et al.* 1995] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. "Learning to act using real-time dynamic programming". *Artificial Intelligence* 72.1 (Jan. 1995), pp. 81–138. issn: 0004-3702. doi: 10.1016/0004-3702(94)00011-O (cit. on pp. 29, 31).

[Bellman 1957] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957. isbn: 978-0-691-07951-6 (cit. on p. 13).

[D. Bertsekas 2019] Dimitri Bertsekas. *Reinforcement Learning and Optimal Control.* Athena Scientific, July 2019. isbn: 978-1-886529-39-7 (cit. on p. 54).

[D. Bertsekas 2022] Dimitri Bertsekas. *Abstract Dynamic Programming: 3rd Edition.* Athena Scientific, Jan. 2022. isbn: 978-1-886529-47-2 (cit. on p. 17).

[D. Bertsekas 2023] Dimitri Bertsekas. *A Course in Reinforcement Learning.* Athena Scientific, June 2023. isbn: 978-1-886529-49-6 (cit. on pp. xi, 1, 3, 4, 45, 68).

[D. P. Bertsekas and J. N. Tsitsiklis 1991] Dimitri P. Bertsekas and John N. Tsitsiklis. "An Analysis of Stochastic Shortest Path Problems". *Mathematics of Operations Research* 16.3 (Aug. 1991), pp. 580–595. issn: 0364-765X. doi: 10.1287/moor.16.3.580 (cit. on pp. 1, 2, 17).

[D. P. Bertsekas, J. N. Tsitsiklis, and J. Tsitsiklis 1996] Dimitri P. Bertsekas, John N. Tsitsiklis, and John Tsitsiklis. *Neuro-Dynamic Programming.* 1st edition. Belmont, Mass: Athena Scientific, May 1996. isbn: 978-1-886529-10-6 (cit. on p. 17).

[Bonet and Geffner 2003] Blai Bonet and Hector Geffner. "Faster heuristic search algorithms for planning with uncertainty and full feedback". In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence.* IJCAI'03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Aug. 2003, pp. 1233–1238 (cit. on pp. 29, 30).

[Boutilier *et al.* 2000] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. "Stochastic dynamic programming with factored representations". *Artificial Intelligence* 121.1 (Aug. 2000), pp. 49–107. issn: 0004-3702. doi: 10.1016/S0004-3702(00)00033-3 (cit. on p. 31).

[Brockman *et al.* 2016] Greg Brockman *et al. OpenAI Gym.* June 2016. doi: 10.48550/arXiv.1606.01540. arXiv: 1606.01540 [cs] (cit. on p. 59).

[Bryant 1986] Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". *IEEE Transactions on Computers* C-35.8 (Aug. 1986), pp. 677–691. issn: 1557-9956. doi: 10.1109/TC.1986.1676819 (cit. on p. 31).

[Bryce and Buffet 2008] Daniel Bryce and Olivier Buffet. "6th International Planning Competition: Uncertainty Part". In: *Proceedings of the 6th International Planning Competition (IPC'08).* 2008 (cit. on pp. 6, 59).

[Dai *et al.* 2011] P. Dai, Mausam, D. S. Weld, and J. Goldsmith. "Topological Value Iteration Algorithms". *Journal of Artificial Intelligence Research* 42 (Oct. 2011), pp. 181–209. issn: 1076-9757. doi: 10.1613/jair.3390 (cit. on p. 15).

[Karina V. Delgado *et al.* 2016] Karina V. Delgado, Leliane N. de Barros, Daniel B. Dias, and Scott Sanner. "Real-time dynamic programming for Markov decision processes with imprecise probabilities". *Artificial Intelligence* 230 (Jan. 2016), pp. 192–223. issn: 0004-3702. doi: 10.1016/j.artint.2015.09.005 (cit. on p. 31).

[Karina Valdivia Delgado *et al.* 2010] Karina Valdivia Delgado, Cheng Fang, Scott Sanner, and Leliane Nunes de Barros. "Symbolic Bounded Real-Time Dynamic Programming". In: *Advances in Artificial Intelligence – SBIA 2010*. Ed. by Antônio Carlos da Rocha Costa, Rosa Maria Vicari, and Flavio Tonidandel. Berlin, Heidelberg: Springer, 2010, pp. 193–202. isbn: 978-3-642-16138-4. doi: 10.1007/978-3-642-16138-4_20 (cit. on p. 31).

[Fatemi and Sharma 2019] Mehdi Fatemi and Shikhar Sharma. "Dead-ends and Secure Exploration in Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning*. ICML '19. 2019 (cit. on p. 55).

[Feng *et al.* 2002] Zhengzhu Feng, Eric A. Hansen, and Shlomo Zilberstein. "Symbolic generalization for on-line planning". In: *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*. UAI'03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Aug. 2002, pp. 209–216. isbn: 978-0-12-705664-7 (cit. on p. 31).

[Forrest and Lougee-Heimer 2005] John Forrest and Robin Lougee-Heimer. "CBC User Guide". In: *Emerging Theory, Methods, and Applications*. INFORMS TutORials in Operations Research. INFORMS, Sept. 2005. Chap. 10, pp. 257–277. isbn: 978-1-877640-21-6. doi: 10.1287/educ.1053.0020 (cit. on p. 65).

[Freire and Karina Valdivia Delgado 2017] Valdinei Freire and Karina Valdivia Delgado. "GUBS: a Utility-Based Semantic for Goal-Directed Markov Decision Processes". In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '17. 2017 (cit. on pp. 5, 22, 28).

[Freire, Karina Valdivia Delgado, and Reis 2019] Valdinei Freire, Karina Valdivia Delgado, and Willy Arthur Silva Reis. "An Exact Algorithm to Make a Trade-Off between Cost and Probability in SSPs". *Proceedings of the International Conference on Automated Planning and Scheduling* 29 (2019), pp. 146–154. issn: 2334-0843. doi: 10.1609/icaps.v29i1.3470 (cit. on p. 28).

[Garcia and Fernandez 2015] Javier Garcia and Fernando Fernandez. "A Comprehensive Survey on Safe Reinforcement Learning". *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480 (cit. on p. 55).

[Geffner and Bonet 2013] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, June 2013. isbn: 978-1-60845-970-4 (cit. on pp. 3, 20, 59).

[Gelly and D. Silver 2011] Sylvain Gelly and David Silver. “Monte-Carlo tree search and rapid action value estimation in computer Go”. *Artificial Intelligence* 175.11 (July 2011), pp. 1856–1875. issn: 0004-3702. doi: 10.1016/j.artint.2011.03.007 (cit. on p. 30).

[Ghallab *et al.* 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, May 2004. isbn: 978-1-55860-856-6 (cit. on p. 2).

[Gu *et al.* 2023] Shangding Gu *et al. A Review of Safe Reinforcement Learning: Methods, Theory and Applications*. Feb. 2023. arXiv: 2205.10330 [cs] (cit. on p. 55).

[Hansen and Zilberstein 2001] Eric A. Hansen and Shlomo Zilberstein. “LAO∗: A heuristic search algorithm that finds solutions with loops”. *Artificial Intelligence* 129.1 (June 2001), pp. 35–62. issn: 0004-3702. doi: 10.1016/S0004-3702(01)00106-0 (cit. on pp. 29, 32).

[H. Hasselt 2010] Hado Hasselt. “Double Q-learning”. In: *Advances in Neural Information Processing Systems*. Vol. 23. Curran Associates, Inc., 2010 (cit. on p. 51).

[H. P. v. Hasselt 2011] Hado Philip van Hasselt. “Insights in Reinforcement Rearning : Formal Analysis and Empirical Evaluation of Temporal-Difference Learning Algorithms”. PhD thesis. Utrecht University, Netherlands, 2011 (cit. on p. 51).

[Hayes, Rădulescu, Bargiacchi, Kallstrom, *et al.* 2023] Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Kallstrom, *et al.* “A Brief Guide to Multi-Objective Reinforcement Learning and Planning”. In: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS ’23. International Foundation for Autonomous Agents and Multiagent Systems, May 2023, pp. 1988–1990. isbn: 978-1-4503-9432-1 (cit. on p. 55).

[Hayes, Rădulescu, Bargiacchi, Källström, *et al.* 2022] Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, *et al.* “A practical guide to multi-objective reinforcement learning and planning”. *Autonomous Agents and Multi-Agent Systems* 36.1 (Apr. 2022), p. 26. issn: 1573-7454. doi: 10.1007/s10458-022-09552-y (cit. on p. 55).

[Hoey *et al.* 1999] J. Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. “SPUDD: Stochastic Planning using Decision Diagrams”. *ArXiv* (July 1999) (cit. on p. 31).

[Howard 1960] Ronald A. Howard. *Dynamic Programming and Markov Processes*. Technology Press of the Massachusetts Institute of Technology, 1960. isbn: 978-0-262-08009-5 (cit. on p. 14).

[Jafarnia-Jahromi *et al.* 2023] Mehdi Jafarnia-Jahromi, Liyu Chen, Rahul Jain, and Haipeng Luo. “Posterior sampling-based online learning for the stochastic shortest path model”. In: *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence.* Vol. 216. UAI ’23. Pittsburgh, PA, USA: JMLR.org, July 2023, pp. 922–931 (cit. on p. 54).

[Jaksch *et al.* 2010] Thomas Jaksch, Ronald Ortner, and Peter Auer. “Near-optimal Regret Bounds for Reinforcement Learning”. *Journal of Machine Learning Research* 11 (2010), pp. 1563–1600 (cit. on p. 54).

[Jansen *et al.* 2020] Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem. “Safe Reinforcement Learning Using Probabilistic Shields (Invited Paper)” (2020), 16 pages, 3444435 bytes. issn: 1868-8969. doi: 10.4230/LIPICS.CONCUR.2020.3 (cit. on p. 55).

[Kakade 2003] Sham Machandranath Kakade. “On the Sample Complexity of Reinforcement Learning”. PhD thesis. University College London, 2003 (cit. on p. 45).

[Keller and Eyerich 2012] Thomas Keller and Patrick Eyerich. “PROST: Probabilistic Planning Based on UCT”. *Proceedings of the International Conference on Automated Planning and Scheduling* 22 (May 2012), pp. 119–127. issn: 2334-0843. doi: 10.1609/icaps.v22i1.13518 (cit. on p. 30).

[Kocsis and Szepesvári 2006] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning”. In: *Machine Learning: ECML 2006.* Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Berlin, Heidelberg: Springer, 2006, pp. 282–293. isbn: 978-3-540-46056-5. doi: 10.1007/11871842_29 (cit. on p. 30).

[Kolobov 2013] Andrey Kolobov. “Scalable Methods and Expressive Models for Planning Under Uncertainty”. Thesis. July 2013 (cit. on pp. 3, 51).

[Kolobov, M. Mausam, *et al.* 2011] Andrey Kolobov, Mausam Mausam, Daniel Weld, and Hector Geffner. “Heuristic Search for Generalized Stochastic Shortest Path MDPs”. *Proceedings of the International Conference on Automated Planning and Scheduling* 21 (Mar. 2011), pp. 130–137. issn: 2334-0843. doi: 10.1609/icaps.v21i1.13452 (cit. on pp. 2, 20, 23, 24, 29, 30, 51, 59).

[Kolobov and Weld 2012] Andrey Kolobov and Daniel S Weld. “A Theory of Goal-Oriented MDPs with Dead Ends”. In: *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence.* UAI’12. Oct. 2012 (cit. on pp. 1, 2, 5, 17, 20, 22–24, 33, 47).

[Levine *et al.* 2020] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.* Nov. 2020. doi: 10.48550/arXiv.2005.01643. arXiv: 2005.01643 [cs] (cit. on p. 45).

[Lin 1992] Long-Ji Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". *Mach. Learn.* 8.3-4 (May 1992), pp. 293–321. issn: 0885-6125. doi: 10.1007/BF00992699 (cit. on pp. 45, 51).

[Liu *et al.* 2022] Minghuan Liu, Menghui Zhu, and Weinan Zhang. *Goal-Conditioned Reinforcement Learning: Problems and Solutions.* Sept. 2022. arXiv: 2201.08299 [cs] (cit. on p. 55).

[Mausam and Kolobov 2012] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective.* Morgan & Claypool Publishers, 2012. isbn: 978-1-60845-886-8 (cit. on pp. 2–4, 9, 17, 32, 33).

[Min *et al.* 2022] Yifei Min, Jiafan He, Tianhao Wang, and Quanquan Gu. "Learning Stochastic Shortest Path with Linear Function Approximation". In: *Proceedings of the 39th International Conference on Machine Learning.* ICML '22. 2022 (cit. on p. 54).

[Mitchell *et al.* 2011] S. Mitchell, Michael O'Sullivan, and Iain Dunning. "PuLP : A Linear Programming Toolkit for Python". In: The University of Auckland, Auckland, New Zealand, 2011 (cit. on p. 65).

[Mnih *et al.* 2015] Volodymyr Mnih *et al.* "Human-level control through deep reinforcement learning". *Nature* 518.7540 (Feb. 2015), pp. 529–533. issn: 0028-0836, 1476-4687. doi: 10.1038/nature14236 (cit. on pp. 45, 51).

[Nilsson 1982] Nils J. Nilsson. *Principles of Artificial Intelligence.* Springer Science & Business Media, May 1982. isbn: 978-3-540-11340-9 (cit. on p. 29).

[Puterman 2014] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Aug. 2014. isbn: 978-1-118-62587-3 (cit. on pp. 1, 9, 11, 13).

[Roijers *et al.* 2013] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. "A Survey of Multi-Objective Sequential Decision-Making". *Journal of Artificial Intelligence Research* 48 (Oct. 2013), pp. 67–113. issn: 1076-9757. doi: 10.1613/jair.3987 (cit. on p. 55).

[Russell *et al.* 2010] Stuart Russell, Stuart Jonathan Russell, Peter Norvig, and Ernest Davis. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2010. isbn: 978-0-13-604259-4 (cit. on p. 44).

[Sanner, Karina Valdivia Delgado, *et al.* 2011] Scott Sanner, Karina Valdivia Delgado, and Leliane Nunes de Barros. "Symbolic dynamic programming for discrete and continuous state MDPs". In: *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence.* UAI'11. Arlington, Virginia, USA: AUAI Press, July 2011, pp. 643–652. isbn: 978-0-9749039-7-2 (cit. on p. 31).

[Sanner and Yoon 2011] Scott Sanner and Sungwook Yoon. “IPPC Results Presentation” (2011) (cit. on pp. 30, 57).

[T. Silver and Chitnis 2020] Tom Silver and Rohan Chitnis. *PDDLGym: Gym Environments from PDDL Problems.* Sept. 2020. doi: 10.48550/arXiv.2002.06432. arXiv: 2002.06432 [cs] (cit. on p. 59).

[T. D. Simão 2023] T. D. Simão. “Safe Online and Offline Reinforcement Learning”. PhD thesis. 2023 (cit. on p. 55).

[Thiago D. Simão *et al.* 2021] Thiago D. Simão, Nils Jansen, and Matthijs TJ Spaan. “AlwaysSafe: Reinforcement learning without safety constraint violations during training”. In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems.* AAMAS ’21. New York: ACM, 2021 (cit. on p. 55).

[Singh *et al.* 2000] Satinder Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. “Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms”. *Machine Learning* 38.3 (Mar. 2000), pp. 287–308. issn: 1573-0565. doi: 10.1023/A:1007678930559 (cit. on p. 44).

[Steinmetz *et al.* 2016] Marcel Steinmetz, Jörg Hoffmann, and Olivier Buffet. “Goal Probability Analysis in Probabilistic Planning: Exploring and Enhancing the State of the Art”. *Journal of Artificial Intelligence Research* 57 (Oct. 2016), pp. 229–271. issn: 1076-9757. doi: 10.1613/jair.5153 (cit. on pp. 24, 30).

[Alexander L Strehl *et al.* 2006] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. “PAC Model-Free Reinforcement Learning” (2006) (cit. on p. 45).

[Alexander L. Strehl *et al.* 2009] Alexander L. Strehl, Lihong Li, and Michael L. Littman. “Reinforcement Learning in Finite MDPs: PAC Analysis”. *Journal of Machine Learning Research* 10.84 (2009), pp. 2413–2444. issn: 1533-7928 (cit. on p. 45).

[Sutton and Barto 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, Second Edition: An Introduction.* MIT Press, Nov. 2018. isbn: 978-0-262-03924-6 (cit. on pp. xi, 9, 43, 44, 51).

[Tarbouriech 2022] Jean Tarbouriech. “Goal-Oriented Exploration for Reinforcement Learning”. PhD thesis. 2022 (cit. on pp. 3, 54).

[Tarbouriech *et al.* 2020] Jean Tarbouriech, Evrard Garcelon, Michal Valko, Matteo Pirotta, and Alessandro Lazaric. *No-Regret Exploration in Goal-Oriented Reinforcement Learning.* Aug. 2020. arXiv: 1912.03517 [cs, stat] (cit. on p. 54).

[Teichteil-Königsbuch 2012] Florent Teichteil-Königsbuch. “Stochastic Safest and Shortest Path Problems”. *Proceedings of the AAAI Conference on Artificial Intelligence* 26.1 (2012), pp. 1825–1831. issn: 2374-3468, 2159-5399. doi: 10.1609/aaai.v26i1.8367 (cit. on pp. xii, 1, 2, 5, 17, 22, 24, 25, 33, 69).

[Thrun 1992] Sebastian B. Thrun. *Efficient Exploration In Reinforcement Learning*. Technical Report. USA: Carnegie Mellon University, 1992 (cit. on p. 44).

[Towers *et al.* 2024] Mark Towers *et al. Gymnasium: A Standard Interface for Reinforcement Learning Environments*. July 2024. doi: 10.48550/arXiv.2407.17032. arXiv: 2407.17032 [cs] (cit. on p. 59).

[Trevizan *et al.* 2017] Felipe Trevizan, Florent Teichteil-Konigsbuch, and Sylvie Thiebaux. “Efficient Solutions for Stochastic Shortest Path Problems with Dead Ends”. In: *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*. UAI’17. 2017 (cit. on pp. xi, 1, 2, 5, 17, 19, 22, 24, 26, 33, 35–37, 40, 48, 49, 72).

[J. N. Tsitsiklis 1994] John N. Tsitsiklis. “Asynchronous Stochastic Approximation and Q-Learning”. *Machine Learning* 16.3 (Sept. 1994), pp. 185–202. issn: 1573-0565. doi: 10.1023/A:1022689125041 (cit. on pp. 44, 46, 54).

[Vamplew *et al.* 2011] Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. “Empirical evaluation methods for multiobjective reinforcement learning algorithms”. *Machine Learning* 84.1 (July 2011), pp. 51–80. issn: 1573-0565. doi: 10.1007/s10994-010-5232-5 (cit. on p. 28).

[van der Walt *et al.* 2011] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. “The NumPy Array: A Structure for Efficient Numerical Computation”. *Computing in Science & Engineering* 13.2 (Mar. 2011), pp. 22–30. issn: 1558-366X. doi: 10.1109/MCSE.2011.37 (cit. on p. 65).

[Watkins and Dayan 1992] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. *Machine Learning* 8.3 (May 1992), pp. 279–292. issn: 1573-0565. doi: 10.1007/BF00992698 (cit. on p. 43).

[Wiering and Otterlo 2012] Marco Wiering and Martijn van Otterlo. *Reinforcement Learning: State-of-the-Art*. Springer Berlin Heidelberg, Mar. 2012. isbn: 978-3-642-27644-6 (cit. on p. 45).

[Yu and D. P. Bertsekas 2013] Huizhen Yu and Dimitri P. Bertsekas. “On Boundedness of Q-Learning Iterates for Stochastic Shortest Path Problems”. *Mathematics of Operations Research* 38.2 (May 2013), pp. 209–227. issn: 0364-765X, 1526-5471. doi: 10.1287/moor.1120.0562 (cit. on p. 54).

Index