Heuristics based on Projection Occupation Measures for Probabilistic Planning with dead-ends and Risk

Milton Raúl Condori Fernández

MASTER THESIS DISSERTATION SUBMITTED
TO THE
INSTITUTE OF MATHEMATICS AND STATISTICS
OF THE
THE UNIVERSITY OF SÃO PAULO

Program: Computer Science

Adviser: Prof. Ph.D Leliane Nunes de Barros

During the development of this project the author received a scholarship from CAPES

São Paulo, 2020

Abstract

Condori, F. M. **Heuristics based on Projection Occupation Measures for Probabilistic Planning with Dead Ends and Risk**. 2020. Dissertation (Master's Degree) - Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2020.

In probabilistic planning an agent interacts with an environment and the objective is to find an optimal policy (state-action mapping) that allows the agent to achieve a goal state from an initial state, while minimizing the expected accumulated cost. Efficient solutions for large instances of probabilistic planning are, in general, based on Stochastic Shortest Path (SSP) problems, and use heuristic search techniques. However, these approaches have two limitations: (1) they can not guarantee to return optimal policies in the presence of dead-ends (states from which it is not possible to reach the goal) and (2) they may present a high variance in terms of cost. In instances where unavoidable dead-ends exist, we can plan in two phases: maximizing the probability to reach the goal (MAXPROB) and then minimizing the expected cost (MINCOST); or yet, we can define a penalty for reaching a dead-end state and only minimize the expected cost (MINCOST-WITH-PENALTY). While there exist several heuristics to solve the MINCOST problem, there are no efficient heuristics to solve MAXPROB problem. A recent work proposed the first heuristic that takes into account the probabilities, called h_{pom} , which solves a relaxed version of an SSP as a linear program in the dual space. In this work we propose two new heuristics based on h_{pom} to solve probabilistic planning problems with unavoidable dead-ends, that includes new variables and constraints for dead-end states. The first, $h_{ppom}(s)$, estimates the maximum probability to reach the goal from s , and is used to efficiently solve MAXPROB problems by ignoring action costs and considering only the probabilities. The second, used to solve MINCOST-WITH-PENALTY problems, called $h_{pom}^{pe}(s)$, estimates the minimal cost to reach the goal from state s and adds an expected penalty for reaching dead-ends. In order to deal with the second limitation of traditional SSP solutions, we propose a third heuristic, called h_{pom}^{rs} , also based on h_{pom} , for a modified version of an SSP, called risk sensitive SSP (RS-SSP), whose optimization criterion is to minimize an exponential utility function including a risk factor λ to characterize the agent attitude as: (i) risk-averse ($\lambda > 0$); (ii) risk-prone ($\lambda < 0$); or (iii) risk-neutral ($\lambda \rightarrow 0$). Empirical results show that the proposed heuristics can solve larger planning instances when compared to the state-of-the-art solutions for SSPs with dead-ends and RS-SSP problems.

Keywords: Probabilistic planning, Planning as heuristic search, Risk Sensitive SSP, SSP as dual linear program.

Contents

1	Introduction	5
1.1	Deterministic Planning	5
1.2	Probabilistic Planning	7
1.2.1	Classes of SSP problems in the presence of dead-ends	9
1.2.2	Risk sensitive SSP	9
1.3	Objective on this work	10
1.4	Contributions	10
1.5	Organization	10
I	Heuristics for Stochastic Shortest Path with Dead Ends	12
2	Background on Probabilistic Planning	13
2.1	Markov Decision Process	13
2.2	Finite Horizon MDP	14
2.3	Infinite Horizon MDP	15
2.4	Stochastic Shortest Path - SSP	16
2.5	Solutions for SSPs	18
2.5.1	Synchronous Algorithms for Probabilistic Planning	18
2.5.2	Asynchronous Algorithms for Probabilistic Planning	19
2.5.3	The algorithms RTDP and LRTDP	20
2.6	Stochastic Shortest Path with dead-ends	21
2.6.1	SSP with Avoidable dead-ends	22
2.6.2	SSP with Unavoidable dead-ends	23
2.7	Solutions for SSPs with dead-ends	24
2.7.1	Solving an SSP with avoidable dead-ends	24
2.7.2	Solving an SSP with unavoidable dead-ends	24
2.7.2.1	A solution with dead-end penalty	24
2.7.2.2	Maximum Goal Probability	25
2.7.2.3	The MAXPROB-MINCOST Criterion	25
2.7.3	Find, Revise and Eliminate Traps - FRET	27
3	Background on Heuristics for Probabilistic Planning	30
3.1	Planning problem description languages	30
3.1.1	Action languages based on predicates: PDDL and PPDDL	30
3.1.2	The SAS ⁺ task Representation	32
3.2	Determinization of a Probabilistic Planning Problem specified in PPDDL	32
3.3	Classical Heuristics	33

3.3.1	The h_{add} and h_{max} heuristics	33
3.3.2	Heuristic h_{FF}	34
3.3.3	Heuristic h_m	34
3.3.4	Heuristic h_{m-m}	35
3.3.5	Heuristic HMDPP	35
3.4	The Projection Occupation Measure Heuristic	36
3.4.1	An SSP as a Primal and Dual Linear Program	36
3.4.2	The Heuristic h_{pom}	39
3.4.3	Empirical results of the use of h_{pom} to solve SSPs e SSPs with unavoidable dead-ends	40
4	The Heuristics h_{pom}^{pe} and h_{ppom}	41
4.1	The Heuristic h_{pom}^{pe}	41
4.1.1	Linear Program for SSP with penalty	41
4.2	The Heuristic h_{ppom}	43
4.2.1	Linear Program for MaxProb	43
4.3	Example of h_{pom}^{pe} and h_{ppom} Computation	44
4.4	Empirical Analysis	46
4.4.1	Analysing the performance of h_{ppom}	47
4.4.2	Analysing the performance of h_{pom}^{pe}	48
4.5	Conclusions of Part I	52
II	Risk Sensitive Planning as a Linear Program in the Dual Space	53
5	Background on Risk Sensitive SSPs with Exponential Utility Function	54
5.1	Risk Sensitive Stochastic Shortest Path	54
5.2	The Risk in SSPs	56
5.2.1	Example 1	56
5.2.2	Example 2	56
5.3	Solutions for RS-SSP with exponential utility	57
5.3.1	A synchronous algorithm for RS-SSP: RS-PI	58
5.3.2	Asynchronous algorithms for RS-SSP	58
5.3.3	Finding a maximum λ -feasible	60
6	Risk Sensitive SSP as a Linear Program in the Dual Space	61
6.1	The Primal Linear Program for Risk Sensitive SSP	61
6.1.1	Primal LP with Implicit Goal Formulation	61
6.1.2	Primal LP with Explicit Goal Formulation	62
6.2	The Dual Linear Program for Risk Sensitive SSP	62
7	A Heuristic for RS-SSP: h_{pom}^{rs}	64
7.1	The heuristic h_{pom}^{rs}	64
7.2	Empirical Analysis	66
7.2.1	Planning Domains	67
7.2.2	Analysing the performance of h_{pom}^{rs}	67
7.3	Conclusions of Part II	71

8 Overall Conclusions and Future Work	73
8.1 Conclusions	73
8.2 Future Works	73
8.2.1 Sequential Search with Dual Linear Program (seqsearch + DLP)	83
8.2.2 Sequential Search with LRTDP (seqsearch + LRTDP)	83
8.2.3 Binary Search with Dual Linear Program (binsearch + DLP)	84
8.2.4 A Comparative analyzes of the Binary and Sequential Search for the extreme λ -feasible	85
8.2.5 Maximum λ -feasible for the domains tested in this work	89

List of Figures

1.1	Box-loader Domain.	5
1.2	State transition graph of the Box-loader Domain.	6
1.3	Box-loader domain represented by propositions.	6
1.4	The Box-Loader Domain in PDDL.	7
1.5	Action $grab(loc)$ in SAS ⁺	7
1.6	Probabilistic action.	8
1.7	Action $grab(loc)$ in PPDDL.	8
2.1	Interaction between an agent and his environment.	13
2.2	Example of an MDP represented as a directed hypergraph.	16
2.3	Example of an SSP with proper and improper policies.	17
2.4	Example of a greedy graph in an MDP.	18
2.5	Example of an SSP with dead-ends.	22
2.6	SSPs with avoidable and unavoidable dead-ends.	22
2.7	Pareto front policies of an SSPUDE.	26
3.1	Example of an SSP with deterministic actions and unitary costs	37
4.1	State space of a simple SSPUDE described as a SAS ⁺ task $\mathcal{T}$	44
4.2	Projection $\mathcal{M}^{v_1, s_0}$ for task $\mathcal{T}$	45
4.3	Projection $\mathcal{M}^{v_2, s_0}$ for task $\mathcal{T}$	45
5.1	Exponential utility function	55
5.2	Simple RS-SSP with a deterministic action a_1 and no dead-ends.	56
5.3	Exponential accumulated cost of policies in RS-SSPs.	57
5.4	Instance 7x10 of the River domain.	57
5.5	Extreme risk policies for instance 7×10 of the River Domain	58
7.1	State space of a simple SSP described as a SAS ⁺ task $\mathcal{T}$	65
7.2	Example of projections for states in h_{pom}^{rs} heuristic.	66
7.3	Analysing h_{pom}^{rs} : plots of Table 7.1.	68
7.4	Analysing h_{pom}^{rs} : plots of Table 7.2.	70
7.5	Analysing h_{pom}^{rs} : plots of Table 7.3.	70

List of Tables

3.1	Results for h_{pom} with SSPUDE (extracted from [Trevizan et al., 2017b])	40
4.1	Occupation measure values for LP3 with $pe = 10$	45
4.2	Cost function C_P of the SAS^+ task $\mathcal{T}$ in Fig. 4.2.	46
4.3	Occupation measure values for LP4.	46
4.4	Evaluating the heuristic h_{ppom} : computational time	49
4.5	Evaluating the heuristic h_{ppom} : number of states explored	50
4.6	Evaluating the heuristic h_{pom}^{pe}	51
7.1	Analysing h_{pom}^{rs} : average time (secs) to solve RS-SSP instances of the Triangle Tire World domain	68
7.2	Analysing h_{pom}^{rs} : average time (secs) to solve RS-SSP instances of the River domain	69
7.3	Analysing h_{pom}^{rs} : average time (secs) to solve RS-SSP instances of the Navigation domain	71
8.1	Extreme λ -feasible in the Triangle Tire World Domain: random transitions.	86
8.2	Extreme λ -feasible in the River Domain: random transitions.	87
8.3	Extreme λ -feasible in the Navigation Domain: random transitions.	88

List of Algorithms

1	Value Iteration Algorithm.	19
2	Find and Revise [Bonet and Geffner, 2003a].	19
3	LRTDP Algorithm [Bonet and Geffner, 2003b].	21
4	Find, Revise and Eliminate Traps Algorithm (adapted from [Kolobov et al., 2012]).	27
5	RS-PI	58
6	RS-LRTDP algorithm	59
7	SEQSEARCH + PI : Compute the maximum λ -feasible value	60
8	SEQSEARCH + DLP : Compute the maximum λ -feasible value	83
9	SEQSEARCH + LRTDP: Compute the maximum λ -feasible value	84
10	BINSEARCH + DLP	84

Chapter 1

Introduction

Artificial Intelligence (AI) research aims to study and develop intelligent agents capable of perceiving and acting in the world, while trying to optimize its performance. A fundamental aspect of an intelligent agent is the ability to autonomously plan for its actions in order to realize a task in environments with all sorts of dynamics and constraints. This area is called *automated planning* [Russell and Norvig, 2009], one of the oldest branch of AI, concerned with the construction of autonomous agents capable of generating plans (total or partial ordered set of actions) or policies (mapping from states to actions), that when executed makes the agent to achieve the task goal.

Planning agents appear in a broad set of applications, e.g. the Mars rover planning [Mausam et al., 2005], military operations planning [Aberdeen et al., 2004], robocup soccer planning [Stone et al., 2005]; Planning is also required in games like blackjack [Popyack, 2009], elevators operation [Crites and Barto, 1995], and intervention on bio-cellular processes [Bryce and Kim, 2006] [Tisovec et al., 2015], etc.

In the next sections, we describe two types of automated planning tasks that will be involved in this dissertation: deterministic and probabilistic planning.

1.1 Deterministic Planning

Among the sub-areas of automated planning, deterministic planning, also called *classical planning*, studies tasks involving a completely observable deterministic environment and a single agent, whose objective is to find a plan composed by a sequence of actions that takes the system (and the agent) from an initial state to a goal state.

Figure 1.1 shows a simple illustrative example of a classical planning domain, called Box-loader. In this problem example, there is a robot in an environment with two rooms, $L1$ and $L2$, and one box. The robot is able to move between rooms (actions $move(L1, L2)$ and $move(L2, L1)$) and also grab and drop the box (actions $grab$ and $drop$, respectively). Given that in the initial state the robot and the box are in room $L1$ and the robot is not holding the box, the goal is to have the box in the floor of room $L2$.

Figure 1.1: Example of a planning problem in the Box-loader Domain, involving two rooms and one box.

The dynamics of the environment can be described by a *state transition model* [Russell and Norvig, 2009], which can be represented explicitly as a directed graph, called *state transition graph*, in which the

vertices represent the states and the edges represent the possible actions the agent can execute. In this way, a deterministic planning problem can be seen as a shortest path problem in the state transition graph: given two vertices s (initial state) and s' (goal state), one must find the shortest path, that is, a sequence of edges (plan) connecting the vertex s to the vertex s' .

Figure 1.2: State transition graph of the Box-loader problem depicted in Fig. 1.1.

Figure 1.2 shows the state transition graph of the Box-loader domain with two rooms and one box, where the vertex s_1 represents the state in which the robot is in the right room and the box is in the left room; the vertex s_2 represents the state that the robot and the box are in the left room and the robot is not holding the box, and so on. The edges indicate a state transition, e.g., when the agent executes action $move(L2, L1)$ in state s_1 , the environment deterministically changes to state s_2 .

A state s_i can be represented by a set of *facts* (propositions) that describe the properties of the environment when it is in state s_i . In the Box-loader example, a state can be described by: the robot location ($robot-at(L1)$ or $robot-at(L2)$); the box location ($box-at(L1)$ or $box-at(L2)$); and if the box is held (or not) by the robot ($holding-box$ or $not\ holding-box$). We make the *closed-world assumption* (CWA), where a state is represented only by the set of facts that are true. Figure 1.3 shows the states' representation under the CWA assumption, e.g. state s_2 is described by the propositions $\{robot-at(L1), box-at(L1)\}$, which means the robot and the box are in room L1 and since $holding-box$ is not included, we infer that $holding-box$ is false, meaning the robot is not holding the box.

Figure 1.3: The Box-loader domain represented by the facts that describe the environment.

The state transition model can also be represented even more compactly through an *action language* [Fikes and Nilsson, 1971, Huang and Zhang, 2012, Ghallab et al., 1998]. The compactness comes from the fact that instead of representing each triple of the graph (state, action, state), the action language describes the preconditions and effects of the actions, commonly based on predicate logics.

Figure 1.4 uses a simplified version of PDDL action language [Ghallab et al., 1998] with conditional effects, describing the Box-loader domain. Notice that actions $move(L1, L2)$ and $move(L2, L1)$ are instances of $move(loc: x, loc: y)$; $grab(L1)$ and $grab(L2)$ are instances of $grab(loc: x)$; finally, $drop(L1)$ and $drop(L2)$ are instances of $drop(loc: x)$. Notice that the action language PDDL, after grounding, describes each action through three sets of facts: (i) **precondition**, list of facts that must be true in the state the action is executed; (ii) **add effects**, list of facts that become true after the action is executed; and (iii) **delete effects**, list of facts that become false after the action is executed. Note that the effects can also be conditioned to some state properties.

Figure 1.5 shows the action $grab(loc)$ described in the SAS⁺ action multi-valued language [Huang and Zhang, 2012]. In this language, instead of predicates we use multi-valued variables.

```

move(loc: x, loc: y)
  Precondition = robot-at(x)
  Effect = when holding-box
            then robot-at(y) and box-at(y) and not robot-at(x)
            else robot-at(y) and not robot-at(x)

grab(loc: x)
  Precondition = robot-at(x) and box-at(x)
  Effect = holding-box

drop(loc: x)
  Precondition = robot-at(x) and holding-box
  Effect = not holding-box

```

Figure 1.4: The Box-Loader Domain in PDDL - Planning Domain Description Language.

```

grab(loc:  $D_{robot-at}$ )
  #  $D_{robot-at} = \{L1, L2\}$ 
  Precondition: { robot-at = loc, box-at = loc, holding-box = False }
  Effect: { holding-box = True } with 0.9
           { holding-box = False } with 0.1

```

Figure 1.5: Action *grab*(loc) described in the action language SAS⁺.

A key aspect of research on planning is to reason about state transitions through an action language. State-of-the-art algorithms for classical planning [Bonet and Geffner, 2001] [Hoffmann, 2001] [Helmert, 2006] efficiently reason about actions in two ways: (1) to search for a plan without having to expand the entire state transition graph, and (2) to compute a heuristic based on a relaxed version of the problem description. For example, the FF planner (Fast-Forward) [Hoffmann, 2001], one of the most known planner, can solve classical planning problems with up to 10^{20} states [Edelkamp, 2000]. Heuristics for classical planning are also used for other types of planning problems, such as in probabilistic and non-deterministic planning.

1.2 Probabilistic Planning

In probabilistic planning the outcomes of actions are stochastic, and problems are modeled as a *Markov Decision Process* (MDP) [Puterman, 1994a]. An MDP models the interaction between an agent and its environment: at each stage t , the agent is in a state s_t (completely observable) and decides to execute an action a_t that will produce a future state s_{t+1} with probability p and a cost c_{t+1} .

MDPs are defined by a set of states S , a set of actions A , a cost function C , and a probabilistic transition function P . The number of stages that the agent acts in the environment is called horizon, which can be finite, infinite or indefinite. In general, the objective of an MDP agent is to find a policy π that maps states into actions, i.e. $\pi : S \rightarrow A$, that minimizes the expected accumulated cost over along horizon.

Another model for probabilistic planning is the Stochastic Shortest Path (SSP) [Bertsekas and Tsitsiklis, 1991]. An SSP is an MDP in which a set of goal states G and an initial state $s_0 \in S$ are known. In an SSP the agent stops acting upon reaching a goal state (or a dead-end state) and, thus, the agent's action horizon is undefined.

An SSP can be compiled from a domain described in a probabilistic action language. For instance,

Figure 1.6: Example of a probabilistic action for the Box-Loader domain.

<p>grab(loc: x)</p> <p><i>Precondition</i> = <i>robot-at(x)</i> and <i>box-at(x)</i></p> <p><i>Effect</i> = {<i>holding-box</i>} with 0.9</p> <p style="padding-left: 40px;">{} with 0.1</p>

Figure 1.7: Action `grab(loc)` in PPDDL probabilistic language..

consider the action *grab* as depicted in Figure 1.6. In this domain, after executing the action *grab*, the robot will be holding the box with probability 0.9 and with a probability 0.1 it will fail. Figure 1.7 shows a description of this new action in the probabilistic version PDDL [Ghallab et al., 1998], called PPDDL (*Probabilistic Planning Domain Description Language*).

SSP solution based on Dynamic Programming. The Value Iteration (VI) algorithm [Puterman, 1994a] is a traditional method for solving MDP and SSP problems. This approach computes an optimal policy (state to action mapping) that minimizes the expected cost of the agent to the goal by applying Dynamic Programming (DP) techniques [Puterman, 1994a]. This is done by computing successive approximations (until convergence) of a state value function $V(s)$ that estimates the expected accumulated cost in state s . At each iteration, all states are simultaneously updated which implies in $O(|S| \times |A| \times |S|)$. Moreover, consider states described by n Boolean variables implying $|S| = 2^n$, that is, the number of states in an SSP grows exponentially with the number of state variables. This is called "*Bellman's curse of dimensionality*" and shows the real complexity of solving a probabilistic planning problem. Thus, since Value Iteration (VI) is a synchronous dynamic programming, which approximates the value of all states at each iteration, it is not a scalable strategy [Bonet and Geffner, 2003b].

A more efficient solution also based on DP performs a special type of heuristic search. It uses heuristics to decide on which state to approximate (update) first and applies asynchronous dynamic programming algorithms, i.e. not all states are updated at each iteration. This idea is used by the algorithm called RTDP (*Real Time Dynamic Programming*) [Bonet and Geffner, 2003a], which only updates a subset of the state space at each iteration, giving more priority to states that are most likely to be reached and with a better cost estimation to the goal. Thus, these techniques are able to solve larger SSP problems, but still their efficiency depends on the use of good heuristics. As we will see in this work, these heuristics usually are the same heuristics applied in classical planning, making first a determinization (problem relaxation) of the stochastic actions, and then applying efficient deterministic planning algorithms to compute the cost estimation to the goal.

SSP solution based on Linear Programming. Another way to solve an SSP is to model it as a *linear program in the primal space*, where the variables to be optimized represent the expected accumulated cost for reaching a goal state. Recently, the work of Trevizan et.al. (2017) brought back the SSP formulation as a *linear problem in the dual space* [Altman, 1999, Trevizan et al., 2017a], which can solve SSP problems by optimizing a set of occupation measure variables representing the expected number of times an action is executed in a state. This formulation inherits the same limitations in terms of performance from DP and the linear program formulation in the primal space, however it has been successfully used to develop

efficient heuristics [Trevizan et al., 2017b, Trevizan et al., 2017a]. This is because it is the first approach to consider the probabilities in its problem relaxation, in contrast to the determinization relaxation approach, vastly used in previous works that only consider the costs.

1.2.1 Classes of SSP problems in the presence of dead-ends

Traditional heuristic search solutions for SSP, like RTDP and ILAO [Barto et al., 1995, Bonet and Geffner, 2003b, Hansen and Zilberstein, 2001] make the assumption that it is possible to reach a goal state from any state $s \in S$. An SSP under this assumption does not contain dead-ends, i.e. states from which the agent is unable to reach the goal. E.g., in the Box-loader domain, a dead-end state could be a state where the box is broken and the robot is no longer able to grab it, so the goal can not be reached. In the presence of dead-ends these heuristic search algorithms (and the classical dynamic programming algorithm Value Iteration as well) lose their convergence guarantee and may fail or return non-optimal policies. To overcome this limitation, it is necessary to define an SSP with new assumptions, as well as to develop algorithms and heuristics to solve problems with dead-ends. Recent works [Kolobov et al., 2012, Teichteil-Königsbuch, 2012, Freire and Delgado, 2016, Trevizan et al., 2017b, Trevizan et al., 2017a] have proposed extensions of SSP and developed heuristics that can deal with these situations. Following, we list the extensions that are discussed in this dissertation:

SSP with avoidable dead-ends (SSPADE): An SSP problem has avoidable dead-end states if, starting from the initial state s_0 , there is a policy that takes the agent to a goal state with probability 1, that is, there is a policy with no risk of reaching a dead-end state.

SSP with unavoidable dead-ends (SSPUDE): An SSP problem has unavoidable dead-ends when any policy that starts at the initial state have some chance of reaching a dead end, that is, every policy has a probability less than 1 to reach a goal state.

SSP with maximum goal probability (MAXPROB): When dealing with an SSP problem in the presence of unavoidable dead-ends, the agent task may be to reach the goal with maximum probability, whatever it takes. This problem is a subclass of the SSPUDE problem, where the goal is to find a policy that maximizes the probability of reaching a goal state, ignoring the action costs.

1.2.2 Risk sensitive SSP

One can say that avoiding dead-end states is to avoid the "risk" of the agent to fail and never be able to reach its goal. However in this work we will consider another type of risk, not directly related to dead-end states.

There are different approaches to quantify risk in MDPs and SSPs, among them: (i) the use of an expected exponential utility using a risk factor [Marcus, 1997, Howard and Matheson, 1972, Jaquette, 1976, Denardo and Rothblum, 1979, Rothblum, 1984, Patek, 2001]; (ii) the use of a piece-wise linear transformation function with a discount factor [Mihatsch and Neuneier, 2002]; (iii) a weighted sum between expectation and variance [Sobel, 1982, Filar et al., 1989]; and (iv) the estimation of performance in a confidence interval [Filar et al., 1995, Yu et al., 1998, Hou et al., 2014, Hou et al., 2016]. Note that, the traditional utility of minimizing the expected accumulated cost corresponds to a *neutral attitude* regarding risk. However, due to the complexity of the above mentioned risk approaches, finding optimal policies for them are computationally more costly than solving risk-neutral SSPs [García and Fernández, 2015].

In this work we will use the exponential utility approach to model risk for SSP problems with no dead-ends. Thus, an RS-SSP (risk sensitive SSP) problem incorporates a risk attitude by considering the

expected exponential of accumulated cost, weighted with a risk factor. Within this approach, its possible to evaluate policies with respect to the variance of costs, assuming agents with risk-prone or risk-averse attitudes.

1.3 Objective on this work

The objective of this work is to develop heuristics that can be used to efficiently solve probabilistic planning problems modeled as SSPs. We will consider three types of problems: (I) SSPUDE, (II) MAXPROB and (III) RS-SSP with no dead-ends. These heuristics are evaluated empirically over different benchmark planning domains and are compared to existing heuristics in the area.

1.4 Contributions

The main contribution of this MsC Thesis is to improve the efficiency of solutions based on heuristic search for special classes of SSP problems. For that, we propose three different heuristics, all based on the work of [Trevizan et al., 2017a]. Thus, the main contributions of this work are:

- Development of a new heuristic for SSPs with unavoidable dead-ends (SSPUDES), called h_{pom}^{pe} , that is the first heuristic to consider the state transition probabilities and a penalty for reaching dead-ends. The empirical results show that h_{pom}^{pe} can be used to solve large instances in the analysed domains and is more efficient when compared with the state-of-the-art heuristic.
- Development of the first heuristic to solve MAXPROB problems, also for SSPs with unavoidable dead-ends, called h_{ppom} . The empirical results show that h_{ppom} is able to efficiently solve large instances of MAXPROB problems.
- Formally specify an RS-SSP as a linear program in the dual space. This dual linear program can be interpreted as a probabilistic flow network problem, where we want to optimize a set of occupation measure variables. Unlike the RS-SSP formulation in primal space where the objective is to maximize the exponential utility function for each state, the objective of the dual linear program is to minimize the sum of the expected occupation measure variables weighted by the natural exponential function $e^{\lambda C(s,a)}$, where $C(s,a)$ is the cost of applying action a in state s and λ is the risk factor.
- Development of a heuristic for RS-SSP called h_{pom}^{rs} based on the formulation described in the above item. To the best of our knowledge, this is the first heuristic to risk-sensitive SSPs that takes into account the probabilities and the risk factor. The empirical results show that a heuristic search algorithm with the proposed heuristic h_{pom}^{rs} outperforms the solution of Freitas (2019), considered the state-of-the-art for RS-SSPs with exponential utility function.

The following paper was published, as a partial result of this work (2nd best paper award of BRACIS 2018):

- Milton Condori Fernandez, Leliane Nunes de Barros and Karina Valdivia Delgado. *Occupation Measure Heuristics to solve Stochastic Shortest Path with dead-ends*. In Proceedings of the 7th Brazilian Conference on Intelligent Systems (BRACIS 2018), São Paulo, Brazil.

1.5 Organization

This work is organized in two main parts. **Part I, Heuristics for Probabilistic Planning Problems with dead-ends**, is composed by three chapters, as follows:

Chapter 2 [Background on Probabilistic Planning]: This chapter presents the definitions for MDP, SSP and its three sub-classes (SSPADE, SSPUDE and MAXPROB). We also present solutions for these problems which are well-known heuristic search algorithms from the literature.

Chapter 3 [Background on Heuristics for Probabilistic Planning]: In this chapter we present the background on heuristics for probabilistic planning. We first present the two known action languages used for heuristic extraction, PPDDL and SAS⁺. Then we introduce six classical heuristics (also used for classical planning) plus the heuristic h_{pom} , which is the main inspiration for this work.

Chapter 4 [Two new Heuristics: h_{pom}^{pe} and h_{ppom}]: This chapter presents the contribution of Part I, which are the two heuristics h_{pom}^{pe} and h_{ppom} , used to solve SSPUDE and MAXPROB problems, respectively. Also we show the empirical evaluation of these proposed heuristics over three different planning domains.

Part II, Risk Sensitive Planning as a Linear Program in the Dual Space, is also composed by three chapters, as follows:

Chapter 5 [Background on Risk Sensitive SSPs with Exponential Utility Function]: In this chapter we briefly present the fundamentals on risk sensitive SSPs and define the exponential utility based optimization criterion. We also discuss previous solutions based on dynamic programming, the concept of λ -feasible risk factor and the difficulties of finding its maximum value, which corresponds to the risk extreme-averse attitude.

Chapter 6 [Risk Sensitive SSP as a Linear Program in the Dual Space]: This chapter defines how an RS-SSP can be formulated as a primal and a dual linear program. We also show that the dual linear program can be interpreted as a flow network problem. This is the first contribution of Part II.

Chapter 7 [A Heuristic for RS-SSP: h_{pom}^{rs}]: In this chapter we introduce a first dedicated heuristic for RS-SSPs and evaluate it over three domains with no dead-ends, which is the second contribution of Part II.

Finally, in Chapter 8, **Overall Conclusions and Future Works**, we present the conclusions of this dissertation and a discussion about future works.

Part I

Heuristics for Stochastic Shortest Path with Dead Ends

Chapter 2

Background on Probabilistic Planning

Probabilistic planning is a sub-area of artificial intelligence that deals with decision making in stochastic environments. It defines how an agent should act in an environment in which his actions have stochastic effects. A probabilistic planning problem can be modeled as a Markovian Decision Process (MDP) [Puterman, 1994a]. An MDP describes the interaction between an agent and its environment, Figure 2.1 illustrates how this interaction occurs. At stage t the agent is in state s_t and decides to apply action a_t ; then it goes to the next state s_{t+1} , receives the cost c_{t+1} and the process repeats until a given number of stages.

Figure 2.1: Interaction between an agent and his environment [Barto et al., 1995].

An MDP can be seen as an optimization problem in which the objective is to minimize a utility function, e.g. the expected cumulative cost of the interactions between the agent and the environment during a given number of stages (*horizon*). One can consider three types of horizons:

Finite: The agent acts in the environment for a predefined finite number of stages;

Infinite: The agent never stops acting in the environment; and

Indefinite: The agent acts in the environment for an indefinite number of stages, for example, until it reaches a goal state or a dead-end state.

2.1 Markov Decision Process

Definition 2.1.1. (Markov Decision Process - MDP.) An MDP is defined as a tuple $M = \langle S, A, P, C \rangle$ [Puterman, 1994a], where:

- S is a set of states, also called the state space.
- A is a set of actions; $App(s) \subseteq A$ is the set of actions applicable in a state $s \in S$.

- $P : S \times A \times S \rightarrow [0, 1]$ is the probability transition function that specifies the probability $P(s'|s, a)$ of taking action $a \in A$ at state $s \in S$ and ending in the state $s' \in S$. A transition function must satisfy the constraint $\sum_{s' \in S} P(s'|s, a) = 1, \forall s \in S$.
- $C : S \times A \rightarrow \mathbb{R}$ is the cost function that gives the agent an immediate cost value $C(s, a)$ for taking an action $a \in A$ in the state $s \in S$. $\square$

A solution of an MDP is a *policy* π that describes the behavior of an agent, indicating which action to execute at each state and stage. The optimal policy is the one that minimizes the expected cumulative cost of the agent over a given horizon.

2.2 Finite Horizon MDP

In a finite horizon MDP the horizon of decision stages is finite and the agent behavior depends on both, the state s and stage t . In this case the solution is a *non-stationary policy*.

Definition 2.2.1. (Non-stationary Policy) A *non-stationary policy* is a function $\pi : S \times \mathbb{N}^* \rightarrow A$ that defines an action $\pi(s, t) \in A$ when the agent is at state s and stage t . $\square$

An *execution history* h of an MDP following a policy π is the sequence of states visited by the agent. Let $\mathcal{H}^k$ be the set of all possible execution histories of a finite MDP until the horizon k . So, given an execution history $h = (s_0, s_1, \dots, s_k) \in \mathcal{H}^k$, the probability that h happen, following the policy π , is:

$$P(h|\pi) = \prod_{t=0}^{k-1} P(s_{t+1}|s_t, \pi(s_t, t)), \quad (2.1)$$

where s_t and s_{t+1} are the states of h visited at time t and $t + 1$, respectively; and $\pi(s_t, t)$ is the action executed at state s and stage t , following policy π .

Expected Cumulative Cost of a Policy

The accumulated cost of a history $h = (s_0, s_1, \dots, s_k)$ induced by the policy π is:

$$V^\pi(h) = \sum_{i=0}^{k-1} C(s_i, \pi(s_i, i)), s_i \in h, \quad (2.2)$$

and the expected accumulated cost of any history in $\mathcal{H}^k$ following π is:

$$\sum_{h \in \mathcal{H}^k} V^\pi(h) P(h|\pi). \quad (2.3)$$

Let $\mathcal{H}_s^k \subseteq \mathcal{H}^k$ be the set of histories that start at s , i.e. $s_0 = s$. We can now define the expected accumulated cost of a state s following a policy π in k stages as:

$$V^\pi(s) = \sum_{h \in \mathcal{H}_s^k} V^\pi(h) P(h|\pi), \quad (2.4)$$

or yet, if we consider a state s being any state of a history h of horizon k and t is the number of stages-to-go to reach the end of the horizon, we can also define the expected accumulated cost of any state s and t states-to-go, following a policy π , as follows [Puterman, 1994a]:

$$V^\pi(s, t) = C(s, \pi(s, t)) + \sum_{s' \in S} P(s'|s, \pi(s, t)) V^\pi(s', t-1). \quad (2.5)$$

Note that for a finite horizon MDP, $V^\pi(s, t)$ is a finite value. But in an infinite horizon MDP $t = \infty$.

2.3 Infinite Horizon MDP

In an infinite horizon MDP the number of decision stages is infinite and the agent behavior depends only on the state s , and not on the number of stages-to-go. Thus, the solution of an infinite horizon MDP is a *stationary policy*.

Definition 2.3.1. (Stationary Policy) A *stationary policy* is a function $\pi : S \rightarrow A$ that defines an action $\pi(s) \in A$ when the agent is at state s . $\square$

An important property of an infinite horizon MDP is the value of state s following a policy π does not depend of the stage t . This makes Equation 2.5 result in:

$$V^\pi(s) = C(s, \pi(s)) + \gamma \sum_{s' \in S} P(s' | \pi(s), s) V^\pi(s'), \quad (2.6)$$

where $0 < \gamma < 1$ is the discount factor that weighs the accumulated future cost for convergence.

The optimal value of a state s in an infinite horizon, denoted by $V^*(s)$, is the one that minimizes the discounted expected cost and is called Bellman Equation [Bellman, 1957]:

$$V^*(s) = \min_{a \in A} \left\{ C(s, a) + \gamma \sum_{s' \in S} P(s' | a, s) V^*(s') \right\}. \quad (2.7)$$

Equation 2.7 means that $V^*(s)$ is the function that minimizes the cost of applying the optimal action in s , plus the expected discounted optimum values of the possible future states.

In an infinite horizon MDP, we can also compute the expected discounted accumulated cost of a policy, analogous to Expression 2.3, and considering only the set of histories $\mathcal{H}_s^\infty$ that start in state s , we can compute the expected cost of a policy π from s as:

$$J_s(\pi) = \sum_{h \in \mathcal{H}_s^\infty} V^\pi(h) P(h | \pi). \quad (2.8)$$

A solution for an infinite horizon MDP can be found by the dynamic programming algorithm Value Iteration (VI), that uses the Bellman equation as an update function [Bellman, 1957]:

$$V_{t+1}(s) = \min_{a \in A} \left\{ C(s, a) + \gamma \sum_{s' \in S} P(s' | a, s) V_t(s') \right\}, \forall s \in S. \quad (2.9)$$

The algorithm VI initializes $V_0(s)$ with any value and iteratively updates the value function of each state until convergence. It is common to consider that V converges if the residual error of all states is less than an ϵ , where the residual error of a state s , $Res(s)$, at iteration t is defined as:

$$Res(s) = |V_t(s) - V_{t-1}(s)|.$$

So the algorithm VI ends when $\forall s \in S, Res(s) < \epsilon$.

Definition 2.3.2. (Greedy policy) Given a value function $V : S \rightarrow \mathbb{R}$, a greedy policy π^V assigns to each state $s \in S$ a greedy action that minimizes $V(s)$:

$$\pi^V(s) = \underset{a \in A}{\operatorname{argmin}} \left\{ C(s, a) + \gamma \sum_{s' \in S} P(s' | a, s) V(s') \right\}$$

Thus, with the optimal value function V^* (Equation 2.7), obtained after the convergence of the algorithm VI, it is possible to define an optimal policy, denoted by π^* , as the greedy policy associated with $V^*(s)$, that is, $\pi^*(s) = \pi^{V^*}(s)$.

Definition 2.3.3. (Optimal Policy for an infinite horizon MDP) An optimal policy, denoted by π^* , is the policy that minimizes the expected discounted accumulated cost from all states, that is, $\forall \pi : J_s(\pi^*) \leq J_s(\pi), \forall s \in S$. $\square$

An MDP can also be represented by a (stochastic) hypergraph in which the nodes represent the states $s \in S$ and the edges represent the actions $a \in A$ guided by the probability transition function P with an associated cost C . Figure 2.2 shows an example of an MDP represented as a hypergraph. E.g. the action a_3 reaches state s_1 with probability 0.2 and reaches state s_g with probability 0.8, which corresponds to a hyperedge a_3 . In this dissertation, we will use this representation to illustrate simple examples of MDPs. For simplicity, we call it *graph*, instead of *hypergraph*.

Figure 2.2: Example of an MDP represented as a directed graph, where nodes are labeled as states, edges represent actions that can be applied in a state s guided by the probability transition function.

2.4 Stochastic Shortest Path - SSP

In general, planning problems involve an initial state s_0 and a set of goal states G . In this case, the objective is to find a policy that optimizes the undiscounted cumulative cost from the initial state to a goal state. Such problems have an indefinite horizon, since the agent stops acting when reaching a goal state. In this work, we specify such problem through a *Stochastic Shortest Path* (SSP) model. An SSP is an MDP with an initial state s_0 (or a set of initial states), a set of goal states G , and with an specific optimization criterion.

We say that a policy π is *s-proper* if it has probability 1 to take the agent to a goal state, starting from the state $s \in S$, that is, $P_\pi^G(s) = 1$, where $P_\pi^G(s)$ is the probability to reach a state $g \in G$ following policy π and starting in state s .

Definition 2.4.1. (Proper Policy) A policy π is *proper* if for every state $s \in S$, π is *s-proper*. In other words, π is proper if it has probability 1 to take the agent to a goal state from any state $s \in S$. Formally, π is *proper* if $\forall s \in S : P_\pi^G(s) = 1$. A policy π is said to be *improper* if it is not *proper*, that is, when there is at least one state with probability less than 1 to take the agent to the goal, i.e. π is *improper* if $\exists s \in S : P_\pi^G(s) < 1$. $\square$

Figure 2.3 shows an example of an SSP with a proper policy $\pi_1 = \{s_0 : a_0; s_1 : a_1\}$ and an improper policy $\pi_2 = \{s_0 : a_1; s_1 : a_1\}$. Note that following policy π_1 the agent reaches the goal with probability 1. However with policy π_2 the agent may not reach the goal from state s_0 . Proper policies eventually reach a goal state and therefore have a finite expected cost. Considering a positive cost function, improper policies may have an infinite expected cost.

Definition 2.4.2. (Stochastic Shortest Path - SSP) An SSP is a tuple $M = \langle S, A, P, C, s_0, G \rangle$ in which S is a finite set of states; A is a finite set of actions, $App(s) \subseteq A$ is a set of actions applicable in

Figure 2.3: Example of an SSP with a proper and an improper policy.

state s ; $P(s'|s, a)$ is the probability of taking action $a \in \text{App}(s)$ in state s ending in state s' ; $C(s, a)$ is the cost function that gives the immediate cost value for taking an action $a \in \text{App}(s)$ in state s ; $s_0 \in S$ is the initial state and G is the set of goal states [Bertsekas and Tsitsiklis, 1991]. Additionally, it is common to make two assumptions:

- $\forall s \in S$, there exists at least one proper policy and;
- all improper policies have an infinite expected cost (Equation 2.8).

Under the above assumptions, a solution for an SSP is a proper, deterministic and stationary *policy* $\pi : S \rightarrow A$; π is also optimal if it minimizes the expected accumulated cost. $\square$

Note that, an SSP that satisfies Definition 2.4.2 **does not have dead ends**, i.e. states from which it is not possible to reach a goal state [Kolobov et al., 2012]. Besides, since the agent stops acting upon finding a goal state, the horizon is indefinite and therefore, the cost does not need to be discounted for the value function to converge (Equation 2.9). Thus, in an SSP, the objective is to find a policy that minimizes the expected accumulated cost (without discount) of taking the agent from the initial state to a goal state.

In order to formalize the optimization criterion for SSPs, we first define the probability of reaching a goal state from any state s and the expected cost of a history that reaches a goal state from s .

Let $S' \subseteq S$ be any set of states and $\mathcal{H}_s^{S'}$ be the set of histories that start in state $s \in S$ and reach a state $s' \in S'$. We define the probability of the agent to reach a state $s' \in S'$ from a state s then following a policy π , called $P_\pi^{S'}(s)$, as the sum of the probabilities of the histories in $\mathcal{H}_s^{S'}$ occur [Mausam and Kolobov, 2012], following π , i.e.:

$$P_\pi^{S'}(s) = \sum_{h \in \mathcal{H}_s^{S'}} P(h, \pi), \quad (2.10)$$

where $P(h, \pi)$ is the probability of the history h occurs following the policy π . We can also define the expected cost of a history that reaches a state in $S' \subseteq S$ from s following a policy π , denoted by $C_\pi^{S'}(s)$, as the accumulated cost of a history $h \in \mathcal{H}_s^{S'}$ (similar to Equation 2.3) weighted by their probabilities, as follows:

$$C_\pi^{S'}(s) = \sum_{h \in \mathcal{H}_s^{S'}} V^\pi(h) P(h, \pi). \quad (2.11)$$

Greedy Graph. Given an SSP and a policy π whose value is V (Equation 2.6), called π^V , the greedy graph $\mathcal{G}_V$ is a graph whose the nodes are the states reachable by following the greedy policy π^V starting at the initial state s_0 . Thus, s_0 is the root node of $\mathcal{G}_V$, and the successors of each state $s \in \mathcal{G}_V$ are the nodes that can be obtained by executing the action $\pi^V(s)$. Figure 2.4 shows an example of a greedy graph of an SSP (gray subgraph) for the greedy policy $\pi^V = \{s_0 : a_0; s_1 : a_0; s_2 : a_0; s_4 : a_0; s_g : a_2\}$ applied from the initial state. Note that s_3 , s_5 and s_6 are not part of $\mathcal{G}_V$, since π^V never visits those states.

Figure 2.4: Example of a greedy graph $\mathcal{G}_V$ (grey nodes and dark edges) in an SSP induced by the greedy policy $\pi^V = \{s_0 : a_0; s_1 : a_0; s_2 : a_0; s_4 : a_0; s_g : a_2\}$, starting from s_0 .

Optimal Policy. An optimal policy of an SSP is any proper policy that minimizes the expected cost of a history that reaches a goal state (Equation 2.11), as follows:

$$\pi^* = \underset{\pi \text{ is proper}}{\operatorname{argmin}} C_\pi^G(s), \forall s \in S. \quad (2.12)$$

Like in the solution of an infinite horizon MDP, the optimal policy π^* of an SSP is the greedy policy associated with the function $V^*(s) = \min_{\pi} V^\pi(s)$, with $\gamma = 1$, which satisfies the optimality equation of Bellman [Bertsekas and Tsitsiklis, 1991]:

$$V^*(s) = \begin{cases} 0 & \text{if } s \in G, \\ \min_{a \in \operatorname{App}(s)} \{C(s, a) + \sum_{s' \in S} P(s'|s, a) V^*(s')\}, & \text{otherwise.} \end{cases} \quad (2.13)$$

2.5 Solutions for SSPs

In this section we review two main approaches to find the optimal value function V^* . The first is based on asynchronous DP, which iteratively solves the problem updating the entire state space in each iteration; and the second is based on asynchronous DP that only updates a subset of the state space in each iteration.

2.5.1 Synchronous Algorithms for Probabilistic Planning

One way to compute the solution of an SSP is to compute the value function V^* using dynamic programming. The value iteration (VI) algorithm can solve an SSP using Equation 2.13 as an assignment function for the value function:

$$V_{t+1}(s) = \min_{a \in \operatorname{App}(s)} \{C(s, a) + \sum_{s' \in S} P(s'|s, a) V_t(s')\} \quad (2.14)$$

being $V_0(s)$ any value $\forall s \notin G$ and $V_0(s) = 0, \forall s \in G$.

The VI algorithm updates the value function of all states at each iteration and therefore is called a *synchronous dynamic programming algorithm*. After convergence, this algorithm creates a total policy, that is, a policy defined for all states of S .

Algorithm 1 Value Iteration Algorithm.

Input: An SSP M and the threshold error value ϵ .

Output: $\langle V, P \rangle$, the value and policy function.

```
1: initialize  $V$ 
2: while true do
3:    $Bellman\_error \leftarrow 0$ 
4:   for each state  $s \in S$  do
5:      $oldV \leftarrow V(s)$ 
6:      $V(s) \leftarrow \min_{a \in App(s)} [C(s, a) + \sum_{s' \in S} P(s'|a, s)V(s')]$ 
7:      $pi(s) \leftarrow \operatorname{argmin}_{a \in App(s)} [C(s, a) + \sum_{s' \in S} P(s'|a, s)V(s')]$ 
8:      $Bellman\_residual(s) \leftarrow |V(s) - oldV(s)|$ 
9:      $Bellman\_error \leftarrow \max(Bellman\_error, Bellman\_residual(s))$ 
10:  end for
11:  if  $Bellman\_error < \epsilon$  then
12:    return  $\langle V, pi \rangle$ 
13:  end if
14: end while
```

Algorithm 1 initializes the value function arbitrarily (line 1). Then, the values are updated iteratively (line 6) using the Bellman backup operator (Equation 2.13) in order to get better approximations to the optimal value. The Bellman residual error is computed in line 8. The algorithm VI converges to the optimal value function in polynomial time of $|S|$.

However, in an SSP where the initial state s_0 is known, we only need to define the policy for the states reachable from s_0 . This is an important feature, since the size of the state space grows exponentially with the number of atomic propositions that describe the state, which makes it unfeasible to keep all state space in memory. In the following section, we describe asynchronous algorithms that update only a subset of states at each iteration.

2.5.2 Asynchronous Algorithms for Probabilistic Planning

In the previous section we described a synchronous approach for solving SSP, the algorithm VI, where the update step is performed for all states at each iteration. In this section we describe a more efficient approach for solving an SSP, which only updates a subset of state space at each iteration.

The *Find and Revise* (F&R) framework defines a general way of solving an SSP [Bonet and Geffner, 2003a]. Note that, **this framework assumes that the traditional SSP does not contain dead-end states**. The F&R proposes that only one state have to be updated at each iteration and uses an admissible heuristic to initialize the value function, reducing the convergence time.

Algorithm 2 Find and Revise [Bonet and Geffner, 2003a].

Input: SSP M

Output: V^* : an optimal value function for M

```
1: initialize  $V$  with an admissible heuristic
2: repeat
3:   FIND a state  $s$  in the greedy graph  $\mathcal{G}_V$  with  $Res(s) > \epsilon$ 
4:   REVISE  $V$  at  $s$ 
5: until no such state is found
6: return  $V$ 
```

Given an SSP M , the F&R framework uses the greedy graph $\mathcal{G}_V$, rooted in s_0 , to find states that have not yet converged and need to be updated, thus computing the value function only for the states reachable from s_0 that has not converged. The efficiency of an algorithm that implements the F&R scheme depends on the order in which the states are updated and, mainly, on the heuristic that is used to initialize the value function. The closer is the heuristic value of a state s , i.e. $h(s) = V_0(s)$, to the optimal value, $V^*(s)$, the smaller is the number of updates required for this state to converge to the optimal value. In Chapter 3 we will see different types of heuristics used to initialize the value function $V(s)$.

2.5.3 The algorithms RTDP and LRTDP

The *Real-Time Dynamic Programming* (RTDP) algorithm [Barto et al., 1995], is a probabilistic planning algorithm that implements the F&R framework. It starts from the initial state and selects the states to be updated through sampling following the greedy policy, which is performed by *trials*. I.e., a trial simulates the execution of the greedy policy starting from the initial state.

Given a state s , the agent selects a greedy action a and computes the successor state s' by sampling according to the $P(s'|s, a)$. This operation is repeated until a goal state is found. The RTDP algorithm updates, at each iteration, the value function of each state of a trial using the Bellman equation without discount (Equation 2.10), similar to VI. The labeled RTDP algorithm (LRTDP) [Bonet and Geffner, 2003b] is an extension of RTDP that labels the states that have already converged as *solved* and stops updating them, thus accelerating the convergence of the algorithm.

Algorithm 3 is the LRTDP pseudo-code. The main LRTDP function (lines 1-5) receives performs trials until s_0 has been labeled as solved, which means that the value function of all states of the greedy graph has converged. The LRTDP-TRIAL procedure of (lines 6-23 of Algorithm 3) receives a state and selects a greedy action to execute (Line 13); updates the value function of the visited state (Line 14); and then selects the next state to be visited simulating an interaction with the environment through the next-state method (Line 15). A trial continues visiting states until a goal state, or a state labeled as solved, is visited. If a state s is visited for the first time, LRTDP calls the INITIALIZE method, which uses the heuristic h to initialize the value function of this state. Note that each visited state is stored in the visited stack (Line 9). At the end of each trial (Lines 17-22) a call to the CHECK-SOLVED method (Line 19) is called for each state stored in visited (in reverse order to which they were stacked) to check if that state has converged and therefore can be labeled as solved. The CHECK-SOLVED method verifies that all states of the greedy graph rooted in the s_i state have already converged and, in this case, label itself as solved. If the state is not labeled, the algorithm does not analyze the other states in the trial (Line 20). The idea is that the state itself is part of the greedy graph of states visited previously in the trial (the predecessors of itself) and if it did not converge, its predecessors did not converge either. As states are labeled, the size of the trials decreases. As a result, the non-converged states will be visited more frequently, this explains the efficiency of the LRTDP algorithm when compared to RTDP. The way in which the trials are carried out, weighting the selection of the successor state by the probability of transition of the greedy action, causes the LRTDP to visit more often states with greater probability of being reached, leading these states to converge more quickly. This reinforces the feature called *anytime* of this algorithm: at any moment it is possible to extract a useful policy.

Algorithm 3 LRTDP Algorithm [Bonet and Geffner, 2003b].

Input: SSP $\mathcal{M}$, heuristic function h

Output: V^* : an optimal value function of $\mathcal{M}$

```

1: function LRTDP
2:   while  $s_0$ .SOLVED is false do
3:     LRTDP-TRIAL( $s_0$ )
4:   end while
5: end function
6: procedure LRTDP-TRIAL( $s$ )
7:    $visited \leftarrow$  EMPTY-STACK
8:   while  $s \notin G$  and  $s$ .SOLVED is false do
9:      $visited.PUSH(s)$ 
10:    if  $s$  is not defined in  $V$  then
11:      INITIALIZE( $s, h$ )
12:    end if
13:     $a \leftarrow$  GREEDY-ACTION( $s$ )
14:     $V(s) \leftarrow$  min Q-VALUE( $s, a$ )
15:     $s \leftarrow$  NEXT-STATE( $s, a$ )
16:  end while
17:  while  $visited$  is not EMPTY-STACK do
18:     $s \leftarrow visited.POP$ 
19:    if CHECK-SOLVED( $s$ ) is false then
20:      break
21:    end if
22:  end while
23: end procedure
24: procedure INITIALIZE( $s, h$ )
25:    $V^0(s) \leftarrow h(s)$ 
26: end procedure
27: procedure GREEDY-ACTION( $s$ )
28:   return  $\underset{a \in App(s)}{\operatorname{argmin}} \text{ Q-VALUE}(s, a)$ 
29: end procedure
30: procedure Q-VALUE( $s, a$ )
31:   return  $C(s, a) + \sum_{s' \in S} P(s'|s, \pi(s))V(s')$ 
32: end procedure

```

Note that in GREEDY-ACTION procedure (line 27), the trial choose the action that have the minimum expected cost. This means that for a given state s , the GREEDY-ACTION procedure evaluates all possibles actions and choose one that minimizes the accumulated expected cost (line 31) (Equation 5.3).

2.6 Stochastic Shortest Path with dead-ends

SSP problems have been extensively studied by the planning community and efficient solutions have been proposed. However, most of these solutions are based on the two assumptions we have described in Definition 2.4.2, which are very restrictive and not satisfied by many real problems. For instance, an SSP with dead-ends may cause these algorithms to fail or return a sub-optimal policy. Thus, new models and

algorithms have been proposed that relax these assumptions and deal with more general classes of SSPs [Kolobov et al., 2012, Teichteil-Königsbuch, 2012, Trevizan et al., 2017b]. Figure 2.5 illustrates a situation that is not handled by an SSP as in Definition 2.4.2. In the problem of Figure 2.5 it is impossible to reach the goal state s_g from the states d_1 , d_2 and d_3 , since no action is able to take the agent to the goal state s_g from these states and therefore they are dead-ends.

Figure 2.5: Example of SSP with dead-ends: states d_1, d_2, d_3 .

Definition 2.6.1. (Dead end) Given an SSP $\mathcal{M} = \langle S, A, P, C, s_0, G \rangle$, a state $s \in S \setminus G$ is a *dead end* if and only if, the probability of reaching a goal state $g \in G$ from s is zero, that is, $\forall \pi : P_\pi^G(s) = 0$. $\square$

2.6.1 SSP with Avoidable dead-ends

[Kolobov et al., 2012] classify dead-end states into two types, explicit and implicit. Explicit dead-ends are absorbent states (no action takes the agent out of these states) and they are not part of the set of goal states (for example, state d_1 in Figure 2.5). Implicit dead-ends are not absorbent states, but do not have a path that leads them to the goal (for example, states d_2 and d_3 in Figure 2.5).

We can also classify planning problems with dead-ends according to their avoidability, dividing them into problems with avoidable and unavoidable dead-end states. Figure 2.6 illustrates the difference between these two classes of problems. In the SSP of Figure 2.6a, with a dead-end d_1 , there is a policy $\pi_1\{s_0 : a_0\}$ that takes the agent from the initial state with probability one to the goal and thus this problem has avoidable dead-ends. Figure 2.6b shows an SSP, with dead-end d_1 , for which no policy takes the agent from s_0 to s_g with probability 1 and in this case, the problem has unavoidable dead-ends.

Figure 2.6: Problems with avoidable and unavoidable dead-ends: (a) SSP with an avoidable dead-end and (b) SSP with an unavoidable dead-end.

Definition 2.6.2. (SSP with Avoidable dead-ends - SSPADE) A Stochastic Shortest Path with Avoidable dead-ends (SSPADE) is a tuple $\langle S, A, P, C, G, s_0 \rangle$ where S, A, P, C, G , and s_0 are as in the SSP definition, under the following conditions:

- There exist at least one s_0 -proper policy and
- Every improper policy has an infinite expected cost (Equation 2.8). $\square$

The SSPADE model has only two notable differences from the SSP of Definition 2.4.2: (i) it assumes that the initial state is known, and only requires the existence of a partial proper policy; and (ii) the optimal policy π^* of an SSPADE is an s_0 -proper policy that minimizes the expected cost of a history that reaches a goal state starting from the initial state s_0 following a policy π , that is:

$$\pi^* = \underset{\pi \text{ is proper}}{\operatorname{argmin}} C_\pi^G(s_0). \quad (2.15)$$

For instance, in the Figure 2.6a the policy $\pi_1 = \{s_0 : a_0\}$ is proper ($P_{\pi_1}^G = 1$), since it is able to avoid the dead-end d_1 .

2.6.2 SSP with Unavoidable dead-ends

Another class of SSPs proposed by [Kolobov et al., 2012] deals with an SSP with unavoidable dead-ends. Even considering that any improper policy has an infinite expected cost from the initial state, the criterion of minimizing the expected cost is inadequate in problems with unavoidable dead-ends, since it does not distinguish between different improper policies. For example, given two policies π_1 and π_2 , with $P_{\pi_1}^G(s_0) < P_{\pi_2}^G(s_0) < 1$ and $J_{s_0}(\pi_1) = J_{s_0}(\pi_2) = \infty$, these policies are indistinguishable by the criterion of minimize the expected cost from s_0 . However, the agent still needs to make a decision. Observing that π_2 is more likely to reach a goal state, so we can say that this policy is better than π_1 . Thus, we need an optimization criterion to distinguish between these two situations. [Kolobov et al., 2012] proposed two new criteria based on assigning a penalty $D \in \mathbb{R} \cup \{\infty\}$ to the agent in case he visits a dead-end state. The first deals with problems in which this penalty is finite and the second deals with problems in which it is infinite.

Finite Penalty. In the case of an SSP with unavoidable dead-ends and finite value penalty D , the agent may give up trying to reach a target state and pay that amount. Thus, the expected cost is bounded by D . Given a penalty $D \in \mathbb{R}^+$ assigned to the agent when it can not reach a goal state, the limited expected value of a policy π from a state s , denoted by $J_s^D(\pi)$, is given by:

$$J_s^D(\pi) = \min\{D, J_s(\pi)\}. \quad (2.16)$$

Infinite Penalty. In the case of an SSP with unavoidable dead-ends and infinite penalty ($D = \infty$), we have $J_s^D(\pi) = J_s(\pi)$, and we return to the original problem of not being able to distinguish policies that are not s_0 -proper. There are three possible approaches to solve an SSPUDE: (I) we can disregard the cost of actions and use the probability of reaching a goal state from s following a policy π , $P_{pi}^G(s_0)$ (Equation 2.10), as an objective function to be maximized; (II) we can ignore the cost of histories that do not reach a goal state, in this case we use the expected cost of the histories that reach a goal state from s following a policy π , $C_\pi^G(s)$ (Equation 2.11) as an objective function to be minimized; and (III) we can adopt a lexicographical criterion, first maximizing $P_\pi^G(s_0)$ (Equation 2.10) and then minimizing of $C_\pi^G(s_0)$ (Equation 2.11) restrict to the policies that satisfy the first criterion. The third criterion is the one considered in this work.

Definition 2.6.3. (SSP with unavoidable dead-ends - SSPUDE) an SSPUDE [Kolobov et al., 2012] is a tuple $M = \langle S, A, P, C, s_0, G, D \rangle$ where S, A, P, C, s_0 and G are defined as in an MDP and:

- $D \in \mathbb{R}^+ \cup \{\infty\}$ is the penalty assigned to the agent when visits a dead-end.

If $D < \infty$, then M is an SSPUDE with finite penalty (f SSPUDE). The optimal solution of a f SSPUDE is a policy that minimizes the expected cost limited from the state initial s_0 , that is $\pi^* = \operatorname{argmin}_{\pi} J_{s_0}^D(\pi)$

If $D = \infty$, then M is an SSPUDE with infinite penalty (i SSPUDE). The optimal solution of an i SSPUDE is a policy $\pi^* = \operatorname{argmin}_{\pi \in \Pi_{\maxprob}} C_{\pi}^G(s)$.

□

2.7 Solutions for SSPs with dead-ends

In this section we show how to modify the previous solutions to be able to solve SSP with dead-ends, as well as other approaches to solve them.

2.7.1 Solving an SSP with avoidable dead-ends

The algorithm VI does not solve SSPADE because it does not end (converge) in the presence of dead-ends, since the value function $V(s)$ of the dead-end states increases at each iteration and the residual error will always grow.

The framework F&R is able to solve SSPADE. Note that as the value function is updated the expected cost of the states tends to infinity. However, according to the first assumption of SSPs, there is at least one proper policy whose expected cost from s_0 is finite. For example in Figure 2.6a supposes that a_1 is the greedy action for s_0 , as $V(d_1)$ tends to the infinite, and $V(s_g) = 0$ the greedy action of the state s_0 would become a_0 , thus avoiding the dead-end state d_1 .

Despite implementing F&R, the LRTDP algorithm may fail to solve an SSPADE since if a trial reaches a dead-end state this trial can be infinite. A simple way to work around this problem is to stop a trial when a state is visited for the second time.

2.7.2 Solving an SSP with unavoidable dead-ends

As we have mentioned before, a possible solution for an SSPUDE is to apply a lexicographical criterion: (1) maximizing the probability to reach the goal and then; (2) minimizing the expected cost [Kolobov et al., 2012, Trevizan et al., 2017a]; we call this optimization criterion as (MAXPROB, MINCOST) criterion. Another possible solution is to give a high fixed-cost penalty for reaching a dead-end and find a policy under the MINCOST optimization criterion; we call this optimization criterion as fixed cost penalty (FCP) criterion.

2.7.2.1 A solution with dead-end penalty

Let $V_{\text{FCP}}^*(s)$ denote the optimal expected accumulated cost of state s under the fixed-cost penalty for dead-end states. Thus the Bellman equation for the FCP optimization criterion is given by:

$$V_{\text{FCP}}^*(s) = \begin{cases} 0, & \text{if } s \in G, \\ \min\{D, \min_{a \in \mathcal{A}_{pp}(s)} \{C(s, a) + \sum_{s' \in S} P(s'|s, a) \cdot V_{\text{FCP}}^*(s')\}\}, & \text{otherwise.} \end{cases} \quad (2.17)$$

Note that when using Equation 2.17 as a Bellman update operator, the $V_{\text{FCP}}^t(s)$ will eventually converge to D , if s is a dead-end state. However, the use of an efficient heuristic, capable of detecting dead-ends and estimate well the value of states leading to dead-ends, can make the planning algorithm to converge faster to V_{FCP}^* .

A simple heuristic would be an adaptation of $h_{\max}$ (defined in Chapter 3), which is a determinization-based heuristic for cost-estimation that returns D if no solution is found from state s (indicating that s is a dead-end) and the maximum cost-estimate of a deterministic policy, otherwise.

2.7.2.2 Maximum Goal Probability

Given an SSPUDE, the MAXPROB problem is to find a set of improper policies that reach the goal with maximum probability. For this purpose, the cost function $C(s, a)$ is ignored and the optimization criterion is to maximize a value function $V_p(s)$ denoting the probability of reaching the goal from state s .

Definition 2.7.1. (MAXPROB). A MAXPROB problem is a special case of SSPUDE problem where the costs are ignored and the optimization criterion is to find a policy that maximizes the probability to reach a goal state. The value of a policy for a MAXPROB problem is defined as [Steinmetz et al., 2016]:

$$V_p^\pi(s) = \begin{cases} 1, & \text{if } s \in G \\ 0, & \text{if } s \in S_\perp \\ \sum_{s'} P(s'| \pi(s), s) \cdot V_p^\pi(s'), & \text{otherwise,} \end{cases} \quad (2.18)$$

where $S_\perp$ is the set of *dead-end states* (non-goal absorbing states).

□

Note that when the state space is acyclic, every execution will end in an absorbing state within a finite number of steps. However, for cyclic spaces, the Bellman update operator for MAXPROB may have sub-optimal fixed points. This is because, the SSPUDE may have *traps*, i.e. strongly connected components composed by sets of states from which there is no path to a goal state. In this case, we can not update all state space, converging to a sub-optimal fixed point. A solution for MAXPROB problem is to find an optimal closed policy π s.t $V_p^\pi(s_0) = V_p^*(s_0)$.

Thus, the optimal value function V^* has to obey the following Bellman equation [Steinmetz et al., 2016]:

$$V_p^*(s) = \begin{cases} 0, & \forall s, s \text{ is a dead end,} \\ 1, & \forall s \in G, \\ \max_{a \in A} \sum_{s'} P(s'|s, a) \cdot V_p^*(s') & \text{otherwise,} \end{cases} \quad (2.19)$$

which can have multiple sub-optimal fixed points, some of them are the $V_p^*(s)$ of maximum probability. This is due to cycles that are *traps*. In a MAXPROB problem, a trap is a set of states where the agent can transit freely without changing the probability to reach the goal. The presence of traps implies that dynamic programming solutions based on the Bellman update operator may fail since updates from an optimistic (upper-bound) initialization are not guaranteed to converge to the optimum. One can either use a pessimistic initialization, or Kolobov et al.'s *Find, Revise and Eliminate Traps* (FRET) algorithm that is able to solve MAXPROB problems optimally and efficiently by eliminating traps and with the help of an admissible heuristic. However, heuristics for SSPs may not be effective for MAXPROB since they are based on a relaxation that solves a deterministic version of the problem, i.e. ignoring the probabilities [Steinmetz et al., 2016].

A simple heuristic for this problem can be h_1 , that initially assigns probability 1 for all states. Another simple heuristic for MAXPROB is an adaptation of h_{max} (defined in Chapter 3) that returns probability 0 if no solution is found from state s and probability 1, otherwise.

2.7.2.3 The MAXPROB-MINCOST Criterion

In the lexicographical criterion [Kolobov et al., 2012, Trevizan et al., 2017b] we first find a set of policies that maximize the probability of reaching the goal, $P_\pi^G(s)$ (according with Equation 2.10 which can be computed by Equation 2.19), and then, among them, find the ones that minimize the expected cost, $C_\pi^G(s)$ (according with Equation 2.11 which can be computed by Equation 2.13)).

Figure 2.7: Pareto front policies of an SSPUDE (extracted from [Simão, 2017]). The right arrow indicates the best policy when we consider that the maximum probability of reaching a goal state has a higher preference. The left arrow indicates the best policy when we consider that the minimum expected cost has a higher preference.

This problem could be seen as a multi-objective sequential decision making and the solution is the Pareto front policy set [Roijers et al., 2013]. Figure 2.7 plots the policies of an SSPUDE in terms of their maximum probability of reaching a goal state from the s_0 and the minimum expected cost of their trajectories ending in a goal state. For example, the policy π_1 have a probability to reach the state from the initial state of 0.5 with expected cost of the histories that reach the goal of 1.5. Note that, in this example, π_1 dominates all policies with $C_\pi^G(s_0) = 1.5$, that is, between policies with $C_\pi^G(s_0) = 1.5$, π_1 is the one with the highest probability. Similarly, π_1 dominates all policies $P_\pi^G(s_0) = 0.5$, i.e. between policies with $P_\pi^G(s_0) = 0.5$, π_1 is the one that has the lowest expected cost of the histories to the goal from s_0 . Note that a more reasonable criterion is to maximize the probability of reaching the goal before minimizing the cost of the histories that lead to the goal, since smaller probabilities increase the risk of the agent receiving an infinite penalty. Therefore, maximizing $P_\pi^G(s)$ has preference over minimizing $C_\pi^G(s)$. This allows us to impose an order among the objectives. According with this criterion, in Figure 2.7, the optimal policy is policy π_2 , i.e. the pareto front policy that maximizes the probability of reaching a goal state, that is, $\pi^* = \operatorname{argmax} P_\pi^G(s) \text{ s.t. } \pi \in \text{pareto front}$. In problems that more than one policy is returned, the minimum expected cost criterion is used as a tiebreaker.

Thus, given an SSP with infinite penalty, the expected value of a policy π from a state s , denoted by $J_s^\infty(\pi)$, is given by an ordered pair [Kolobov et al., 2012]:

$$J_s^\infty(\pi) = (P_\pi^G(s), C_\pi^G(s)), \quad (2.20)$$

which defines two criteria for evaluating a policy based on a relation of order " $<$ " between two policies π_1 and π_2 , where $\pi_1(s) < \pi_2(s)$ denotes that π_2 is preferable in relation to π_1 in s : $\pi_1(s) < \pi_2(s)$ when $P_{\pi_2}^G(s) > P_{\pi_1}^G(s)$ or, when $P_{\pi_1}^G(s) = P_{\pi_2}^G(s)$ and $C_{\pi_2}^G(s) < C_{\pi_1}^G(s)$. Thus, the optimal policy π^* is given by:

$$\pi^* = \operatorname{argmax}_{<\pi} J_{s_0}^\infty(\pi) = \operatorname{argmax}_{<\pi} (P_\pi^G(s), C_\pi^G(s)), \quad (2.21)$$

where $\max_{<\pi}$ is the maximization operator according to the order relation $<$. The calculation of π^* can also be defined in two consecutive steps:

$$\begin{aligned} \text{Step 1 : } \Pi_{\maxprob} &= \left\{ \pi' \mid \pi' \operatorname{argmax}_{\pi} P_\pi^G(s) \right\}, \\ \text{Step 2 : } \operatorname{argmin}_{\pi \in \Pi_{\maxprob}} & C_\pi^G(s). \end{aligned} \quad (2.22)$$

2.7.3 Find, Revise and Eliminate Traps - FRET

Although VI can be used to solve MAXPROB problems, it is inefficient when dealing with problems with a large number of states. [Kolobov et al., 2012] introduces the FRET (*Find, Revise and Eliminate Traps*) framework (Algorithm 4) that allows to use a heuristic search approach to solve MAXPROB problems, and SSPs with multiple fixed points. FRET can deal with *traps*, i.e. strongly connected components (SCCs) composed by sets of states from which there is no path to a goal state.

FRET extends the *Find and Revise* framework (Algorithm 2) and solves the convergence problem caused by the existence of multiple sub-optimal fixed points. It is composed by two main steps. The first step is *Find and Revise*, where it tries to find the next largest V_i' that satisfies $V_i' = \mathcal{B}V_i'$ ($\mathcal{B}$ is the Bellman Backup Operator), i.e., that converges, which may have problematic regions (*traps*). The second step is *Eliminate Trap*, which changes V_i' to remove the problematic regions and for this, the algorithm collapse all the states involved in the cycle belonging to a trap in a new meta-state. The result is a new admissible estimation of V_{i+1} , s.t $V_{i+1} < V_i'$. Finally the algorithm iterates these two steps until convergence to V^* , this final V^* will not have more problematic regions which allows FRET to extract an optimal policy from V^* .

Algorithm 4 Find, Revise and Eliminate Traps Algorithm (adapted from [Kolobov et al., 2012]).

Input: MDP $\mathcal{M}$ (could be a MaxProb), admissible value function V_0 .

Output: Optimal policy π^*

```

1:  $\mathcal{G} = \{S_G, A_G\} \leftarrow$  reachability graph extracted from  $\mathcal{M}$ 
2:  $V_i \leftarrow V_0$ 
3:  $V_i' \leftarrow \text{Find\_and\_Revise}(M, V_i)$ 
4:  $V_{i+1} \leftarrow \text{Eliminate\_Traps}(M, V_i')$ 
5: while  $V_{i+1} \neq V_i'$  do
6:    $V_i \leftarrow V_{i+1}$ 
7:    $V_i \leftarrow \text{Find\_and\_Revise}(M, V_i)$ 
8:    $V_{i+1} \leftarrow \text{Eliminate\_Traps}(M, V_i')$ 
9: end while
10:  $V^* \leftarrow V_{i+1}$ 
11:  $\pi^* \leftarrow$  the optimal policy
12:  $Processed \leftarrow$  the set of goal states extracted from  $\mathcal{M}$ 
13:  $\mathcal{G}^{V^*} = \{S_{\mathcal{G}^{V^*}}, A_{\mathcal{G}^{V^*}}\} \leftarrow$  greedy graph of  $V^*$ 
14: while  $Processed \neq S_{\mathcal{G}^{V^*}}$  do
15:   choose  $s \in S_{\mathcal{G}^{V^*}} \setminus Processed$ 
16:   choose  $a \in A$  s.t  $P(s'|a, s) > 0$  for some  $s' \in Processed$ 
17:    $Processed \leftarrow Processed \cup \{s\}$ 
18:    $\pi^*(s) \leftarrow a$ 
19: end while
20: return  $\pi^*$ 

```

Running FRET on a simple example of SSPUDE with zero-cost traps

To illustrate how FRET can eliminate traps, consider the SSP depited in the following figure (extracted from Kolobov' slides presented at ICAPS 2011), which contains zer-cost cycles and deterministic actions (for simplicity):

Notice that the costs are the labels below the deterministic actions. The algorithm FRET starts by initializing the value function with an admissible heuristic (line 2). We show these heuristic values inside each node (state) in the SSP, as follows:

When FRET updates the value function by calling the *Find_and_Revise* algorithm (lines 3 and 7) it identifies zero-cost action cycles (traps), using the Tarjan's algorithm to identify SCCs in the reachability graph starting in s_0 (line 1). Then, FRET collapses these states (lines 4 and 8) in a meta-state and assign to it the value ∞ , if the trap is a dead end, and c_{out} if the trap has some other alternative action that leads to the goal, where c_{out} is the lowest expected cost to the goal starting at this meta-state. In our example, the trap involving states s_1 and s_2 are identified in line 3 and depicted by the dotted red ellipse as follows:

This trap is then eliminated in line 4 and assigned to it the value ∞ , since there is no other action that can be selected from this trap to the goal:

Note that the process of collapsing the states of a trap into a new meta-state makes references to the *eliminate-traps* step of FRET, that temporally works without traps. FRET repeats the *Find_and_Revise* and *Eliminate_Traps* steps (lines 5-9) to identify and eliminate all possible traps. In our example, the trap involving states s_3 and s_4 is identified in line 7 and depicted by the dotted red ellipse as follows:

And since it has an action of expected cost -1 , FRET eliminates this second trap assigning the value -1 to it:

Finally, when the value function converges (line 5), i.e. $V_{i+1} = V'_i$, the states will have the value of the meta-state they belong to and FRET can extract the optimal policy that reaches the goal. We should notice that the greedy policy not always is the optimal policy: when extracting a policy we must be sure that it reaches the goal state, by iteratively connecting states using greedy actions starting from the goal state (lines 14-19). In our example, the green edges characterize the optimal policy:

FRET is proved to converge to the optimal policy if an admissible heuristic is used [Kolobov et al., 2012]. We can also use FRET to solve a MAXPROB problem by considering only the probabilities instead of the costs and using Equation 2.19 to update the value function. In this case FRET collapses traps in a meta-state and assign to it the value 0, if the trap has some other alternative action that leads to the goal and c_{out} is the highest expected probability to the goal.

Chapter 3

Background on Heuristics for Probabilistic Planning

In Chapter 2 we discussed how probabilistic planning problems can be modeled by SSPs and described algorithms that iteratively approximate the optimal value function. An important aspect in this approach is how the value function is initialized. In this chapter we describe different heuristics that can be used to initialize the value function and therefore help to reduce the number of iterations needed for the algorithms to converge to the optimal value function.

A heuristic is a function $h : S \rightarrow \mathbb{R}$ that estimates the lowest cost (or the highest probability) to reach a goal state $s' \in G$ from a state $s \in S$. Heuristics are used by search algorithms to define which states will be visited first. A heuristic h is *admissible* if it does not overestimate the optimal cost of any state, that is $\forall s \in S : h(s) \leq h^*(s)$, where $h^*(s)$ is the real lowest cost (or the real highest probability) to reach the goal from state s .

Heuristics can be obtained in two ways: considering specific information of the problem or automatically extract information from the problems description (in a formal language). Then we classify heuristics following the type of the domain knowledge:

- **Domain Dependent Heuristic:** uses specific information of the problem to estimate the cost of a given state s to a goal state s_g . In general, a domain dependent heuristic is only applicable to the problems of the domain for which it was developed.
- **Domain Independent Heuristic:** uses information automatically extracted from the problem description language and can be applied in different types of problems described in that language.

In this work we are interested only on the study and development of domain independent heuristics.

3.1 Planning problem description languages

The use of a formal language to describe a planning problem, besides representing the state space in a compact way, allows an agent (an algorithm) to automatically reason about the structure of the problem and to extract good estimates about the costs and probabilities of an SSP or SSPUDE. In fact, this is a key research feature of the automated planning area.

3.1.1 Action languages based on predicates: PDDL and PPDDL

An action language can be used to specify a planning problem in terms of the world properties and how they can be changed by the agent's actions. In this work we discuss three types of languages:

propositional, relational and multi-valued (variables). In a propositional language the properties of the world is represented by a set of atomic propositions $\mathbb{P}$ and a deterministic action a is defined as a triple $\langle Precond(a), Add(a), Del(a) \rangle$, where:

- $Precond(a) \subseteq \mathbb{P}$ represents the applicability condition of action a in any state s of the world. It is defined as a set of propositions that must be true in a state in which the action a will be executed;
- $Add(a) \subseteq \mathbb{P}$ represents the properties that become true after executing action a in a state s (called *positive effects* of an action); and
- $Del(a) \subseteq \mathbb{P}$ represents the properties that become false after executing action a in a state s (called *negative effects* of an action).

Definition 3.1.1. (Deterministic Planning Problem). A deterministic planning problem is defined by a tuple $\mathcal{P}^{det} = \langle \mathbb{P}, \mathbb{A}, C, s_0, \mathbb{G} \rangle$ where:

- $\mathbb{P}$ is a set of atomic propositions that describe the properties of the world;
- $\mathbb{A}$ is a set of deterministic actions, each one described by the triple $\langle Precond(a), Add(a), Del(a) \rangle$, with $Precond(a) \subseteq \mathbb{P}$, $Add(a) \subseteq \mathbb{P}$ and $Del(a) \subseteq \mathbb{P}$;
- $C : \mathbb{A} \rightarrow \mathbb{R}$ is a function that defines the cost of each action (if it is not defined we assume cost 1 for all actions);
- $s_0 \subseteq \mathbb{P}$ is a set of true propositions in the initial state; and
- $\mathbb{G} \subseteq \mathbb{P}$ is a set propositions that describe the goal. □

Definition 3.1.2. (State transition model) Let $\mathcal{P}^{det} = \langle \mathbb{P}, \mathbb{A}, C, s_0, \mathbb{G} \rangle$ be a deterministic planning problem. The state transition model $\mathcal{M}$ of $\mathcal{P}^{det}$ is given by the tuple $\langle S, A(s), T, s_0, G \rangle$ where:

- $S = 2^{|\mathbb{P}|}$ is the set of states, $2^{|\mathbb{P}|}$ is the power set of $\mathbb{P}$;
- $App(s) = \{a | a \in \mathbb{A} \text{ and } Precond(s) \subseteq s\}$ is the set of actions applicable in $s \in S$;
- $T : S \times \mathcal{A} \rightarrow S$ is the deterministic transition function, being $T(s, a) = s \setminus Del(a) \cup Add(a)$, $\forall s \in S$, $\forall a \in A(s)$;
- $s_0 \in S$ is the initial state; and
- $G = \{s \in S | \mathbb{G} \subseteq s\}$ is the set of goal states in which all the propositions of the goal $\mathbb{G}$ are true, i.e., $s \in G \Leftrightarrow \mathbb{G} \subseteq s$. □

To represent the probabilistic effect of the actions the action specification language must define a list of probabilistic effects:

$$a = \langle Precond, Effects \rangle, \quad (3.1)$$

where $Effects$ is a list of effects with associated probabilities $[p_1 : (Add_1, Del_1), \dots, p_n : (Add_n, Del_n)]$, such that $\sum_{i=1}^n p_i = 1$. Then, applying the action a in a state s satisfying $Precond$, the effect $(Add_i, Del_i) \in Effects$ occurs with probability p_i .

Definition 3.1.3. (Probabilistic Planning Problem). A probabilistic planning problem can be described by an action language (e.g. PPDDL) and defined by a tuple $\mathcal{P}^{prob} = \langle \mathbb{P}, \mathbb{A}, C, s_0, \mathbb{G} \rangle$ in which $\mathbb{P}, s_0, C$ and $\mathbb{G}$ are defined in the same way as in a deterministic planning problem (Definition 3.1.1), and:

- $\mathbb{A}$ is a set of probabilistic actions , such that $a \in A$ is specified by Equation 3.1. $\square$

Given a probabilistic planning problem $\mathcal{P}^{prob} = \langle \mathbb{P}, \mathbb{A}, C, s_0, \mathbb{G} \rangle$, described by an action language, we can infer the probabilistic state transition model $\mathcal{M}(\mathcal{P}^{prob}) = \langle S, A, P, C, s_0, G \rangle$ in which S, A, C, s_0 and G are defined as in a deterministic problem (*Definition 3.1.1*) and:

- P is a probabilistic transition function as in an MDP and $P(s_i|s, a) = p_i$, such that $s_i = (s \setminus Del_i) \cup Add_i$ for the effect $p_i : (Add_i, Del_i)$ of $a \in \mathbb{A}$.

3.1.2 The SAS⁺ task Representation

Another way in which a probabilistic planning problem can be represented is through a SAS⁺ task [Helmert, 2009].

Definition 3.1.4. (Probabilistic SAS⁺ task.) A probabilistic SAS⁺ task is a tuple $\langle \mathbb{V}, A, s_0, s_*, C \rangle$ where:

- $\mathbb{V}$ is a finite set of *state variables* and each variable $v \in \mathbb{V}$ has a finite domain D_v . A *partial state* is a function $\mathbf{s}$ on a subset $\mathbb{V}_s$ of $\mathbb{V}$, such that $\mathbf{s}[v] \in D_v$ for $v \in \mathbb{V}_s$, and $\mathbf{s}[v] = \perp$, otherwise. A *partial valuation* e in state s is the state $res(s, e)$ s.t. $res(s, e)[v] = e[v]$ if $e[v] \neq \perp$ and $res(s, e)[v]$, otherwise;
- A is a finite set of *probabilistic actions*. Each action $a \in A$ has: (i) a precondition $pre(a)$, represented by a partial valuation over $\mathbb{V}$, (ii) a set of effects $eff(a)$, which is a partial valuation over $\mathbb{V}$; and (iii) a *probabilistic distribution* $Pr_a(\cdot)$ over $e \in eff(a)$ that represents the probability of $res(s, e)$ being the state resulting from applying action a in s ;
- s_0 is the initial state, with a complete valuation of variables in $\mathbb{V}$, i.e., $\mathbb{V}_s = \mathbb{V}$;
- s_* is a partial state representing the *goal* ; and
- $C(a) \in \mathbb{R}_+^*$ is the immediate cost of applying a . $\square$

For more details about Probabilistic SAS⁺ representation see [Helmert, 2009].

3.2 Determinization of a Probabilistic Planning Problem specified in PPDDL

One way to define heuristics for probabilistic planning is to estimate the cost (or probability) to reach a goal state in a relaxed deterministic version of the problem [Bonet and Geffner, 2001]. That is, given a probabilistic planning problem $\mathcal{P}^{prob}$ and its relaxed version $\mathcal{P}^{det}$, an estimate of the expected cost of a state s is given by the cost of taking the agent from a state s to a goal state in $\mathcal{P}^{det}$. And to compute the cost in a relaxed problem, in general, we use classical planning heuristics (described in the next section).

Let $\mathcal{P}^{prob} = \langle \mathbb{P}, \mathbb{A}, s_0, \mathbb{G} \rangle$ be a probabilistic planning problem in PPDDL, and $\mathcal{P}^{det} = \langle \mathbb{P}, \mathbb{A}', s_0, \mathbb{G} \rangle$ its relaxed deterministic version. The set of deterministic actions $\mathbb{A}'$ can be defined in two ways:

Most likely Determinization

In a problem relaxation via the most likely determinization, for each probabilistic action we create one single deterministic action. Thus, given a PPDDL probabilistic action $a_i \in \mathbb{A}$ in which $a_i = \langle Precond, [p_1 : (Add_1, Del_1), \dots, (Add_n, Del_n)] \rangle$, its deterministic version is given by $det(a_i) = \langle Precond, Add_j, Del_j \rangle$ where (Add_j, Del_j) is the most likely effect of the action a_i . Then, $\mathbb{A}' = \{a'_i | a'_i = det(a_i), \forall a_i \in \mathbb{A}\}$.

All outcomes Determinization

In a problem relaxation via a most likely determinization, all the effects of a probabilistic action is considered, i.e. we create a different action for each probabilistic effect. Thus, given a PPDDL probabilistic action $a_i \in \mathbb{A}$, where $a_i = \langle \text{Precond}, [p_1 : (\text{Add}_1, \text{Del}_1), \dots, (\text{Add}_n, \text{Del}_n)] \rangle$, the all outcomes determinization creates a set of deterministic actions, one for each action outcome, i.e. $\text{det}(a_i) = \{a_{ij} = \langle \text{Precond}, \text{Add}_j, \text{Del}_j \rangle | 1 \leq j \leq n\}$. Then, the set of relaxed actions of $\mathbb{A}$ is given by $\mathbb{A}' = \bigcup \text{det}(a_i), \forall a_i \in \mathbb{A}$.

3.3 Classical Heuristics

In the previous section we described how we can create a relaxed version of a probabilistic planning problem (through actions' determinization). Each relaxed version, most likely and all outcomes determinization, explores a specific characteristic of the problem. In this section we will describe how to use these relaxed version to compute efficient heuristics, commonly used for deterministic planning.

3.3.1 The h_{add} and h_{max} heuristics

In classical planning, a common relaxation of a deterministic planning problem is to ignore the negative effects of the actions, denoted by $\mathcal{P}^+ = \langle \mathbb{P}, \mathbb{A}^+, s_0, \mathbb{G} \rangle$. These heuristics estimate the cost of reaching a set of propositions $\Delta \in \mathbb{P}$ starting from a state $s \in S$, as a function of the cost of making each proposition $p \in \Delta$ be true, independently. The set Δ can represent a goal or a precondition of an action.

The cost of making a proposition $p \in \mathbb{P}$ true from state s , denoted by $c_s(p)$, is defined as [Bonet and Geffner, 2001]:

$$c_s(p) = \begin{cases} 0 & \text{if } p \in s, \\ \infty & \text{if } \forall a \in \mathbb{A}^+ : p \notin \text{Add}(a), \\ \min_{a \in \mathbb{A}^+ | p \in \text{Add}(a)} C(a) + c_s(\text{Precond}(a)) & \text{if } \exists a \in \mathbb{A}^+ : p \in \text{Add}(a). \end{cases}$$

That is, for a given action a , $c_s(p)$ is updated with the minimum between: (I) its current value and the sum of the cost of an action a that has p as its positive effect, denoted by $C(a)$ and; (II) the cost of reaching the precondition of a from state s , (i.e. $c_s(\text{Precond}(a))$). Note that this update depends on the computation of the cost of a set of atoms $\Delta = \text{Precond}(a) \subseteq P$.

There are two ways of doing this calculation and each one creates a different heuristic, as following.

The heuristic $h_{add}(s)$: We can compute the cost of reaching a set of atoms $\Delta \subseteq P$ using the sum of the costs to reach each atom $p \in \Delta$, i.e.:

$$c_s(\Delta) = \sum_{p \in \Delta} c_s(p),$$

and in this case, we obtain the heuristic $h_{add}(s)$ as:

$$h_{add}(s) = c_s(\mathbb{G}) = \sum_{p \in \mathbb{G}} c_s(p). \quad (3.2)$$

The heuristic $h_{max}(s)$: We can also compute the cost of reaching a set of atoms $\Delta \subseteq \mathbb{P}$ considering the maximum value of the costs of reaching each atom $p \in \Delta$, i.e.:

$$c_s(\Delta) = \max_{p \in \Delta} c_s(p),$$

and in this case, we obtain the heuristic $h_{max}(s)$ as:

$$h_{max}(s) = c_s(\mathbb{G}) = \max_{p \in \mathbb{G}} c_s(p). \quad (3.3)$$

While h_{max} is proved to be an admissible heuristic, h_{add} is not admissible for all outcomes determination (but its use in some domains can be very informative and return relevant sub-optimal plans).

3.3.2 Heuristic h_{FF}

This heuristic is also computed from a relaxed version of the $\mathcal{P}^+$ problem, which does not consider the set of negative effects of the actions. However, the heuristic h_{FF} does not assume the total independence of propositions, being able to recognize when an action can add more than one proposition. Its calculation is based on the deterministic planner *GraphPlan* [Blum and Furst, 1997] briefly described as follows.

GraphPlan. The *GraphPlan* [Blum and Furst, 1997] is divided into two phases. In the first, a planning graph is constructed and in the second, the graph is used for the extraction of a plan solution, if there is one. The construction of the planning graph is done by intercalating layers of literals (positive and negative atomic propositions) and actions: the first layer contains all the literals of the initial state s_0 ; the second layer contains all the applicable actions in s_0 where each action is linked to the literals of its precondition in the previous layer; then a new layer is created containing all the literals of the first layer plus the literals of the effects of the actions of the previous layer, adding the links between actions and their respective effects. In each layer, conflicting actions (which could not be performed in any order without one interacting with each other) are marked *mutex* (mutual exclusion). Literals of the same layer can also be marked with *mutex*.

The heuristic h_{FF} [Hoffmann, 2001] constructs a relaxed planning graph from a state, ignoring the negative effects of the actions and returns as an estimate the cost of the plan extracted from that graph. If, during the construction of the graph, the goal is not reached, the heuristic returns the infinite value

Formally, let $\mathcal{P}^{det} = \langle \mathbb{P}, \mathbb{A}, C, s_0, \mathbb{G} \rangle$ be a deterministic planning problem and $\mathcal{M}(\mathcal{P}^{det}) = \langle S, A, T, C, s_0, G \rangle$ the respective state model. Given a state $s \in S$, the value of $h_{FF}(s)$ is the size of the solution plan (quantity of plan actions) obtained by Graphplan from the relaxed problem $P^+ = \langle \mathbb{P}, \mathbb{A}^+, C, s, \mathbb{G} \rangle$, where $\mathbb{A}^+$ is the set of actions without negative effects and s is the new initial state.

3.3.3 Heuristic h_m

The heuristic h_m [Haslum and Geffner, 2000] is computed in the space of atoms, that is, it is a function of the costs of the atoms of the problem, however, it does not ignore the negative effects of the actions. Instead of computing the cost to reach the atoms of a set of propositions independently, the heuristic h_m computes the cost to reach subsets of propositions with a limited cardinality $m \geq 1$. Let $\mathcal{P}^{det} = \langle \mathbb{P}, \mathbb{A}, C, s_0, \mathbb{G} \rangle$ be a deterministic planning problem. Given a set of propositions $\Delta \subseteq \mathbb{P}$ to be reached from the state $s \in S$, the heuristic h_m is defined recursively:

$$h_m(s, \Delta) = \begin{cases} 0, & \text{if } \Delta \subseteq s, \\ \infty, & \text{if } \forall a \in \mathbb{A} : R(\Delta, a) = \perp, \\ \min_{a \in \mathbb{A}} [h_m(s, R(\Delta, a)) + C(s)] & \text{if } |\Delta| \leq m, \\ \max_{\Delta' \subseteq \Delta, |\Delta'| \leq m} h_m(s, \Delta') & \text{if } |\Delta| > m, \end{cases} \quad (3.4)$$

where $R(\Delta, a)$ is the regression operation of a set of propositions $\Delta \subseteq P$ by an action $a \in A$, defined as:

$$R(\Delta, a) = \begin{cases} (\Delta \setminus \text{App}(a)) \cup \text{Precond}(a), & \text{if } \text{Add}(a) \subset \Delta \text{ and } \text{Del}(a) \cap \Delta = \emptyset, \\ \perp, & \text{otherwise.} \end{cases} \quad (3.5)$$

Equation 3.4 says that if all propositions of Δ are already true in state s then the cost of Δ is null; if no action can add some atom of Δ , then it is impossible to reach Δ from s , and the cost for Δ is infinite; If Δ have until m propositions then the cost is defined recursively by the action of lower cost relevant to Δ according to the regression of Δ (Equation 3.5); and if Δ has even m propositions the cost is defined recursively in function of the set of propositions of greater cost with up to m atoms. Thus, the cost to reach the goal G from the state $s \subseteq \mathbb{P}$, according to the heuristic h_m , is given by: $h_m(s)$

$$h_m(s) = h_m(s, \mathbb{G}).$$

3.3.4 Heuristic h_{m-m}

The heuristic *min-min* (h_{m-m}), unlike the previous ones, is computed in state space. It calculates the minimum accumulated cost to reach a state considering the lower cost effect. The heuristic h_{m-m} is defined recursively by the following equation [Bonet and Geffner, 2005]:

$$h_{m-m}(s) = \min_{a \in \text{App}(s)} \left\{ C(a) + \min_{s': P(s'|s, a) > 0} h_{m-m}(s') \right\}. \quad (3.6)$$

Where $h_{m-m}(s) = 0, \forall s \in G$. Note that this heuristic is very optimistic, since it assumes that a probabilistic transition will go to the lowest cost state. Considering actions with unit cost, this heuristic returns the minimum number of steps that the agent takes to a target state.

3.3.5 Heuristic HMDPP

Another way to relax a probabilistic problem is the *self-loop* relaxation, which considers that the effects of actions are independent, each effect occurring with probability p_i and with probability $1 - p_i$ it fails and leaves the agent in the same state.

To solve this relaxed version of the problem, [Keyder and Geffner, 2008] observed that uncertainty could be removed from the problem by compiling the probabilities in the costs of a deterministic action. This process is called *self-loop* determinization. Given a probabilistic planning problem $\mathcal{P}_i^{prob}$, the *self-loop* determinization creates a deterministic $\mathcal{P}_i^{det}$ version of relaxed problem, whose solution is equals to the *self-loop* relaxation.

As with all outcomes determinization, the determinization of the *self-loop* creates a new deterministic action for each effect of the action, however, the cost of the new deterministic action is defined by the cost divided by the probability of the original effect occurring:

$$C(a_i) = \frac{C(a)}{p_i}.$$

Note that in this way the lower the probability of an effect occurring the greater the cost of the respective deterministic action, so the solution of the problem determinized gives greater preference to the effects of greater probability.

The planner HMDP (HMDPP) [Keyder and Geffner, 2008] is a probabilistic planner that is based on a determinization process. This planner creates a plan for the relaxed (deterministic) version and follow this plan, re-planning if any unexpected effects happen.

3.4 The Projection Occupation Measure Heuristic

The classical planning heuristics described in the previous sections are proven to be efficient, but to be used in probabilistic planning they require the determinization of the original problem, which takes the action cost into account but forgets information about the original state transition probabilities.

A recent work proposes the first heuristic to take into account the probabilities, called h_{pom} , which solves a relaxed version of an SSP modeled as a linear program in the dual space and considers both, the action probabilities and actions cost [Trevizan et al., 2017b]. The heuristic h_{pom} turns out to be efficient for SSPs, but can not deal alone with SSPUDE problems, neither with MAXPROB problems, as we are going to show in Chapter 4.

3.4.1 An SSP as a Primal and Dual Linear Program

In order to explain the heuristic h_{pom} , we need first to introduce the formulation of an SSP as a linear program. Let us first formulate an SSP as a linear optimization problem in the primal space. The most common formulation is one that optimizes the value function of a policy as the following [Puterman, 1994b]:

$$\begin{aligned}
& \underset{v}{\text{maximize}} && \sum_{s \in S} v_s \\
& \text{subject to} && v_s \geq 0 \\
& && v_s = 0, \quad \forall s \in G \\
& && v_s \leq C(s, a) + \sum_{s' \in S} P(s'|s, a)v_{s'}, \quad \forall s \in S \setminus G, a \in App(s).
\end{aligned} \tag{3.7}$$

where variables v_s represent the expected accumulated cost $V(s)$ of reaching the goal from a state s . Note that solving this linear program is equivalent to solve the Bellman Equation for SSPs (Equation 2.13).

Based on the primal space formulation [Altman, 1999] proposed a solution for SSPs in the dual space. This new formulation was rewritten by [Trevizan et al., 2017b], which interpreted it as a network flow problem. In this formulation, the objective is to optimize a set of occupation measure variables that are used to represent the expected number of times an action $a \in App(s)$ is executed in the state s . The linear program in dual space is defined as following:

$$\begin{aligned}
& \underset{x}{\text{minimize}} && \sum_{s \in S, a \in App(s)} x_{s,a} C(s, a) && \text{(LP1)} \\
& \text{subject to} && x_{s,a} \geq 0 && \forall s \in S, a \in App(s) && \text{(C1)} \\
& && out(s) = \sum_{a \in App(s)} x_{s,a} && \forall s \in S && \text{(C2)} \\
& && in(s) = \sum_{s' \in S, a \in A(s')} x_{s',a} P(s|s', a) && \forall s \in S, a \in App(s) && \text{(C3)} \\
& && out(s) - in(s) = 0 && \forall s \in S \setminus \{G \cup s_0\} && \text{(C4)} \\
& && out(s_0) - in(s_0) = 1 && && \text{(C5)} \\
& && \sum_{s_g \in G} in(s_g) = 1 && && \text{(C6)}
\end{aligned}$$

This new linear program, called LP1, can be interpreted as a flow problem, where C2 and C3 define expected flow entering and leaving the state s , respectively; constraint C4 is the flow conservation principle of SSP with no dead ends, i.e., all flows reaching s must leave s (for all states $s \in S$ that are neither the initial state s_0 , called source, nor a goal state, called sink); and C5 and C6 define, respectively, the source and the sinks. The objective function captures the minimization of the total expected cost to reach the

goal from the initial state. The optimal solution of LP1 can be converted into an optimal policy $\pi^* = a$, where $a \in A(s)$ is any action such that $x_{s,a}^* \neq 0$ [Trevizan et al., 2017b].

Based on LP1, [Trevizan et al., 2017b] proposes the heuristic h_{pom} that relaxes program LP1 by projecting the problem onto individual state variables while enforcing the consistency of all projections. To do so, an SSP must be defined in terms of state variables, as in Definition 3.1.4.

Example: From primal to dual space linear program

To illustrate how the dual linear program can be interpreted as a network flow problem we define an SSP $M = \langle S, A, P, C, s_0, G \rangle$ where:

- $S = \{s_1, s_2, s_3\}$
- $A = \{a_1, a_2\}$.
- $C(s, a) = 1, \forall s \in S, \forall a \in A$.
- s_1 is the initial state.
- s_3 is the goal state.
- The probability transition function is represented as in Figure 3.1. Note that all actions in this example are deterministic.

Figure 3.1: Example of an SSP with deterministic actions and unitary costs, where s_1 is the initial state and s_3 is the goal state.

Following the LP formulation (Equation 3.7), the linear program in the primal space for the SSP M is given by:

$$\begin{aligned}
 & \underset{v}{\text{maximize}} && \sum_{s \in S} \alpha(s) v_s \\
 & \text{subject to:} && \\
 & && v_{s_1} \leq C_{s_1, a_1} + v_{s_2} \\
 & && v_{s_2} \leq C_{s_2, a_1} + v_{s_1} \\
 & && v_{s_2} \leq C_{s_2, a_2} + v_{s_3}
 \end{aligned}$$

where $\alpha(s)$ are known as the state-relevance weights. For the exact solution these values can be set to any positive number (say, to 1). Note that we set $v_1 = 1, v_2 = 0$, and $v_3 = -1$ in order to obtain the same constraints showed by [Trevizan et al., 2017a]. Also it is possible to rewrite the above linear program in matrix form as follow:

$$\underset{v}{\text{maximize}} \quad [1, 0, -1][v_{s_1}, v_{s_2}, v_{s_3}]^T$$

subject to:

$$\begin{array}{ccc} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & -1 \end{bmatrix} & \begin{bmatrix} v_{s_1} \\ v_{s_2} \\ v_{s_3} \end{bmatrix} & \leq \begin{bmatrix} C_{s_1, a_1} \\ C_{s_2, a_1} \\ C_{s_2, a_2} \end{bmatrix} \\ A & x & \alpha \end{array} \quad (3.8)$$

This formulation (Equation 3.8) follows the form described in Equation 8.8 and can be transformed to the dual space following the Equation 8.9. Then the linear program in the dual space for the SSP M is the follow:

$$\min \begin{array}{c} \begin{bmatrix} C_{s_1, a_1} \\ C_{s_2, a_1} \\ C_{s_2, a_2} \end{bmatrix}^T \\ b^T \end{array} \begin{array}{c} \begin{bmatrix} y_{s_1, a_1} \\ y_{s_2, a_1} \\ y_{s_2, a_2} \end{bmatrix} \\ y \end{array}$$

subject to:

$$\begin{array}{ccc} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 0 & -1 \end{bmatrix} & \begin{bmatrix} y_{s_1, a_1} \\ y_{s_2, a_1} \\ y_{s_2, a_2} \end{bmatrix} & \leq \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \\ A^T & y & \alpha \end{array} \quad (3.9)$$

The equation 3.9 can be rewritten to the form:

$$\underset{s \in S, a \in \text{App}(s)}{\text{minimize}} \quad x_{s,a} C(s, a)$$

subject to:

$$\begin{aligned} x_{s_1, a_1} - x_{s_2, a_1} &= 1 \\ -x_{s_1, a_1} + x_{s_2, a_1} + x_{s_2, a_2} &= 0 \\ -x_{s_2, a_2} &= -1 \end{aligned}$$

Note that in this formulation the variables $x_{s,a}$ are called *occupation measures variables*. They represent the number of times an action a is executed in a state s . In [Trevizan et al., 2017a] the constraints of this linear program are interpreted as a flow network problem, where for each state s an in-coming flow $in(s)$ and an out-coming flow $out(s)$ are defined. The difference between the out-coming and in-coming flow for a state s should be 0 in order to keep the flow conservation principle; except for the initial state and for the final state, which must be as follow:

$$out(s) - in(s) = \begin{cases} 1 & \text{if } s = s_0 \\ -1 & \text{if } s \in G \\ 0 & \text{otherwise} \end{cases}$$

Following the above description we can differentiate the in-coming and out-coming flow for each state $s \in S$ of the SSP M for state s_1 :

$$\begin{aligned} out(s_1) &= x_{s_1, a_1} \\ in(s_1) &= x_{s_2, a_1} \\ x_{s_1, a_1} - x_{s_2, a_1} &= 1, \end{aligned}$$

for state s_2 :

$$\begin{aligned} out(s_2) &= x_{s_2, a_1} + x_{s_2, a_2} \\ in(s_2) &= x_{s_1, a_1} \\ x_{s_2, a_1} + x_{s_2, a_1} - x_{s_1, a_1} &= 0, \end{aligned}$$

and for state s_3 :

$$\begin{aligned} out(s_3) &= x_{s_3, a_1} \\ in(s_3) &= x_{s_2, a_2} \\ x_{s_2, a_1} &= -1. \end{aligned}$$

Note that the flow value of the initial state s_1 represent the initial incoming flow of the problem (1 unit), it pass to the state s_2 and is consumed completely to end in the goal state s_3 which recovers all the initial flow.

3.4.2 The Heuristic h_{pom}

Trevizan et al. (2017) proposed a domain independent heuristic based on the dual linear program LP1. For that, an SSP (with no dead ends) has to be represented in the planning description language SAS⁺ (Definition 3.1.4).

Projection of an SSP. Given a SAS⁺ probabilistic task $\langle \mathbb{V}, A, s_0, s_*, C \rangle$, the projection of variable $v \in V$ from s onto v is the SSP $\mathcal{M}^{v,s}$. This new SSP has an extra artificial action a_g leading the agent to an absorbing state g when v reaches a goal value. This is necessary to synchronize all projections of a state. Thus, the projection is defined as [Trevizan et al., 2017a]:

$$\mathcal{M}^{v,s} = \langle D_v \cup \{g\}, A \cup \{a_g\}, P, C', s[v], \{g\} \rangle,$$

where $C'(d, a_g) = 0$, $C'(d, a) = C(a)$, $\forall a \in A$, $d \in s$ and

$$P(d'|d, a) = \begin{cases} \sum_{\substack{e \in eff(a) \text{ s.t.} \\ e[v]=d'}} Pr_a(e), & \text{if } d \neq d', a \in A, pre(a)[v] \in \{d, \perp\} \\ \sum_{\substack{e \in eff(a) \text{ s.t.} \\ e[v] \in \{d, \perp\}}} Pr_a(e), & \text{if } d = d', a \in A, pre(a)[v] \in \{d, \perp\} \\ 1, & \text{if } d = g, a = a_g, s_*[v] \in \{d, \perp\} \\ 0, & \text{otherwise,} \end{cases}$$

$\forall d \in D_v, \forall d' \in D_v \cup \{g\}$ and $\forall a \in A \cup \{a_g\}$.

Given $v \in V$, let $C^{v,s}$ be the flow constraints (C1) - (C6) of the dual formulation of $\mathcal{M}^{v,s}$. Each occupation measure variable of $\mathcal{M}^{v,s}$ is $x_{d,a}^{v,s}$, for $d \in D_v$ and $a \in A \cup \{a_g\}$, representing the expected number of times a is executed in the (abstract) state d of the SSP $\mathcal{M}^{v,s}$.

Tying Constraints. To tie the occupation measure variables and constraints, for each state s , into a single linear program, we must include the following set of tying constraints, denoted by $Tying^s$:

$$\sum_{d_i \in D_{v_i}} x_{d_i, a}^{v_i, s} = \sum_{d_j \in D_{v_j}} x_{d_j, a}^{v_j, s} \quad \forall v_i, v_j \in V, a \in A.$$

The $Tying^s$ constraints ensure that policies for each projection agree on the expected number of times an action is executed.

Definition 3.4.1. Projection Occupation Measure Heuristic (h_{pom}). Given a probabilistic SAS⁺ task $\langle \mathbb{V}, A, s_0, s_*, C \rangle$, the *projection occupation measure heuristic* h_{pom} at state s is the solution of the following dual linear program formulation [Trevizan et al., 2017b]:

$$h_{pom}(s) = \min_x \sum_{\substack{d \in D_v \\ a \in A}} x_{d,a}^{v,s} C'(d, a) \text{ s.t. } Tying^s, C^{v',s} \forall v' \in V. \quad (\text{LP2})$$

The formulation of h_{pom} is similar to the LP1 formulation, with $C^{v',s}$ representing the flow constraints (C1-C6) of the dual formulation $\mathcal{M}^{v,s}$ which includes new occupation measure variables $x_{d,a}^{v,s}$ and $Tying^s$ constraints over them.

3.4.3 Empirical results of the use of h_{pom} to solve SSPs e SSPs with unavoidable dead-ends

The domains used in Trevizan et.al. (2017) contain dead ends and in this case, it is shown the heuristic h_{pom} not always returns good estimations. To overcome this limitation, the authors: (i) *relax this (no dead end) assumption using the fixed-cost penalty formulation of dead ends* in the search algorithm, and (ii) run experiments with a combination of two heuristics, h_{pom} and h_{max} as a dead end detector [Bonet and Geffner, 2001] (which is a determinisation-based heuristic for cost-estimation).

		h_{max}	$h_{max} \& h_{pom}$
Exploding BW	7	30	30
	8	30	0
	9	30	30
	10	30	0
	11	0	0
	12	0	0
	15	0	0
Tri. Tire	3	30	30
	4	30	30
	5	30	0
	6	0	0

Table 3.1: Coverage (# of instances solved out of 30) for **SSPs with dead ends** (table extracted from Table 1 in [Trevizan et al., 2017b]), using LRTDP with heuristics h_{max} and $h_{max} \& h_{pom}$.

Table 3.1 shows the empirical results extracted from [Trevizan et al., 2017b] of LRTDP with fixed-cost penalty criterion (denoted by LRTDP+pe) [Trevizan et al., 2017b] using heuristics h_{max} and h_{pom} augmented with h_{max} (denoted by $h_{max} \& h_{pom}$) [Trevizan et al., 2017b]). In the Exploding Blocks World and Triangle Tire World domains (which can contain dead ends), $h_{max} \& h_{pom}$ was not able to solve many instances (some of them solved by h_{max} alone).

Chapter 4

The Heuristics h_{pom}^{pe} and h_{ppom}

In this Chapter, we propose two new heuristics based on h_{pom} . The first, called h_{pom}^{pe} , estimates the minimal cost of reaching the goal from a state s , including new variables and constraints for dead-ends and adding an expected penalty for reaching them. The second, called h_{ppom} , estimates the maximum probability of reaching the goal from s , and is used to solve MAXPROB problems by ignoring action costs. We claim that h_{ppom} is the first effective heuristic for MAXPROB. We also make an empirical evaluation of the proposed heuristics over domains selected from the International Probabilistic Planning Competition. The results show that h_{pom}^{pe} can solve larger planning instances when compared to previous solutions using only h_{pom} .

4.1 The Heuristic h_{pom}^{pe}

In this section we propose a modification of h_{pom} that is able to detect dead-ends directly, i.e. without using h_{max} , and can efficiently solve SSPUDE problems with a heuristic search algorithm. The new heuristic, called *projection occupation measure with penalty heuristic* (h_{pom}^{pe}) can estimate the accumulated cost function of states by extending h_{pom} as follows: (i) it adds a new set of occupation measure variables over dead-ends ($x_{d,a}^{DE}$) with new constraints over them, and (ii) it adds an expected penalty for reaching a dead-end state. This new heuristic is based on the same relaxation done for h_{pom} but now over the LP formulation to solve SSPs with dead-ends (i.e. SSPUDEs) [Trevizan et al., 2017b].

4.1.1 Linear Program for SSP with penalty

Following the idea of Kolobov ([Kolobov et al., 2012, Trevizan et al., 2017a]), it is possible to define an SSP with a fixed penalty as a linear optimization problem as follows:

$$\begin{aligned}
 & \underset{x}{\text{minimize}} && \sum_{s \in S, a \in \text{App}(s)} x_{s,a} C(s, a) + \sum_{s \in S} x_s^{DE} pe \\
 & \text{subject to} && (C1), (C3), (C4), (C5), (C7), (C8) \text{ and } (C9) \\
 & && x_s^{DE} \geq 0 && \forall s \in S, a \in \text{App}(s) && (C7) \\
 & && out(s) = \sum_{a \in \text{App}(s)} x_{s,a} + x_s^{DE} && \forall s \in S \setminus G && (C8) \\
 & && \sum_{s_g \in G} in(s) + \sum_{s \in S} x_s^{DE} = 1 && && (C9)
 \end{aligned}$$

The previous formulation represent the dual linear program for SSPs with a fixed penalty pe for dead-end states. For each state s , a new occupation measure variable x_s^{DE} is added to represent the flow escaping from s if s is proved to be a dead-end. Note that the x_s^{DE} variables are necessary in order to

maintain the flow conservation constraints as in LP1. The constraints (C1) - (C4) can be interpreted as in LP1, and the constraint (C8) ensures that if a flow is trapped in a dead-end state it will be forced to escape via x_s^{DE} . Finally constraint (C9) defines a *sink* that enforces that the flow not reaching the goal must be directed to the dead-end sink x_s^{DE} .

Definition 4.1.1. Projection Occupation Measure Heuristic with Penalty(h_{pom}^{pe}). Given a probabilistic SAS⁺ task $\langle \mathbb{V}, A, s_0, s_*, C \rangle$, the projection occupation measure heuristic with fixed-cost penalty pe at state s , called $h_{pom}^{pe}(s)$, is the solution of the following dual linear program:

$$h_{pom}^{pe}(s) = \min_x \sum_{d,a} x_{d,a}^{v,s} C'(d,a) + \sum_d x_d^{DE,s} pe \mid Cons, \forall v \in \mathbb{V}, \quad (\text{LP3})$$

where $Cons = Tying^s \cup Tying^{DE} \cup (D1 - D7)$; $Tying^{DE}$ are the tying constraints over variables x_d^{DE} ; and constraints (D1-D7), $\forall v' \in \mathbb{V}$, are:

$$x_{d,a}^{v',s} \geq 0, \quad \forall d \in D_{v'} \cup \{g\}, a \in A(d) \quad (\text{D1})$$

$$in(d) = \sum_{\substack{d' \in D_{v'} \cup \{g\} \\ a \in A(d')}} x_{d',a}^{v',s} P(d|d',a) \quad \forall d \in D_{v'} \cup \{g\} \quad (\text{D2})$$

$$out(d) - in(d) = 0, \quad \forall d \in D_{v'} \setminus \{s[v']\} \quad (\text{D3})$$

$$out(s[v']) - in(s[v']) = 1 \quad (\text{D4})$$

$$out(d) = \sum_{a \in A(d)} x_{d,a}^{v',s} + x_d^{DE,s}, \quad \forall d \in D_{v'} \quad (\text{D5})$$

$$in(g) + \sum_{d \in D_{v'}} x_d^{DE,s} = 1, \quad (\text{D6})$$

$$x_d^{DE,s} \geq 0, \quad \forall d \in D_{v'}. \quad (\text{D7})$$

In LP3, the variable pe is a large positive penalty for reaching a dead-end; which means that every state $d \in D_v$ such that $V^*(d) \geq pe$ is treated as a dead-end (this is similar to solve an SSP with a fixed-cost penalty in the search algorithm [Kolobov et al., 2012] (but here, the penalty is given in the heuristic). In LP3, we additionally consider the occupation measure variables $x_d^{DE,s}$ representing the outgoing flow from d which will be greater than 0 if d is a dead-end, and 0 otherwise; and D7 is a new set of constraints over $x_d^{DE,s}$. Note that this formulation extends the original h_{pom} heuristic (LP2) including in the objective function a *total expecting penalty for reaching dead-ends* ($\sum_{d \in D_v} x_d^{DE,s} pe$). Additionally, in order to ensure the flow conservation constraints, h_{pom}^{pe} replaces the constraints C5 and C6 of the formulation LP2 with constraints D5 and D6, respectively; and adds the constraints D7. The rationality of these changes is: *if the flow is trapped in a dead-end state d , the flow is forced to leave d using the occupation measure variable $x_d^{DE,s}$; and the constraint D6 ensures that the flows not reaching the goal state will not be lost at the end.*

Theorem 1. (Admissibility of h_{pom}^{pe}). For all states s of a given probabilistic SAS⁺ task, $h_{pom}^{pe}(s) \leq V^*(s)$.

Proof. To prove it we can consider two cases.

- If s is not a dead end e.g. $P_s^G = 1$. The heuristic $h_{pom}^{pe}(s)$ will behave like $h_{pom}(s)$ since the occupation measure variables for dead ends will never be used ($\sum_d x_d^{DE,s} pe = 0$). Since $h_{pom}(s)$ is an admissible heuristic then $h_{pom}^{pe}(s)$ is also an admissible heuristic for such states.
- If s is a state such that $P_s^G < 1$, every policy that starts from s is an improper policy i.e. $V^*(s) = \infty$. The occupation measure variables will be $\sum_d x_d^{DE,s} pe < V^*(s)$. This means that $h_{pom}^{pe}(s)$ is still admissible when s is a dead end.

□

4.2 The Heuristic h_{ppom}

We can also solve SSPUDES according to the two-stages optimization criterion [Kolobov et al., 2012]. However, one limitation of this approach is the non existence of heuristics to estimate the probability to reach a goal from state s .

In this section we propose the *probabilistic projection occupation measure heuristic* to solve MAXPROB problems, called h_{ppom} , that modifies the heuristic h_{pom} (LP2) w.r.t. the objective function as follows: (1) it maximizes the occupation measure variables that reach the goal; it ignores the action costs only considering the probabilities; and (ii) it modifies the cost function $C(d, a)$ to a *probability to immediately reach the goal*, called $C_P(d, a, d')$, where $C_P(d, a, d') = 1$ if a reaches a subgoal d' , and $C_P(d, a, d') = 0$, otherwise.

4.2.1 Linear Program for MaxProb

In order to define this new heuristic for MAXPROB we define the dual formulation for MAXPROB problems following the formulation showed in [Trevizan et al., 2017a].

$$\begin{aligned} & \underset{x}{\text{maximize}} && \sum_{s \in G} x_{s',a} P(s|s',a) \\ & \text{subject to} && x_{s,a} \geq 0 && \forall s \in S, a \in App(s) && (C1) \end{aligned}$$

$$\text{out}(s) = \sum_{a \in App(s)} x_{s,a}, \quad \forall s \in S \quad (C2)$$

$$\text{in}(s) = \sum_{s' \in S, a \in A(s')} x_{s',a} P(s, |s', a) \quad \forall s \in S, a \in App(s) \quad (C3)$$

$$\text{out}(s) - \text{in}(s) = 0 \quad \forall s \in S \setminus \{G \cup s_0\} \quad (C4)$$

$$\text{out}(s_0) - \text{in}(s_0) = 1 \quad (C5)$$

The previous linear formulation use the original SSP flow constraints (C4) and (C5), but now its represent the dual formulation for the MAXPROB problem, a unit flow is injected in s_0 and the objective is maximizing the flow reaching the sink, i.e the goal ($\text{in}(s_g) = \sum_{s \in G} x_{s',a} P(s|s',a)$).

Definition 4.2.1. Probabilistic Projection Occupation Measure Heuristic(h_{ppom}). Given a probabilistic SAS⁺ task $\langle \mathbb{V}, A, s_0, s_*, C_P \rangle$, the *probability projection occupation measure heuristic* at state s , called $h_{ppom}(s)$, is the solution of the following dual linear program:

$$h_{ppom}(s) = \underset{x}{\text{max}} \sum_{d,a,d'} x_{d,a}^{v',s} C_P(d, a, d') \mid Cons^*, \quad (LP4)$$

$\forall v \in \mathbb{V}$, where $Cons^* = Tying^s \cup (E1 - E5)$ and constraints $(E1 - E5)$, $\forall v' \in \mathbb{V}$, are:

$$x_{d,a}^{v',s} \geq 0, \quad \forall d \in D_{v'} \cup \{g\}, a \in A(d) \quad (E1)$$

$$\text{in}(d) = \sum_{\substack{d' \in D_{v'} \cup \{g\} \\ a \in A(d')}} x_{d',a}^{v',s} P(d, |d', a) \quad \forall d \in D_{v'} \cup \{g\} \quad (E2)$$

$$\text{out}(d) - \text{in}(d) = 0, \quad \forall d \in D_{v'} \setminus \{s[v']\} \quad (E3)$$

$$\text{out}(s[v']) - \text{in}(s[v']) = 1 \quad (E4)$$

$$\text{out}(d) = \sum_{a \in A(d)} x_{d,a}^{v',s}, \quad \forall d \in D_{v'} \quad (E5)$$

Note that, since $C_P(d, a, d') = 1$ only if d' is the goal, otherwise it is $C_P(d, a, d') = 0$, we can rewrite LP4 as:

$$h_{ppom}(s) = \max[in(g)] \mid Cons^*, \quad (LP4')$$

where $in(g) = \sum_d x_{d,a_g}^{v,s}$ (which corresponds to the projection of the linear program LP2 from [Trevizan et al., 2017a] to MAXPROB).

Theorem 2. (Admissibility of h_{ppom}). For all states s of a given probabilistic SAS⁺ task, $h_{ppom} \geq V^*(s)$.

4.3 Example of h_{pom}^{pe} and h_{ppom} Computation

To illustrate the computation of heuristics h_{pom}^{pe} and h_{ppom} , consider a simple SSPUDE problem, as depicted in Fig. 4.1, with four states s_0, s_1, s_2 (dead-end) and s_g described as a probabilistic SAS⁺ task $\mathcal{T} = \langle \mathbb{V}, A, s_0, s_*, C_P \rangle$ as follows:

- $V = \{v_1, v_2\}$ such that $D_{v \in V} = \{d_0, d_1, d_2, d_3\}$;
- $A = \{a_1, a_2, a_3\}$ with:
 - $pre(a_1) = (d_0, d_0)$, $eff(a_1) = (d_1, d_3)$ with $Pr = 1$;
 - $pre(a_2) = (d_0, d_0)$, $eff(a_2) = (d_2, d_1)$ with $Pr = 1$;
 - $pre(a_3) = (d_1, d_3)$, $eff(a_3) = \{(d_1, d_3), (d_2, d_1), (d_3, d_3)\}$ with $Pr = \{0.2, 0.4, 0.4\}$, respectively;
- $s_0 = (d_0, d_0)$;
- $s_* = s_g = (d_3, d_3)$; and
- $C(s, a) = 1, \forall s \in S, \forall a \in A$.

Thus, the states in Fig. 4.1 are written in factored form as: $s_0 = (d_0, d_0)$, $s_1 = (d_1, d_3)$, $s_2 = (d_2, d_1)$ and $s_g = (d_3, d_3)$.

Figure 4.1: State space of a simple SSPUDE described as a SAS⁺ task $\mathcal{T}$. Actions a_1 and a_2 are deterministic, and a_3 has three probabilistic outcomes.

To compute the heuristic h_{pom}^{pe} for state s_0 , we need to compute the projection of s_0 onto its state variables, that are v_1 and v_2 . The projection of s_0 onto v_1 results in the SSP $\mathcal{M}^{v_1, s_0} = \langle S', A', P', C', s'_0, s'_g \rangle$ (Fig. 4.2), where :

- $S' = \{d_0, d_1, d_2, d_3, g\}$,
- $A' = \{a_1, a_2, a_3, a_g\}$,
- $C'(d, a_g) = 0$ and $C'(d, a) = C(s, a), \forall a \in A'$,
- $s'_0 = \{d_0\}$,
- $s'_g = \{g\}$, and

- the transition probability function P' is as shown in Fig.2 (with probability 1 for the deterministic actions).

Analogously, the projection of s_0 onto v_2 is the SSP $\mathcal{M}^{v_2, s_0}$ depicted in Fig. 4.3. Note that, for this projection, the probability function for d_3 with action a_3 is now $P'(d_3|d_3, a_3) = 0.2 + 0.4 = 0.6$ and $P'(d_1|d_3, a_3) = 0.4$.

Figure 4.2: Projection $\mathcal{M}^{v_1, s_0}$ for task $\mathcal{T}$.

The result of occupation measure variables $x_{d,a}^{v_1, s_0}$ and $x_{d,a}^{v_2, s_0}$ after solving LP3 with $pe = 10$ is shown in Table 4.1, resulting in $h_{pom}^{pe}(s_0) = 1 + 1.25 + 0.5 + 0.5 \cdot 10 = 7.75$, considering the projection of v_1 . Note that due to the *Tying* ^{s_0} constraints the value of $h_{pom}^{pe}(s_0)$ is the same considering the projection of v_2 .

d	a	$\mathcal{M}^{v_1, s_0}$		$\mathcal{M}^{v_2, s_0}$	
		$x_{d,a}^{v_1, s_0}$	x_d^{DE, s_0}	$x_{d,a}^{v_2, s_0}$	x_d^{DE, s_0}
d_0	a_1	1	0	1.375	0
d_0	a_2	0	0	0	0
d_1	a_3	1.25	0	0	0.5
d_2	-	-	0.5	-	0
d_3	a_3	-	0	0	0
d_3	a_g	0.5	0	1.375	0

Table 4.1: Occupation measure values for LP3 with $pe = 10$.

Figure 4.3: Projection $\mathcal{M}^{v_2, s_0}$ for task $\mathcal{T}$.

To illustrate the computation of h_{ppom} for the MAXPROB version of the SSPUDE of Fig. 4.1, consider the same SAS⁺ task $\mathcal{T}$ but with a new cost (probability) function C_P , as defined in Table 4.2. The new generated projections, $\mathcal{M}^{v_1, s_0}$ and $\mathcal{M}^{v_2, s_0}$, are the same depicted in Fig. 4.2 and Fig. 4.3 without the

dashed lines (since now there are no occupation measure variables for dead-ends). Table 4.3 shows the occupation measure values after solving LP4, resulting in $h_{ppom}(s_0) = 0.4$, that happens to be the optimal MAXPROB value for task $\mathcal{T}$.

d	a	d'	$C_P(d, a, d')$
d_0	a_1	d_1	0
d_0	a_2	d_2	0
d_1	a_3	d_1	0
d_1	a_3	d_2	0
d_1	a_3	d_3	0
d_3	a_g	g	1

Table 4.2: Cost function C_P of the SAS⁺ task $\mathcal{T}$ in Fig. 4.2.

d	a	$x_{d,a}^{v_1, s_0}$	$x_{d,a}^{v_2, s_0}$
d_0	a_1	0.8	1.0
d_0	a_2	0.2	0.0
d_1	a_3	1.0	-
d_3	a_3	-	0.0
d_3	a_g	0.4	1.0

Table 4.3: Occupation measure values for LP4.

4.4 Empirical Analysis

The goal of this empirical analysis is to evaluate the two proposed heuristics, h_{pom}^{pe} and h_{ppom} to solve SSPUDE instances using the two different optimization criteria: (MAXPROB, MINCOST) and FCP. We compare the results in terms of average computational time and the number of explored states. We also evaluate the quality of the returned policy in terms of its probability to reach the goal from s_0 . We expected that, using both optimization criteria, (MAXPROB, MINCOST) and FCP, we should find policies with the same maximum probability to the goal.

To analyse the h_{ppom} heuristic to solve SSPUDE problems, we show the results of applying the (MAXPROB, MINCOST) optimization criterion, running the FRET algorithm, for the first stage and LRTDP for the second stage (Tables 4.4 and 4.5).

To analyse the h_{pom}^{pe} heuristic, we show the results of running the LRTDP+*pe* algorithm, using the FCP optimization criterion (Table 4.6).

We solved problems on 5 planning domains, running the algorithms 10 times for each instance and computing the average time, with a maximum of 15 minutes and 3GB of memory in an intel i7 processor at 2.7Hz. To solve the linear programs to compute the heuristics, we used the Gurobi7.5 solver. All algorithms were implemented in a probabilistic version of the Fast Downward framework [Steinmetz et al., 2016]. To run our tests, we choose different IPPC domains that can have dead-end states, as described bellow.

Exploding Blocks World Domain. This domain is an extension of the traditional Blocks World domain, where an agent is able to move blocks to the top of other blocks or to the table. When a block A is placed on a block B, there is a probability that A explodes B (it is said that A is detonated and B is destroyed). A block can only be detonated once and the destroyed blocks can no longer be moved. A state in which a block has been destroyed not in its final position is a dead-end state.

Triangle Tire World Domain. This is a standard domain for probabilistic planning with dead-ends, where a car can move to a different location through routes. The objective is to go from an initial location loc_0 to a goal location loc_g . In each movement, there is a probability of puncturing a tire. States with a flat-tire and no spare-tire are dead-ends.

Drive Domain. In this domain a car moves in a network of roads and traffic lights. When moving through the roads the driver can die which constitutes a dead-end state. The probability of dying depends on the car direction, the width of the road, the traffic light color and the routes priority.

Elevators Domain. In this domain there is a building with $N + 1$ floors, numbered from 0 to N . The building can be separated in blocks of size $M + 1$, where M divides N . Adjacent blocks have a common floor. For example, suppose $N = 12$ and $M = 4$, then we have 13 floors in total (ranging from 0 to 12), which form 3 blocks of 5 floors each, being 0 to 4, 4 to 8 and 8 to 12. The building has K fast elevators that stop only in floors that are multiple of $M/2$ (so M has to be an even number). Each fast elevator has a capacity of X persons. Furthermore, within each block, there are L slow elevators, that stop at every floor of the block. Each slow elevator has a capacity of Y persons. The goal of this domain is to take people from one floor to another.

Rectangle Tire World Domain. This domain is a variant of the Triangle Tire World domain, where the network is a rectangular grid, in this domain there is no flat tires, but exists a probability of failure for each connection. All connections have the same failure probability except on some specific rows, columns or locations.

4.4.1 Analysing the performance of h_{ppom}

Tables 4.4 and 4.5 show the results of solving 37 SSPUDE instances of the 5 analyzed probabilistic planning domains, applying the (MAXPROB, MINCOST) optimization criterion. The MAXPROB optimization phase was implemented using FRET with heuristics h_1 , h_{max} and h_{ppom} . The MINCOST optimization phase was implemented using LRTDP with heuristic h_0 (a simple heuristic that assigns 0 to all states). As mentioned before, the version of h_{max} for MAXPROB phase, returns 0 if no solution is found, i.e., if a queried state is a dead-end, and 1 otherwise. The heuristic h_1 simply assigns 1 to all states. Both heuristics, h_1 and h_{max} are admissible for MAXPROB problem.

In both tables, "-" indicates the instances that were not solved due to lack of time or memory. Thus, the results show that using FRET with our proposed heuristic h_{ppom} was able to solve all instances of the analyzed domains, outperforming h_{max} and h_1 in terms of total average time and the MAXPROB phase average time. The importance of having an efficient heuristic to compute MAXPROB can be observed by the fact that average time consumed in this phase is about 90% the total average time consumed by the two phases.

The optimal values $V_p^*(s_0)$ were found by FRET with the three heuristics, which indicates that h_{ppom} is also admissible. This result can be explained by the fact that h_{ppom} in fact makes good estimate of the probabilities of reaching the goal, and by doing so it can make the search algorithm to visit less states, as shown in Table 4.5. As we can notice, the number of explored states is much lower when using the h_{ppom} heuristic. In particular, in the Exploding Blocks World domain, for the largest instance solved by h_{max} (instance 5), FRET with h_{ppom} explored 634,402 less states, i.e. 4,500 times less states.

Thus, we claim that h_{ppom} is the first effective heuristic for the MAXPROB problem.

4.4.2 Analysing the performance of h_{pom}^{pe}

Table 4.6 shows the results of solving the 37 SSPUDE instances of the 5 analyzed domains, applying the finite-cost penalty (FCP) optimization criterion (Equation 2.17), and using LRTDP+PE algorithm with the heuristics h_{max} , $h_{max}\&h_{pom}$ and our proposed heuristic h_{pom}^{pe} . In all experiments we fixed the penalty with $pe = 10^6$. The version of h_{max} used in this experiment is the traditional cost estimate heuristic [Bonet and Geffner, 2001]. The heuristic h_{pom} augmented with h_{max} (denoted $h_{max}\&h_{pom}$), returns the penalty pe if no solution is found by h_{max} , i.e., when the queried state is a dead-end; otherwise it returns the h_{pom} value. Since h_{pom} does not consider dead-end states it will not always return a good cost estimate.

The results in Table 4.6 show that using LRTDP+PE algorithm with the FCP optimization criterion and the heuristic h_{pom}^{pe} we were able to solve all instances of the analyzed domains, outperforming h_{max} and $h_{max}\&h_{pom}$, both in terms of total average time and number of states explored. It also returned policies with maximum probability values $V_p^\pi(s_0)$ that are equal to the $V_p^*(s_0)$ found by the (MAXPROB, MINCOST) optimization criterion (last column of Table 4.4). This is a clear indication that h_{pom}^{pe} is also admissible.

These results can be explained by the fact that h_{pom}^{pe} inherits from h_{pom} the capability of estimating the expected cost but, in the presence of dead-ends, h_{pom}^{pe} also includes in its estimate the expected penalty, thus accelerating the LRTDP convergence. While $h_{max}\&h_{pom}$ uses h_{max} to detect dead-ends assigning them a penalty pe , h_{pom}^{pe} is able to consider this penalty for any state that eventually leads to a dead-end. To confirm the better estimate done by h_{pom}^{pe} , Table 4.6 also shows the number of explored states of LRTDP+pe with the three heuristics. Notice that in the Triangle Tire World domain, LRTDP+pe with h_{pom}^{pe} explored less than 1,205 states to solve any instance; and in the Exploding Blocks World domain, for the largest instance solved by $h_{max}\&h_{pom}$, LRTDP+pe with h_{pom}^{pe} explored 734,994 less states.

		(MAXPROB, MINCOST) $\rightarrow$ Avg. Time (s)						$V_p^*(s)$
		Total	MAXPROB(h_1)	Total	MAXPROB(h_{max})	Total	MAXPROB(h_{ppom})	
Expl. Blocks	1	1.32	1.14	0.78	0.14	0.00	0.00	0.90
	3	49.32	48.72	1.12	0.18	0.00	0.00	0.60
	5	-	-	13.08	9.16	0.01	0.00	1.00
	7	-	-	-	-	0.02	0.02	1.00
	8	-	-	-	-	38.06	37.92	0.47
	10	-	-	-	-	21.75	12.24	0.21
	11	-	-	-	-	56.21	49.52	0.49
Tri. Tire	1	1.23	0.16	0.78	0.12	0.06	0.04	0.23
	3	1.38	0.14	1.03	0.08	0.06	0.06	1.00
	5	-	-	-	-	1.72	1.50	0.58
	7	-	-	-	-	3.04	2.82	1.00
	9	-	-	-	-	3.01	2.54	0.84
	11	-	-	-	-	3.24	2.96	1.00
	13	-	-	-	-	6.34	5.90	1.00
	15	-	-	-	-	4.16	3.80	0.94
Drive	1	0.00	0.00	0.00	0.00	0.00	0.00	0.86
	3	0.00	0.00	0.00	0.00	0.00	0.00	0.82
	5	0.06	0.00	0.04	0.00	0.00	0.00	0.84
	7	0.12	0.02	0.05	0.00	0.00	0.00	0.67
	9	0.12	0.04	0.07	0.02	0.01	0.00	0.31
	11	0.32	0.12	0.09	0.02	0.01	0.00	0.63
	13	0.24	0.08	0.11	0.06	0.03	0.00	0.38
	15	0.45	0.26	0.21	0.12	0.07	0.01	0.45
Elevator	1	0.02	0.00	0.00	0.00	0.02	0.00	0.99
	3	0.03	0.00	0.02	0.00	0.02	0.00	1.00
	5	0.07	0.03	0.31	0.20	0.02	0.00	0.93
	7	0.35	0.24	0.92	0.73	0.09	0.02	1.00
	9	1.62	1.42	1.73	1.64	0.47	0.36	0.97
	11	3.92	3.62	2.71	2.54	0.90	0.78	0.84
	13	6.01	5.29	4.07	3.60	1.93	1.63	0.98
	15	11.82	10.72	6.29	5.82	4.15	3.77	0.87
Rect. Tire	1	1.61	1.53	0.72	0.64	0.00	0.00	1.00
	3	3.65	2.95	1.66	1.35	0.16	0.15	1.00
	5	3.83	3.60	2.79	1.92	0.67	0.59	0.58
	7	-	-	5.83	4.72	1.49	1.32	0.79
	9	-	-	-	-	2.93	2.62	1.00
	11	-	-	-	-	4.98	4.19	0.29

Table 4.4: Evaluating the Heuristic h_{ppom} : Average time for solving an SSPUDE using the (MAXPROB, MINCOST) criterion and varying the heuristics in the MAXPROB optimization phase. The *Total* columns show the total average time to solve the two optimizations using different heuristics; the *MaxProb* columns show the average time only for solving the MAXPROB optimization phase using FRET with the three heuristics: h_1 , h_{max} and h_{ppom} . The last column, $V_p^*(s_0)$, shows the probability to reach the goal from s_0 , of the returned policies.

		MAXPROB $\rightarrow$ Avg # States Explored		
		MAXPROB(h_1)	MAXPROB(h_{max})	MAXPROB(h_{pom})
Expl. Blocks	1	81693	14416	112
	3	1966479	15277	192
	5	-	634543	141
	7	-	-	594
	8	-	-	181788
	10	-	-	202896
	11	-	-	1140701
Tri. Tire	1	16830	7633	93
	3	19890	8964	31
	5	-	-	148
	7	-	-	507
	9	-	-	421
	11	-	-	37
	13	-	-	1010
	15	-	-	591
Drive	1	38	17	17
	3	75	27	27
	5	143	113	94
	7	369	241	174
	9	603	459	235
	11	1035	475	292
	13	1355	1049	524
	15	2189	1706	504
Elevator	1	909	834	280
	3	909	875	234
	5	909	901	242
	7	14787	12535	455
	9	14787	13626	883
	11	538299	274367	3557
	13	538299	286262	4179
	15	538326	356920	3914
Rect. Tire	1	47	49	47
	3	90	97	90
	5	82	25	28
	7	-	25	59
	9	-	-	153
	11	-	-	214

Table 4.5: Evaluating the heuristic h_{ppom} : number of states explored for solving an SSPUDE using the MAXPROB optimization criterion with three heuristics: h_1 , h_{max} and h_{ppom} .

		Fixed-cost Penalty (LRTDP + pe)						$V_p^\pi(s_0)$
		Average Time (secs)			Average # States Explored			
		h_{max}	$h_{max} \& h_{pom}$	h_{pom}^{pe}	h_{max}	$h_{max} \& h_{pom}$	h_{pom}^{pe}	
Expl. Blocks	1	0.48	0.53	0.00	17402	17428	152	0.90
	3	1.92	2.09	0.00	20748	21375	251	0.60
	5	13.03	27.77	0.07	734961	735169	175	1.00
	7	-	-	4.54	-	-	835	1.00
	8	-	-	47.96	-	-	253561	0.47
	10	-	-	38.54	-	-	415803	0.21
	11	-	-	153.87	-	-	1738105	0.49
Tri. Tire	1	1.63	1.70	1.13	7835	5263	126	0.23
	3	4.53	5.29	3.54	8153	5616	82	1.00
	5	14.03	8.14	8.02	10435	6825	193	0.58
	7	25.83	-	14.23	14546	-	631	1.00
	9	31.90	-	21.32	25463	-	515	0.84
	11	-	-	27.23	-	-	102	1.00
	13	-	-	31.59	-	-	1205	1.00
	15	-	-	35.94	-	-	714	0.94
Drive	1	0.00	0.00	0.00	38	17	17	0.86
	3	0.00	0.00	0.00	75	27	27	0.82
	5	0.02	0.02	0.00	143	113	94	0.84
	7	0.02	0.04	0.00	369	241	174	0.67
	9	0.04	0.04	0.01	603	459	235	0.31
	11	0.24	0.30	0.07	1035	475	292	0.63
	13	0.11	0.15	0.00	1355	1049	524	0.38
	15	0.22	0.26	0.05	2189	1706	504	0.45
Elevator	1	0.00	0.00	0.00	826	826	514	0.99
	3	0.00	0.00	0.00	835	838	514	1.00
	5	0.00	0.00	0.00	873	903	594	0.93
	7	0.16	0.16	0.01	893	910	539	1.00
	9	0.16	0.17	0.05	1953	2052	601	0.97
	11	8.14	7.40	2.01	2195	2195	634	0.84
	13	8.04	6.80	2.30	2285	2293	804	0.98
	15	8.10	7.13	2.21	2546	2361	825	0.87
Rect. Tire	1	0.34	0.38	0.15	51	51	47	1.00
	3	1.54	1.23	3.52	58	60	52	1.00
	5	13.32	12.45	11.53	79	79	90	0.58
	7	-	-	17.24	-	-	142	0.79
	9	-	-	20.53	-	-	125	1.00
	11	-	-	22.93	-	-	98	0.29

Table 4.6: Evaluating the heuristic h_{pom}^{pe} : Average time (s) and number of states explored for solving SSPUDE instances, under the finite-cost penalty (FCP) criterion, using LRTDP+PE algorithm with heuristics h_{max} , $h_{max} \& h_{pom}$ and h_{pom}^{pe} . The last column, $V_p^\pi(s_0)$, shows the probability to reach the goal from s_0 , of the returned policies.

4.5 Conclusions of Part I

Recently, [Trevizan et al., 2017b] proposed the h_{pom} heuristic which is considered the first admissible heuristic for stochastic shortest path (SSP) planning problems that takes probabilities into account. This heuristic solves a relaxed linear program in the dual space of an SSP. However, h_{pom} can fail in SSP problems with unavoidable dead-ends (denoted by SSPUDEs), even augmented with a heuristic to detect dead-ends such as h_{max} . Based on h_{pom} , in this work we propose two domain independent heuristics, that are:

- The heuristic h_{pom}^{pe} (*projection occupation measure with penalty heuristic*), which was created to solve an SSPUDE under the fixed-cost penalty optimization criterion (FCP) and outperforms the heuristic h_{pom} and $h_{max} \& h_{pom}$. This is because it is able to consider the penalty pe (used in the FCP criterion) in the expected cost estimate of any state s . The empirical results show that h_{pom}^{pe} is able to solve large instances that h_{pom} and $h_{max} \& h_{pom}$ were not able to solve.
- The heuristic h_{ppom} (*probabilistic projection occupation measure heuristic*), which is used in the first stage of the two-stages optimization criterion (MAXPROB, MINCOST). We claim that this is the first effective heuristic for MAXPROB problems to consider only the probabilities. The experiments show that using h_{ppom} with the FRET algorithm, it is possible to efficiently solve large instances of the tested domains.

Part II

Risk Sensitive Planning as a Linear Program in the Dual Space

Chapter 5

Background on Risk Sensitive SSPs with Exponential Utility Function

An important aspect in probabilistic planning is how to consider the risk in the process. In SSPs risk arises from the uncertainties on future events and how they can lead to goal states. An agent that minimizes the expected accumulated cost is considered a risk-neutral agent, while with a different optimization criterion an agent could choose between two attitudes: risk-aversion or risk-prone [Marcus, 1997, Shen et al., 2014]. There are different approaches to quantify risk in MDPs and SSPs, e.g.: (i) the use of an expected exponential utility [Marcus, 1997, Howard and Matheson, 1972, Jaquette, 1976, Denardo and Rothblum, 1979, Rothblum, 1984, Patek, 2001]; (ii) the use of a piece wise linear transformation function with a discount factor [Mihatsch and Neuneier, 2002]; (iii) weighted sum between expectation and variance [Sobel, 1982, Filar et al., 1989]; and (iv) the estimation of performance in a confidence interval [Filar et al., 1995, Yu et al., 1998, Hou et al., 2014, Hou et al., 2016]. However, due to the complexity of such approaches, finding optimal policies are computationally more costly than solutions for risk-neutral SSPs [García and Fernández, 2015]. In this work we use the exponential utility approach to model risk in probabilistic planning problems with no dead-ends.

5.1 Risk Sensitive Stochastic Shortest Path

One of the classical approaches to quantify risk is the exponential utility, which is based on the utility theory [Howard and Matheson, 1972, Jaquette, 1976, Denardo and Rothblum, 1979, Rothblum, 1984, Patek, 2001]. For this approach, we must define a risk factor, and the feasible values for this risk factor varies within problems and domains [Patek, 2001]. One solution to make every problem feasible is to consider a discount factor; however, in this case, the optimal policy is non-stationary [Chung and Sobel, 1987]. Thus, in this work we use a utility function u that takes into account a risk factor [Patek, 2001], that is:

$$u(C) = -\text{sgn}(\lambda)e^{(-\lambda C)}, \quad (5.1)$$

where λ is the risk factor and C is the undiscounted cumulative cost (i.e., $C = \sum_{t=0}^{\infty} c_t$) of an SSP as in Definition 2.4.2, i.e. without dead-end states. Thus, if $\lambda < 0$, the agent has a risk prone attitude, if $\lambda > 0$, the agent has a risk averse attitude, and when $\lambda \rightarrow 0$, the agent has a risk neutral attitude [Howard and Matheson, 1972].

Figure 5.1, shows the utility function curves of $u(C) = -\text{sgn}(\lambda)e^{(-\lambda C)}$ for $\lambda = 0.99$ and $\lambda = -0.99$. Note that the green curve that represents the risk-averse agent's attitude ($\lambda = 0.99$) is concave and the

Figure 5.1: Exponential utility function $u(C) = -\text{sgn}(\lambda)e^{(-\lambda C)}$ for $\lambda = 0.99$ and $\lambda = -0.99$; and utility function $u(C) = C$.

red curve that represents the risk-prone agent's attitude ($\lambda = -0.99$) is convex. The identity function $u(C) = C$ (blue line) represents a risk-neutral attitude.

Definition 5.1.1. (Risk Sensitive Stochastic Shortest Path (RS-SSP)) An RS-SSP is a tuple $M = \langle S, A, P, C, s_0, G, \lambda \rangle$ in which S is a finite set of states; A is a finite set of actions, $A(s) \subseteq A$ is a set of actions applicable in state s ; $P(s'|s, a)$ is the probability of taking action $a \in \text{App}(s)$ in state s ending in state s' ; $C(s, a)$ is the cost function that gives the immediate cost value for taking an action $a \in \text{App}(s)$ in state s ; $s_0 \in S$ is the initial state; G is the set of goal states and λ is the risk-attitude factor.

An RS-SSP is an SSP with an optimization criterion that takes into account a risk factor λ indicating if the agent is averse, neutral or prone to risk. In this work, we consider a solution for an RS-SSP based on an exponential value function, i.e. an exponential utility function (Figure 5.1), $\forall s \in S$ [Patek, 2001, Freire, 2016].

λ -feasible policy: A policy π of an RS-SSP is λ -feasible if the probability of not reaching a goal state vanishes faster than the expected exponential accumulated cost. If π is λ -feasible, then the value function of a policy π can be computed by solving the following system of equations [Freire and Delgado, 2017], $\forall s \in S$:

$$V^\pi(s) = \begin{cases} 1 \times \text{sgn}(\lambda) & \text{if } s \in G, \\ e^{\lambda C(s, \pi(s)) \sum_{s' \in S} P(s'|s, \pi(s)) V^\pi(s')} & \text{otherwise.} \end{cases} \quad (5.2)$$

If there is a λ -feasible policy, then the optimal value function $V^*(s) = \min_{\pi \in \Pi} V^\pi(s)$ is the solution of the following equation, $\forall s \in S$:

$$V^*(s) = \begin{cases} 1 \times \text{sgn}(\lambda) & \text{if } s \in G, \\ \min_{a \in A} \left[e^{\lambda C(s, a) \sum_{s' \in S} P(s'|s, a) V^*(s')} \right] & \text{otherwise,} \end{cases} \quad (5.3)$$

and an optimal policy can be obtained, $\forall s \in S$, from the optimal value function by:

$$\pi^*(s) = \underset{a \in A}{\text{argmin}} \left[e^{\lambda C(s, a) \sum_{s' \in S} P(s'|s, a) V^*(s')} \right] \quad (5.4)$$

5.2 The Risk in SSPs

Given that C_π represents the expected cost of a policy π , it is possible to define three attitudes in relation to risk: neutral, prone and averse [Keeney et al., 1979]. For this, we need to define a certainty equivalent of a policy.

[Patek, 2001] defines the certainty equivalent as the guaranteed cost that an agent prefers to pay, instead of risking to have a lower cost that is more uncertain. If $V^\pi(s) < \infty$ and there is an inverse function $u^{-1}(\mathbb{R}) \rightarrow \mathbb{R}^+$, the certainty equivalent $\bar{C}_\pi(s)$ of a policy π is defined as:

$$\bar{C}_\pi(s) = u^{-1}(V^\pi(s)),$$

and the expected cost $\tilde{C}_\pi(s)$ of a policy π is defined as:

$$\tilde{C}_\pi(s) = E[C_\pi | s_0 = s].$$

Thus, an agent is risk prone if $\bar{C}_\pi(s) < \tilde{C}_\pi(s)$; risk averse if $\bar{C}_\pi(s) > \tilde{C}_\pi(s)$ and risk neutral if $\bar{C}_\pi(s) = \tilde{C}_\pi(s)$ for all state $s \in S$ and policy $\pi \in \Pi$.

5.2.1 Example 1

Consider a simple SSP problem (Figure 5.2) with initial state s_0 and a goal state s_g . There are two available actions in s_0 ; action a_1 leading to the goal state with probability 1; action a_2 leading the goal state with probability $(1 - p)$ and the initial state with probability p . There are only two policies in this problem $\pi_1 = \{(s_0 : a_1), (s_g : a_3)\}$, $\pi_2 = \{(s_0 : a_2), (s_g : a_3)\}$.

Figure 5.2: Simple RS-SSP with a deterministic action a_1 and no dead-ends.

Following Equation 5.3 we obtain $V^{\pi_1}(s_0) = e^{\lambda C_1}$ and $V^{\pi_2}(s_0) = \frac{1-p}{1-pe^{\lambda C_2}}$, considering $\lambda > 0$. Note that the value function $V^{\pi_1}(s_0)$ is a fixed cost while the value function $V^{\pi_2}(s_0)$ is an expected cost.

If we estimate the value function of the policies π_1 and π_2 as in SSPs with Equation 2.13, the value of $V^{\pi_1}(s_0) = 2 \times c$ and $V^{\pi_2}(s_0) = 1.48 \times c$, this means that we prefer policy π_2 over π_1 . However when we evaluate the policies with the exponential value function (Equation 5.3) as shown in Figure 5.3 we see that depending on the λ value, either the policy π_1 or π_2 is preferred. In the example, π_2 is preferred if $\lambda < 2.4$, otherwise policy π_1 is preferred.

5.2.2 Example 2

In the River domain [Freire and Delgado, 2017], there is a grid with N_x lines and N_y columns where columns 1 and N_y are the riversides; line 1 contains a bridge and line N_x contains a waterfall (Figure 5.4). The agent must cross the river by: (i) swimming from any point of the riverside, or (ii) walking along the riverside and the bridge. However, the river flows to a waterfall (in $y = 1$). We resume the agent actions into: north($\uparrow$), south($\downarrow$), west($\leftarrow$) and east($\rightarrow$).

Actions executed on riversides or the bridge have 99% chance of success and 1% chance of failing (i.e., the agent remains in the same location). Actions executed inside the river have 80% chance of being dragged by the flow (towards the south) and 20% chance of success. In our experiments, the goal state is

Figure 5.3: Exponential accumulated cost of policies π_1 (green line) and π_2 (red line), varying according to the risk factor λ . We consider $C_1 = 2 \times c$ and $C_2 = c$ and $p = 0.3$.

Figure 5.4: Instance 7x10 of the River domain.

an absorbing state and if the agent falls into the waterfall, it returns to the initial state (so we do not consider dead-end states).

Note that policies for the River domain present a clear predicted behaviour in terms of risk attitude, i.e. in this domain we can easily depict a risk averse and a risk prone attitude.

- A **risk-prone agent** in the River domain always goes towards the goal location as fast as possible. If the agent is in a west riverside cell he would jump into the river towards the goal; and if he is in the east riverside he would go towards the goal along the riverside. When the agent is inside the river he would either tries to go toward east or north i.e., he will seek for the goal even under the risk of falling into the waterfall (see Figure 5.5a).
- A **risk-averse agent** in the River domain always goes towards the goal walking along the riversides or the bridge. If the agent is in the west (east) riverside, he tries to go towards north (south) along the riverside; if he is in the bridge he tries to go east. If the agent falls inside the river he would always try to achieve the nearest riverside (or bridge) location, i.e., avoiding the waterfall (see Figure 5.5b).

5.3 Solutions for RS-SSP with exponential utility

In this section we discuss different solutions for RS-SSP with exponential utility.

(a) Risk prone policy attitude.

(b) Risk averse policy attitude.

Figure 5.5: Extreme risk policies for instance 7×10 of the River Domain where (a) is the risk-prone policy with $\lambda = -0.99$, and (b) is the risk-averse policy with $\lambda = 0.99$.

5.3.1 A synchronous algorithm for RS-SSP: RS-PI

[Freire and Delgado, 2016] shows a modification of PI algorithm that allows to solve RS-SSPs iteratively updating all state space.

Algorithm 5 RS-PI

Input: An RS-SSP and a feasible λ value.

Output: Optimal policy π .

- 1: Choose an initial policy π_0 arbitrarily
- 2: $i \leftarrow 1$
- 3: $\pi_i(s) \leftarrow \underset{a \in A}{\operatorname{argmin}} [\exp(\lambda C(s, a)) \sum_{s' \in S} P(s'|s, a) V^{\pi_0}(s')]$
- 4: **while** $\pi_i \neq \pi_{i-1}$ **do**
- 5: Policy evaluation: obtain the value of the current policy π_i for every $s \in S$ solving the system of equations in Equation 5.2 .
- 6: Policy improvement : improve the current policy by doing the following update for every $s \in S$:

$$\pi_i(s) \leftarrow \underset{a \in A}{\operatorname{argmin}} [\exp(\lambda C(s, a)) \sum_{s' \in S} P(s'|s, a) V^{\pi_{i-1}}(s')]$$

- 7: $i \leftarrow i + 1$
 - 8: **end while**
 - 9: **return** π_i
-

In Algorithm 5 we first choose an arbitrarily policy (line 1), then calculate the next greedy policy (line 3). Lines 4-7 show how iteratively we improve the policy, which involves two steps: (i) *policy evaluation* where we obtain the value function for the current policy through the Bellman equation for RS-SSPs (Equation 5.3); (ii) *policy improvement*, where we compute an improved policy given the value function obtained in the previous step. Finally, when the policy converges we can return it as the optimal policy.

Since an RS-SSP problem is not well defined for every λ , i.e. we must guarantee to use a λ -feasible value, we must first find a maximum (extreme) value that is feasible and choose a λ value less or equal to this extreme. In Appendix E we discuss different algorithms to find this extreme risk λ factor ¹.

5.3.2 Asynchronous algorithms for RS-SSP

[Freitas, 2019] proposed the first heuristic search solution for risk-sensitive SSPs based on expected exponential utility [Patek, 2001]. In that work, the algorithm ILAO [Hansen and Zilberstein, 2001], a

¹A discussion of algorithms to find the extreme risk λ factor is kept in the Appendix E, since this is not the main focus of this work.

different implementation of the F&R framework (Algorithm 2), was implemented with a heuristic that considers the minimum local utility. This heuristic, which we will call h_{local}^{rs} , considers the risk factor and the lowest immediate cost for each non-goal state. Thus, the heuristic h_{local}^{rs} is defined as follows [Freitas, 2019]:

$$h_{local}^{rs}(s) = \begin{cases} 1 \times \text{sgn}(\lambda) & \text{if } s \in G, \\ 1 \times \text{sgn}(\lambda)e^{\lambda \min C(s,a)} & \text{otherwise.} \end{cases} \quad (5.5)$$

Theorem 3. (Admissibility of h_{local}^{rs}). For all states s of a given probabilistic RS-SSP, $h_{local}^{rs}(s) \leq V^*(s)$.

Proof. Since the $h_{local}^{rs}(s)$ returns the cost of the action applied in s , and $V(s)$ is the expected exponential accumulated cost, we can affirm that $h_{local}^{rs}(s) \leq V^*(s), \forall s \in S$, Therefore h_{local}^{rs} is admissible. $\square$

In Chapter 2 we have shown another asynchronous algorithm that is an implementation of the F&R framework, the LRTDP. Algorithm 6 modifies the algorithm LRTDP (Algorithm 3) for risk-sensitive SSP problems, updating the value function of the states visited in each trial according with Equation 5.3, assuming a λ -feasible value as input and a heuristic that can be the h_{local}^{rs} heuristic proposed by [Freitas, 2019], or any other heuristic that takes the risk into account (as the one we propose in Chapter 3). The remaining functions of the original LRTDP are the same.

Algorithm 6 RS-LRTDP algorithm

Input: RS-SSP $(\mathcal{M}, \lambda)$, heuristic function h

Output: V^*

```

1: function RS-LRTDP
2:   while  $s_0$ .SOLVED is false do
3:     LRTDP-TRIAL( $s_0$ )
4:   end while
5: end function
6: procedure INITIALIZE( $s, h$ )
7:    $V^0(s) \leftarrow h(s)$ 
8: end procedure
9: procedure Q-VALUE( $s, a$ )
10:   $q(s, a) = \text{LogSumExp}(e^{\lambda C(s,a)} P(s_g|s, a) + \sum_{s' \in S \setminus G} e^{\lambda C(s,a)} P(s'|s, a) V(s'))$ 
11:  return  $q(s, a)$ 
12: end procedure

```

In line 10, the procedure Q-VALUE (line 10) was modified with Equation 5.3. In line 11 we apply the *LogSumExp* transformation to avoid errors of numerical precision, which we explain below.

Numerical imprecision of exponential utility functions. When considering the expected exponential utility, a main limitation is the high numerical value resulting from the exponential value function computation. Depending on the parameters of the problem, the exponential values can become so large that they can not be processed using a 64-bit floating-point variable. Even when the computation of intermediate values is possible, the variation of exponents can cause errors of precision. To solve this type of numerical imprecision we use the *LogSumExp* technique [William et al., 2001, Freire and Delgado, 2017], which transforms the exponential growth of a function into an arithmetic growth through a logarithmic function. The definition of this transformation is showed in the Appendix A.

5.3.3 Finding a maximum λ -feasible

In the previous sections we show algorithms capable of determining an optimal policy for RS-SSPs. These algorithms assume that the risk factor is known, for prone risk attitude the value of λ can range from 0 to the negative infinity without problem. However, as we have mentioned before, when working with averse risk attitude the value of λ has to be feasible. This is because a very high value will not allow the algorithms to converge. [Freire and Delgado, 2016] proposed an algorithm to find this extreme-risk λ factor and the corresponding optimal policy based on Algorithm 5, i.e. PI for RS-SSP problems.

The Algorithm 7 was proposed by [Freire and Delgado, 2016] to find an approximate extreme λ -feasible and its respective policy. Given an initial $\lambda < 0$ and an arbitrary policy, the algorithm computes a new policy by updating λ while the constraint in line 6 is satisfied (where β is the approximation error). In each iteration the new policy is evaluated and improved. The algorithm ends when the policy converges with a ϵ -Residual error. Finally, the algorithm returns the last computed λ and the respectively policy.

In Algorithm 7, $\mathbf{P}^\pi$ is the matrix representation of the probability transition function P constrained to policy π , i.e., $P(s, \pi(s), s')$. $\mathbf{P}_{G^c}^\pi$ is the matrix $\mathbf{P}^\pi$ where the columns representing the states belonging to G are assigned to 0. Also $\mathbf{D}^\pi$ is the diagonal matrix $|S| \times |S|$ representing the exponential cost of the policy π . Finally, ρ represents the *spectral radius function* [Patek, 2001] which is responsible to detect a λ -feasible policy by the following rule:

$$\text{if } \rho((\mathbf{D}^\pi)^\lambda \mathbf{P}_{G^c}^\pi) < 1 \text{ then policy } \pi \text{ is } \lambda\text{-feasible.}$$

Algorithm 7 SEQSEARCH + PI : Compute the maximum λ -feasible value

Input: RS-SSP, ϵ , β

Output: Optimal policy π and return the maximum λ -feasible

```

1: Choose an initial policy  $\pi_0$  arbitrarily
2:  $i \leftarrow 1$ 
3:  $\lambda \leftarrow -1$ 
4:  $\pi_i(s) \leftarrow \underset{a \in A}{\operatorname{argmin}} [\exp(\lambda C(s, a)) \sum_{s' \in S} P(s'|s, a) V^{\pi_0}(s')]$ 
5: while  $\pi_i \neq \pi_{i-1}$  do
6:   while  $\rho((\mathbf{D}^\pi)^\lambda \mathbf{P}_{G^c}^\pi) \leq (1 - \beta)$  do
7:      $\lambda \leftarrow \lambda + \frac{\ln(1-\epsilon) - \ln(\rho((\mathbf{D}^\pi)^\lambda \mathbf{P}_{G^c}^\pi))}{\max_{s \in S, a \in A} C(s, a)}$ 
8:   end while
9:    $i \leftarrow i + 1$ 
10: Policy evaluation: obtain the value of the current policy  $\pi_i$  for every  $s \in S$  solving the system of
    equations in Equation 5.2 .
11: Policy improvement : improve the current policy by doing the following update for every  $s \in S$ :


$$\pi_i(s) \leftarrow \underset{a \in A}{\operatorname{argmin}} [\exp(\lambda C(s, a)) \sum_{s' \in S} P(s'|s, a) V^{\pi_{i-1}}(s')]$$


12: end while
13: return  $\pi_i, \lambda$ 

```

Although this solution calculates the maximum λ feasible, it fails to scale for large problems, this is because it uses a synchronous algorithm which needs to evaluate the entire state space.

In Appendix E we present other solutions for this problem and also an empirical analyses. Since in this work we are assuming to know the maximum λ -feasible for all instances of the analyzed domains, we leave the details of this study in the Appendix E.

Chapter 6

Risk Sensitive SSP as a Linear Program in the Dual Space

In the previous chapter we formally defined an RS-SSP and show how to extract optimal policies through the system of equations of the value function (Equation 5.3). This system of equations, which is a modification of the original Bellman's equations (Equation 2.13), can be written directly as an optimization problem in the primal space, where the objective function is to minimize the sum of the value function for all states.

In this chapter we contribute with a linear program formulation in the dual space for an RS-SSP. This dual linear program (DLP) can be interpreted as a probabilistic flow network problem, where the variables to be optimized are occupation measure variables, which represent the number of times an action is executed in a state. For an RS-SSP, unlike the primal space, in DLP the objective function tries to minimize the sum of the weighted occupation variables with the exponential cost of executing an action.

6.1 The Primal Linear Program for Risk Sensitive SSP

An alternative approach to solve an SSP is based on linear programming as we show in Chapter 2 (Section 2.5). Similarly, in this section, we will show how the modified Bellman equations (Equation 5.3) for RS-SSPs can be described as a linear program in the primal space.

Another way to describe the system of equations for RS-SSPs is to anchor the value for the goal states in a fixed value. Usually, the exponential value function for a goal state is chosen to be $1 \times \text{sign}(\lambda)$, and can be rewritten as follow:

$$V^*(s) = \min_{a \in A(s)} \left\{ e^{\lambda C(s,a)} P(s_g | s, a) + \sum_{s' \in S \setminus G} e^{\lambda C(s,a)} P(s' | s, a) V^*(s') \right\}, \forall s \in S. \quad (6.1)$$

6.1.1 Primal LP with Implicit Goal Formulation

The set of equations 6.1 for an RS-SSP can also be solved using the following linear program (LP) formulation in the primal space:

$$\begin{aligned} & \underset{v}{\text{maximize}} && \sum_{s \in S} v_s \\ & \text{subject to} && v_s \geq 0 \\ & && v_s \leq e^{\lambda C(s,a)} P(s_g | s, a) + \sum_{s' \in S} e^{\lambda C(s,a)} P(s' | s, a) v_{s'} \quad \forall s \in S, a \in A. \end{aligned} \quad (6.2)$$

Notice that, in the LP formulation 6.2, as we do not explicitly define the value function for goal states, we call it as a *formulation in the primal space with implicit goal*. In Equations 6.2, the variables to be optimized ($v_s, \forall s \in S$) represent the value function of each state s , so the objective function of this primal linear program is to maximize the sum of all the variables (i.e., the sum of the value function for all states $\sum_{s \in S} v_s$) under the following constraints: **the variables must be positive and have to be bounded by their cumulative exponential expected value.**

6.1.2 Primal LP with Explicit Goal Formulation

Within the previous linear program (Equations 6.2) it is possible to define constraints for the value function of the goal states, as follows:

$$\begin{aligned}
& \underset{v}{\text{maximize}} && \sum_{s \in S} v_s \\
& \text{subject to} && v_s \geq 0 \\
& && v_{s_g} = 1 \times \text{sgn}(\lambda). \\
& && v_s \leq \sum_{s' \in S} e^{\lambda C(s,a)} P(s'|s,a) v_{s'} \quad \forall s \in S \setminus G, a \in A.
\end{aligned} \tag{6.3}$$

In this formulation (Equation 6.3) we explicitly set the value for the goal states variables, s_G . Note that if we want to deal with risk prone attitudes the value of v_{s_g} will be -1 and if we deal with risk averse attitudes the value of v_{s_g} will be 1 , for all other states the value is bounded by their cumulative exponential expected value.

6.2 The Dual Linear Program for Risk Sensitive SSP

Similar to [Altman, 1999, d'Epenoux, 1963] who proposed the dual formulation for SSPs and following the conception of [Trevizan et al., 2017a] to interpret this formulation as a flow network problem, we transform the linear program of Equation 6.3 into a dual linear program. In this new formulation the variables to be optimized are the occupation measures variables in the same way as for SSP in the dual space shown in LP1. The complete transformation is showed in Appendix C.

Thus, the dual linear program after transforming the primal linear program showed by Equation 6.3 is:

$$\begin{aligned}
& \underset{x}{\text{minimize}} && \sum_{s \in S, a \in \text{App}(s)} x_{s,a} e^{\lambda C(s,a)} P(s_g|s,a) && \text{(LP5)} \\
& \text{subject to} && x_{s,a} \geq 0 && \forall s \in S, a \in \text{App}(s) && \text{(F1)} \\
& && \text{out}(s) = \sum_{a \in \text{App}(s)} x_{s,a} && \forall s \in S && \text{(F2)} \\
& && \text{in}(s) = \sum_{s' \in S, a \in \text{App}(s')} x_{s',a} e^{\lambda C(s',a)} P(s'|s,a) && \forall s \in S, a \in \text{App}(s) && \text{(F3)} \\
& && \text{out}(s) - \text{in}(s) = 0 && \forall s \in S \setminus \{G \cup s_0\} && \text{(F4)} \\
& && \text{out}(s_0) - \text{in}(s_0) = 1 && && \text{(F5)} \\
& && \sum_{s_g \in G} \text{in}(s_g) = 1 && && \text{(F6)}
\end{aligned}$$

In this formulation, called LP5, the objective function is to minimize the expected exponential accumulated cost. The constraint F2 defines expected flow exiting the state s . The constraint F3 modifies the way of the incoming flow: instead of a directed cost, the incoming flow is pondered to the exponential cost, allowing model risk. The constraint F4 is the flow conservation principle of SSP with no dead-ends,

i.e., all flows reaching s must leave s (for all states $s \in S$ that are neither the initial state s_0 , called source, nor a goal state, called sink). Constraints F5 and F6 define, respectively, the source and the sinks. The objective function captures the minimization of the total expected exponential cost to reach the goal from the initial state.

The optimal solution of LP5 can be converted into an optimal policy $\pi^* = a$, where $a \in A(s)$ is the only action such that $x_{s,a}^* \neq 0$.

To the best of our knowledge, this is the first LP formulation in a dual space of an RS-SSP with exponential utility function, interpreted as an expected exponential flow network problem.

In the next chapter we show how we can use LP5 to generate an efficient heuristic for RS-SSPs, called h_{pom}^{rs} .

Chapter 7

A Heuristic for RS-SSP: h_{pom}^{rs}

In this Chapter, we propose a new heuristic based on h_{pom} , called h_{pom}^{rs} to solve RS-SSPs that estimates the minimum exponential expected cost of reaching the goal from a state s taking the risk into account. We also make an empirical evaluation of the proposed heuristic over three domains.

7.1 The heuristic h_{pom}^{rs}

In this section we propose the *probabilistic occupation measure risk sensitive heuristic* to solve to solve RS-SSPs problems, called h_{pom}^{rs} , It is based on h_{pom} and uses the LP5 to compute the projections.

Definition 7.1.1. Projection Occupation Measure Risk Sensitive Heuristic(h_{pom}^{rs}). Given a probabilistic SAS⁺ task $\langle \mathbb{V}, A, s_0, s_\star, C \rangle$, the *projection occupation measure risk sensitive heuristic*, called $h_{pom}^{rs}(s)$, is the solution of the following dual linear program:

$$h_{pom}^{rs}(s) = \min \sum_{d,a} x_{d,a} e^{\lambda C_{d,a}} P_{d,a}^g \mid Cons, \quad (\text{LP6})$$

$\forall v \in \mathbb{V}$, where $Cons = \text{Tying}^s \cup (F1 - F6)$; Tying^s are the tying constraints over variables $x_{d,a}^{s,a}$; and constraints (G1-G6), $\forall v' \in \mathbb{V}$, are:

$$x_{d,a}^{v',s} \geq 0, \quad \forall d \in D_{v'} \cup \{g\}, a \in A(d) \quad (\text{G1})$$

$$\text{in}(d) = \sum_{\substack{d' \in D_{v'} \cup \{g\} \\ a \in A(d')}} x_{d',a} e^{\lambda C_{d',a}} P_{d',a}^{d_i} \quad \forall d \in D_{v'} \cup \{g\} \quad (\text{G2})$$

$$\text{out}(d) - \text{in}(d) = 0, \quad \forall d \in D_{v'} \setminus \{s[v']\} \quad (\text{G3})$$

$$\text{out}(s[v']) - \text{in}(s[v']) = 1 \quad (\text{G4})$$

$$\text{out}(d) = \sum_{a \in A(d)} x_{d_i,a}, \quad \forall d \in D_{v'} \quad (\text{G5})$$

$$\sum_{s_g \in G} \text{in}(s_g) = 1 \quad (\text{G6})$$

Example of h_{pom}^{rs} Computation

To illustrate the computation of heuristics h_{pom}^{rs} , consider an SSP version of the triangle tire-world domain, as depicted in Fig.7.1. This problem can be described as a probabilistic SAS⁺ task $\mathcal{T} = \langle V, A, s_0, s_\star, C \rangle$ as follows:

- $\mathbb{V} = \{v_1\}$ s.t. $D_{v \in V} = \{d_0, d_1, d_2, d_3, d_4\}$;

- $A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$ with:
 - $pre(a_1) = (d_0)$, $eff(a_1) = (d_0, d_1)$ with $Pr = \{0.2, 0.8\}$, respectively;
 - $pre(a_2) = (d_0)$, $eff(a_2) = (d_0, d_2)$ with $Pr = \{0.5, 0.5\}$, respectively;
 - $pre(a_3) = (d_1)$, $eff(a_3) = (d_1, d_2)$ with $Pr = \{0.2, 0.8\}$, respectively;
 - $pre(a_4) = (d_1)$, $eff(a_4) = (d_3, d_1)$ with $Pr = \{0.2, 0.8\}$, respectively;
 - $pre(a_5) = (d_2)$, $eff(a_5) = (d_2, d_4)$ with $Pr = \{0.2, 0.8\}$, respectively;
 - $pre(a_6) = (d_3)$, $eff(a_6) = (d_3, d_4)$ with $Pr = \{0.2, 0.8\}$, respectively;
 - $pre(a_7) = (d_3)$, $eff(a_7) = (d_3, d_5)$ with $Pr = \{0.5, 0.5\}$, respectively;
 - $pre(a_8) = (d_4)$, $eff(a_8) = (d_4, d_5)$ with $Pr = \{0.2, 0.8\}$, respectively;
- $s_0 = (d_0)$;
- $s_\star = s_g = (d_5)$; and
- $C(s, a) = 1, \forall s \in S, \forall a \in A$.

Figure 7.1: State space of a simple SSP (first instance of a Triangle Tire World Domain) described as a SAS^+ task $\mathcal{T}$, where the initial state is s_0 and the goal state is s_5 .

To compute the heuristic h_{pom}^{rs} for states s_1 and s_3 , we need to compute the projection of s_1 and s_3 onto its state variable v_1 . The projection of s_1 onto v_1 results in the SSP $\mathcal{M}^{v_1, s_0} = \langle S', A', P', C', s'_0, s'_g \rangle$ (Fig. 2), where :

- $S' = \{1, 2, 3, 4, 5, g\}$,
- $A' = \{a_3, a_4, a_5, a_8, a_g\}$,
- $C'(d, a_g) = 0$ and $C'(d, a) = C(s, a), \forall a \in A'$,
- $s'_0 = \{1\}$,
- $s'_g = \{g\}$, and
- the transition probability function P' is as shown in Fig.7.2(a).

Figure 7.2: Projections for states s_1 and s_3 , where (a) is the projection $\mathcal{M}^{v_1, s_1}$ for task $\mathcal{T}$, i.e. the projection of v_1 onto s_1 and (b) is the projection $\mathcal{M}^{v_1, s_3}$ for task $\mathcal{T}$, i.e. the projection of v_1 onto s_3 .

Analogously, the projection of s_3 onto v_1 is the SSP $\mathcal{M}^{v_1, s_3}$ depicted in Fig.3(b). The result of solving LP6 for states s_1 and s_3 with a feasible lambda ($\lambda = 1.608$) is the following: $h_{pom}^{rs}(s_1) = 2.09$ and $h_{pom}^{rs}(s_3) = 2.50$, which indicates the planner to choose the action a_1 since the expected exponential value in s_1 is lower than s_3 . However when we compute $h_{pom}(s_1) = 4.88$ and $h_{pom}(s_1) = 4.53$, the results differ to the h_{pom}^{rs} heuristic, and tells the agent to choose the action a_2 instead a_1 . The heuristic h_{pom} as well as any other heuristic based on a neutral risk attitude is interested only in minimizing the accumulated cost of reaching the goal losing information about risk that may exist.

7.2 Empirical Analysis

In this section we analyze the performance of the proposed heuristic h_{pom}^{rs} over instances of three planning domains: River, Triangle Tire World and Navigation. For that, we use our proposed h_{pom}^{rs} heuristic within RS-LRTDP (Algorithm 6) and compare it against:

- the classical Policy Iteration (PI) algorithm for RS-SSPs as a baseline, whose Bellman update is based on Equation 5.3;
- RS-LRTDP algorithm (LRTDP with Bellman update based on Equation 5.3) with heuristic h_{local}^{rs} [Freitas, 2019], that computes Equation 5.5;
- RS-LRTDP algorithm (LRTDP with Bellman update based on Equation 5.3) with the heuristic h_{pom} [Trevizan et al., 2017b].

We run 10 times the algorithms for each instance of the analysed domains and compute the average time, with a maximum of 60 minutes and 6GB of memory in an intel i7 processor at 2.7Hz. To solve the LPs we used the Gurobi7.5 solver.

The three domains analysed in this section are RS-SSPs with no dead-ends. Following, we give the description of them showing how they deal with dead-ends.

7.2.1 Planning Domains

River Domain. In this domain the agent must cross the river in two ways: (i) swimming from any point of the river bank, or (ii) going along the river bank until a bridge at $y = N_y$. However, the river flows to a waterfall (in $y = 1$). Since the domain has no dead-ends, after falling in the waterfall, the agent returns to the initial state. The initial state is in one side of the river and far from the bridge, $x_0 = 1$ and $y_0 = 1$, and the goal is in the other side of the river bank far from the bridge, $x_g = N_x$ and $y_g = 1$. Actions can be taken in any of the cardinal directions: N, S, E and W . If actions are taken on the river bank or in the bridge then transitions are deterministic to the cardinal directions; if actions are taken in the river then transitions are probabilistic and follows the chosen cardinal directions with probability $1 - p$ or follows down the river with probability p .

Triangle Tire World Domain. This is a variation of the original version, where a car can move to a different location through routes. The objective in this domain is to go from an initial state s_0 to a goal location s_g . However, in each movement, there is a probability of puncturing a tire. Since the domain has no dead-ends, states with flat-tire have always a spare-tire. The probability of puncturing a tire is much higher when the agent moves along the shortest path as show in Figure 7.1.

Navigation Domain. In this variation of the navigation domain, we consider a robot that must go from an initial location to a goal location through a grid world, the robot is able to execute four actions *move-north*, *move-south*, *move-east* and *move-west*. When applying the *move-north* action, there is a probability of *returning to the initial location* (instead of breaking the robot which would configure a dead-end). Each instance of this domain is parameterized by the total number of columns (col) and rows (row). Another parameter are min_p and max_p that define, respectively, the probability of the robot returning to the initial location when executing the *move-north* action. The probability of the robot returning to the initial in the column $j \in [1, col]$ is given by: $probability_return = (min_p) + (j - 1) \frac{max_p - min_p}{col - 1}$.

7.2.2 Analysing the performance of h_{pom}^{rs}

Tables 7.1, 7.2 and 7.3 and their corresponding plots (Figure 7.3, 7.4 and 7.5) show the results of solving several RS-SSP instances of the analyzed domains (where "-" indicates the instance was not solved due to lack of time or memory), using Algorithm 5 and Algorithm 6 with three heuristics: h_{local}^{rs} , h_{pom} and our proposed heuristic h_{pom}^{rs} . In all tests we previously computed the maximum $\lambda - feasible$ for each domain according with Appendix E. Our goal is to compute optimal extreme risk-averse policies.

All returned policies computed with heuristic h_{pom}^{rs} were optimal since they are the same policies returned with heuristic h_{pom}^{rs} which is an admissible heuristic (Theorem 3). They are also optimal extreme risk averse policies, since we used the maximum $\lambda - feasible$ (Appendix E).

Table 7.1 shows the results for the **Triangle Tire World domain**. The algorithms LRTDP + h_{local}^{rs} and LRTDP + h_{pom} performed better, in terms of convergence time, than using the baseline algorithm PI for most of the instances. Notice that LRTDP + h_{pom} outperformed h_{local}^{rs} and was able to solve problems until 4498500 states in 56 minutes approximately. This is surprising since h_{pom} does not consider the exponential utility. However, LRTDP with our proposed heuristic h_{pom}^{rs} was able to solve the same instance in about 27 minutes. Moreover, the use of h_{pom}^{rs} allowed us to solve all instances in this domain, all of them with the lowest average convergence time. We solved 6 more instances that h_{pom} was able to solved and 7 more than h_{local}^{rs} . Figure 7.3 shows how the time spent by LRTDP + h_{pom} increases fast: it spend almost the same average time (56 minutes) to solve instance 1000 than LRTDP + h_{pom}^{rs} spent to solve the largest instance in this domain (56 minutes).

Triangle Tire World					
instance	# states	PI	LRTDP + h_{local}^{rs}	LRTDP + h_{pom}	LRTDP + h_{pom}^{rs}
1	3	0.26	0.43	0.07	0.07
3	36	1.06	0.80	0.58	0.31
5	105	2.71	1.55	1.09	0.47
7	210	5.23	2.69	1.88	0.83
9	351	8.63	4.25	2.93	1.27
11	528	12.88	6.18	4.26	1.77
13	741	18.00	8.54	5.86	2.40
20	1770	42.72	19.86	13.56	5.50
50	11175	268.72	123.32	27.00	13.98
100	44850	-	493.72	33.19	23.97
500	1124250	-	1236.14	841.81	480.57
1000	4498500	-	-	3368.57	1262.08
1100	5443350	-	-	-	1552.74
1200	6478200	-	-	-	1769.49
1300	7603050	-	-	-	2256.05
1400	8817900	-	-	-	2668.50
1500	10122750	-	-	-	3266.54
1600	11517600	-	-	-	3581.40

Table 7.1: Analysing h_{pom}^{rs} : average time (secs) to compute extreme risk-averse policies in the **Triangle Tire World** domain ($\lambda=1.608$).

Figure 7.3: Analysing h_{pom}^{rs} : plots of Table 7.1.

Table 7.2 shows the results for the **River domain**. Although in this domain LRTDP + h_{pom} was much better than LRTDP + h_{local}^{rs} , LRTDP + h_{pom}^{rs} , our proposed heuristic, used about half of the time to solve the 7 largest instances that h_{pom} was able to solve, and were the only one to solve the 7 largest

instances of this domain. The h_{local}^{rs} heuristic was only able to solve instances with less than 50000 states, a performance similar to the PI baseline algorithm.

River					
instance	# states	PI	LRTDP + h_{local}^{rs}	LRTDP + h_{pom}	LRTDP + h_{pom}^{rs}
5x6	30	0.99	0.29	0.45	0.27
5x8	40	2.51	1.92	0.66	0.37
5x10	50	4.04	3.36	0.91	0.49
5x12	60	5.56	4.64	1.16	0.62
5x14	70	7.09	6.65	1.21	0.71
5x16	80	8.61	6.67	1.69	0.79
5x18	90	10.13	7.64	1.92	0.88
5x20	100	11.66	9.99	1.93	0.99
5x50	250	34.52	30.43	5.54	2.62
5x100	500	72.62	25.99	11.06	5.57
5x500	2500	377.42	45.61	31.23	13.24
5x1000	5000	758.42	61.29	42.60	23.18
5x5000	25000	-	299.03	54.48	27.31
5x10000	50000	-	712.12	97.49	55.83
5x50000	250000	-	-	508.11	265.76
5x100000	500000	-	-	1018.60	529.72
5x150000	750000	-	-	1521.62	792.915
5x200000	1000000	-	-	2026.87	1056.165
5x250000	1250000	-	-	2732.12	1319.415
5x300000	1500000	-	-	3542.62	1582.665
5x350000	1750000	-	-	-	1845.915
5x400000	2000000	-	-	-	2109.17
5x450000	2250000	-	-	-	2372.42
5x500000	2500000	-	-	-	2635.67
5x550000	2750000	-	-	-	2898.915
5x600000	3000000	-	-	-	3162.165
5x650000	3250000	-	-	-	3425.415

Table 7.2: Analysing h_{pom}^{rs} : average time (secs) to compute extreme risk-averse policies in the **River** domain ($\lambda=2.301$).

Figure 7.4: Analysing h_{pom}^{rs} : plots of Table 7.2.

For the **Navigation domain**, Table 7.3 shows that the LRTDP + h_{pom} were able to solve 3 more instances than LRTDP + h_{local}^{rs} , a worse result than in the River domain. On the other hand, we were able to solve 8 more instances than LRTDP + h_{pom}^{rs} , outperforming all other solutions.

Figure 7.5: Analysing h_{pom}^{rs} : plots of Table 7.3.

Navigation					
instance	# states	PI	LRTDP + h_{local}^{rs}	LRTDP + h_{pom}	LRTDP + h_{pom}^{rs}
6x5	30	0.78	0.16	0.18	0.08
8x5	40	1.35	0.20	0.21	0.18
10x5	50	1.92	0.42	0.47	0.21
12x5	60	2.28	0.46	0.78	0.40
14x5	70	3.63	0.87	1.09	0.47
16x5	80	3.70	1.28	1.19	0.76
18x5	90	4.88	1.69	1.47	0.83
20x5	100	6.02	2.11	1.81	0.95
50x5	250	17.23	8.30	5.45	2.77
100x5	500	35.41	18.61	11.63	5.94
500x5	2500	58.69	37.13	26.15	11.62
1000x5	5000	104.80	57.28	43.05	17.61
5000x5	25000	191.08	102.48	61.07	30.14
10000x5	50000	382.26	206.98	123.77	61.39
50000x5	250000	1915.67	1031.98	658.79	309.41
100000x5	500000	-	2062.98	1336.12	618.10
150000x5	750000	-	-	1759.71	825.41
200000x5	1000000	-	-	2372.30	1233.16
250000x5	1250000	-	-	3186.01	1440.91
300000x5	1500000	-	-	-	1698.66
350000x5	1750000	-	-	-	2056.41
400000x5	2000000	-	-	-	2264.16
450000x5	2250000	-	-	-	2671.91
500000x5	2500000	-	-	-	3179.66
550000x5	2750000	-	-	-	3487.41
5x600000	3000000	-	-	-	3162.165
5x650000	3250000	-	-	-	3425.415

Table 7.3: Analysing h_{pom}^{rs} : average time (secs) to compute extreme risk-averse policies in the **Navigation** domain ($\lambda=2.301$).

7.3 Conclusions of Part II

In this Part II, we addressed the problem of Risk Sensitive SSP (RS-SSP) and propose an efficient heuristic search solution for it. This problem was first proposed by [Patek, 2001] and it is based on the approach of [Marcus, 1997] who proposed to model risk attitude through an exponential utility function including a risk factor λ . Patek also showed the convergence of synchronous algorithms for this problem and the need for λ -feasible values. However he did not show how to find those values, neither proposed an efficient solution for an RS-SSP.

Finding the extreme λ -feasible. In [Freire, 2016] is proposed a solution (Algorithm 7) for finding the extreme λ -feasible value that needs to evaluate the entire state space which makes it fails to scale for large problems. In Appendix E we study variations of this solution that can be more efficient. However, the domains analysed in this Part II and described in Section 7.2 (Chapter 7), have a well-structured dynamics: *all instances of these benchmark domains have the same extreme risk-averse factor!* So, we

only have to solve a small instance of these domains to obtain the maximum feasible λ for all instances.

In order to construct an efficient heuristic to solve RS-SSP problems, in this Part II we also formulate an RS-SSP as a linear program in the dual space (LP5), interpreted as a network flow problem, whose relaxation generated the proposed heuristic, called h_{pom}^{rs} . The experimental results show that h_{pom}^{rs} used with a risk sensitive version of LRTDP, outperforms previous solutions in all instances of the analysed domain, being the only approach to solve the largest instances of the three domains.

Chapter 8

Overall Conclusions and Future Work

8.1 Conclusions

In this work we address how to build efficient heuristics for probabilistic planning problems with dead-ends and probabilistic planning problems with risk. We made a review of the models, algorithms and heuristics commonly used to solve them, and we proposed three new heuristics, which contribute with the improvement of the revised algorithms. The three new heuristics of this work are based on the projection occupation measure heuristic h_{pom} [Trevizan et al., 2017b].

In sum, the main contributions of this work are:

- Development of a new heuristic called h_{pom}^{pe} , which is the first one that can be used directly in probabilistic planning problems with unavoidable dead-ends under the FCP optimization criterion, while considering the probabilities as in h_{pom} .
- Development of a new heuristic called h_{ppom} , which is the first one to efficiently estimate the value function for MAXPROB problems.
- Interpreting of a linear program in the dual space for RS-SSPs as a network flow problem.
- Development of a new heuristic called h_{pom}^{rs} , which is the first heuristic developed for risk sensitive problems that considers both the probabilities and the risk factor.

8.2 Future Works

Some of the ideas of future works are:

- to explore risk attitudes in problems with dead-ends, e.g. extending the heuristic h_{pom}^{rs} to efficiently solve RS-SSP problems with dead ends;
- to consider others utility functions for RS-SSPs and propose new solutions based on the dual formulation;
- although we have strong empirical evidences about the admissibility of heuristics h_{ppom} and h_{pom}^{rs} , we intend to write the formal proofs;
- build new heuristics to solve planning problems with complex goals; and

- to propose new probabilistic planning domains with risk.
- .

Appendix

Appendix A

The Log-Sum-Exp Strategy

When considering the expected exponential utility, a main limitation is the high numerical value resulting from the exponential value function computation. Depending on the parameters of the problem, the exponential values can become so large that they can not be processed using a 64-bit floating-point variable. Even when the computation of intermediate values is possible, the variation of exponents can cause errors of precision. To solve this type of numerical imprecision we use the *LogSumExp* technique [William et al., 2001, Freire and Delgado, 2017], which transforms the exponential growth of a function into an arithmetic growth through a logarithmic function. The definition of this transformation is showed in the Appendix A.

In the following formulation the natural logarithm function was applied to all factors of the value function of a policy π for RS-SSP [Freitas, 2019]. Be the function:

$$L^\pi(s) = \begin{cases} 0 & , \text{ if } s \in G \\ \frac{1}{\lambda} \ln \left[\text{sgn}(\lambda) \exp(\lambda C(s, \pi(s))) \sum_{s' \in S} P(s'|s, \pi(s)) V^*(s') \right] & , \text{ otherwise.} \end{cases} \quad (8.1)$$

The system of equations 8.1 can be solved for all $s \notin G$ by following:

$$\begin{aligned} L^\pi(s) &= \frac{1}{\lambda} \ln \left[\exp(\lambda C(s, \pi(s))) \sum_{s' \in S} P(s'|s, \pi(s)) \text{sgn}(\lambda) V^\pi(s') \right] \\ &= C(s, \pi(s)) + \frac{1}{\lambda} \ln \left[\sum_{s' \in S} P(s'|s, \pi(s)) \text{sgn}(\lambda) V^\pi(s') \right] \\ &= C(s, \pi(s)) + \frac{1}{\lambda} \ln \left[\sum_{s' \in S} \exp(\ln[P(s'|s, \pi(s))]) \exp(\ln[\text{sgn}(\lambda) V^\pi(s')]) \right] \\ &= C(s, \pi(s)) + \frac{1}{\lambda} \ln \left[\sum_{s' \in S} \exp(\ln[P(s'|s, \pi(s))]) + \lambda L^\pi(s) \right] \end{aligned} \quad (8.2)$$

Then the *LogSumExp* strategy is applied which consists of identifying the largest term of an exponential sum; for this, we first consider two auxiliary functions $k_{s',s}^\pi$, and k_s^π defined as:

$$k_{s',s}^\pi = \ln[P(s'|s, \pi(s))] + \lambda L^\pi(s) \quad (8.3)$$

$$k_s^\pi = \max_{s' \in S} (k_{s',s}^\pi), \quad (8.4)$$

replacing $k_{s',s}^\pi$, in Equation 8.2:

$$L^\pi(s) = C(s, \pi(s)) + \frac{1}{\lambda} \ln \left[\sum_{s' \in S} \exp(k_{s',s}^\pi) \right], \quad (8.5)$$

and when k_s^π is introduced in Equation 8.5 we have:

$$\begin{aligned}
L^\pi(s) &= C(s, \pi(s)) + \frac{1}{\lambda} \ln \left[\sum_{s' \in S} \exp(k_{s',s}^\pi - k_s^\pi) \exp(k_{s',s}^\pi) \right] \\
&= C(s, \pi(s)) + \frac{1}{\lambda} \ln \left[\exp(k_{s',s}^\pi) \sum_{s' \in S} \exp(k_{s',s}^\pi - k_s^\pi) \right] \\
&= C(s, \pi(s)) + \frac{1}{\lambda} K_s^\pi + \frac{1}{\lambda} \ln \left[\sum_{s' \in S} \exp(k_{s',s}^\pi - k_s^\pi) \right]
\end{aligned} \tag{8.6}$$

Finally a value function $V^\pi(s)$ is obtained using the certainty-equivalent function $L^\pi(s)$ (defined in Equation 8.2) by:

$$V^\pi(s) = \exp(\lambda) \exp(\lambda L^\pi(s)) \tag{8.7}$$

Appendix B

Linear program in the primal and dual space

A general linear optimization problem can be described both, in primal and dual formulation. Let the primal of an LP be the following formulation:

$$\begin{aligned} & \underset{\mathbf{x}}{\text{maximize}} && \alpha^\top \mathbf{x} \\ & \text{s.t.} && \mathbf{Ax} \leq \mathbf{b} \end{aligned} \quad ; \tag{8.8}$$

then, the corresponding symmetric dual problem is

$$\begin{aligned} & \underset{\mathbf{y}}{\text{minimize}} && \mathbf{b}^\top \mathbf{y} \\ & \text{s.t.} && \mathbf{A}^\top \mathbf{y} = \mathbf{c} \quad . \\ & && \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{8.9}$$

Appendix C

Dual Linear Program for an SSP: a vector form

In this appendix we explain how an SSP formulated as a linear program can be transformed into a formulation in the dual space. We also show how to interpret this formulation as a network flow problem in the same way [Trevizan et al., 2017b] has shown for an SSP.

Vector Form: a compact notation

- $P_{s,a}^{s'}$ is read as $P(s'|s, a)$,
- $C_{s,a}$ is read as $C(s, a)$ and
- V_s is read as $V(s)$.

Assume that states in S can be enumerated such that $S \setminus G = \{1, 2, \dots, |S \setminus G|\}$ and $s_0 = 1$, then

- $\mathbf{V}$ is the column vector of $V(s)$ for every state $s \in S \setminus G$, goal state excluded.
- V_G is the terminal value for every state $s \in G$.
- $\mathbf{P}^a$ is the square matrix transition for every state $s \in S \setminus G$, goal state excluded, and each action $a \in A$ where $(\mathbf{P}^a)_{ij} = P(j|i, a)$. We assume that $\text{App}(s) = A$ for every state $s \in S$.
- $\mathbf{P}_G^a$ is the column vector for every state $s \in S$ and each action $a \in A$ that indicates the probability of reaching a goal state, i.e., $(\mathbf{P}_G^a)_i = \sum_{s' \in G} P(s'|i, a)$.

SSP as a Primal LP with Implicit Goal Formulation: vector form

In vector form, we can rewrite the primal LP for SSPs (Equation 3.7):

$$\begin{aligned} & \underset{\mathbf{V}}{\text{maximize}} && \mathbf{b}^\top \mathbf{V} \\ & \text{s.t.} && [\mathbf{I} - \mathbf{P}^a] \mathbf{V} \leq \mathbf{C}^a \quad \forall a \in A \end{aligned} \quad (8.10)$$

SSP as a Primal LP with Explicit Goal Formulation: vector form

Also we can rewrite the primal LP for SSPs (Equation 3.7) in vector form by choosing $\mathbf{b}^\top \mathbf{1} \leq b(g)$:

$$\begin{aligned} & \underset{\mathbf{V}}{\text{maximize}} && \mathbf{b}^\top \mathbf{V} - b(G)V_G \\ & \text{s.t.} && [\mathbf{I} - \mathbf{P}^a] \mathbf{V} - V_G \mathbf{P}_G^a = [\mathbf{I} - \mathbf{P}^a, -\mathbf{P}_G^a] \begin{bmatrix} \mathbf{V} \\ V_G \end{bmatrix} \leq \mathbf{C}^a \quad \forall a \in A \\ & && V_G \geq 0 \end{aligned} \quad (8.11)$$

From Primal to Dual Linear Program Formulation of an SSP

The dual linear program formulation of SSPs was proposed by D'Epenoux (1963), which solves SSPs by optimising the *policy occupation measures* that contrasts with the common primal LP formulation where the variables being optimised represent the expected cost to reach the goal.

Programs (8.10) and (8.11) can have the following dual formulation, respectively:

$$\begin{aligned} & \underset{\mathbf{Y}^a}{\text{minimize}} && \sum_{a \in A} \mathbf{C}^a{}^\top \mathbf{Y}^a \\ & \text{s.t.} && \sum_{a \in A} [\mathbf{I} - \mathbf{P}^a{}^\top] \mathbf{Y}^a = \mathbf{b} \\ & && \mathbf{Y}^a \geq \mathbf{0} \quad \forall a \in A \end{aligned} \quad (8.12)$$

and

$$\begin{aligned}
& \underset{\mathbf{Y}^a}{\text{minimize}} && \sum_{a \in A} \mathbf{C}^a{}^\top \mathbf{Y}^a \\
& \text{s.t.} && \sum_{a \in A} [\mathbf{I} - \mathbf{P}^a{}^\top] \mathbf{Y}^a = \mathbf{b} \\
& && \sum_{a \in A} \mathbf{P}_G^a{}^T \mathbf{Y}^a \leq b(G) \\
& && \mathbf{Y}^a \geq \mathbf{0} \quad \forall a \in A
\end{aligned} \tag{8.13}$$

Note that Trevizan's primal formulation chooses $b(G) = 1$, $\mathbf{b} = [1, 0, 0, \dots, 0, 0]$ (1 for the initial state and 0 for the other states). Note then, that inequation $\sum_{a \in A} \mathbf{P}_G^a{}^T \mathbf{Y}^a \leq b(G)$ can only be respected under equality .

Appendix D

Risk-Sensitive SSP: from Primal to Dual Linear Program formulation

A Risk Sensitive SSP is an SSP with an optimization criterion that takes into account a risk factor λ indicating if the agent is averse, neutral or prone to risk. A solution for a RS-SSP is based on an exponential Value function, $\forall s \in S$:

$$V(s) = \min_{a \in A(s)} \left\{ \sum_{s' \in S} e^{\lambda C(s,a)} P(s'|s, a) V(s') \right\}, \forall s \in S. \quad (8.14)$$

Again, the solution of the previous system of non-linear equations must be anchored in a fixed value for the goal states in G . Usually, for exponential Value function such a terminal value is chosen to be $1 \times \text{sign}(\lambda)$. And in this case, we can rewrite Equation 8 as:

$$V(s) = \min_{a \in A(s)} \left\{ e^{\lambda C(s,a)} P(s_g|s, a) + \sum_{s' \in S \setminus G} e^{\lambda C(s,a)} P(s'|s, a) V(s') \right\}, \forall s \in S. \quad (8.15)$$

We can also formulate a primal LP for RS-SSPs as follows:

$$\begin{aligned} & \underset{V(s)}{\text{maximize}} && \sum_{s \in S \setminus G} b(s) V(s) \\ & \text{s.t.} && V(s) \leq e^{\lambda C(s,a)} P(s_g|s, a) + \sum_{s' \in S \setminus G} e^{\lambda C(s,a)} P(s'|s, a) V(s'), \quad \forall s \in S \setminus G, a \in A \end{aligned} \quad (8.16)$$

And defining $\mathbf{D}^a = \text{diag}(\exp(\lambda \mathbf{C}^a))$, we can rewrite the LP in vector form as:

$$\begin{aligned} & \underset{\mathbf{V}}{\text{maximize}} && \mathbf{b}^\top \mathbf{V} \\ & \text{s.t.} && [\mathbf{I} - \mathbf{D}^a \mathbf{P}^a] \mathbf{V} \leq \mathbf{D}^a \mathbf{P}_G^a, \quad \forall a \in A \end{aligned} \quad (8.17)$$

And we have the following dual formulation:

$$\begin{aligned} & \underset{\mathbf{Y}^a}{\text{minimize}} && \sum_{a \in A} \mathbf{P}_G^a{}^\top \mathbf{D}^a \mathbf{Y}^a \\ & \text{s.t.} && \sum_{a \in A} [\mathbf{I} - \mathbf{P}^a{}^\top \mathbf{D}^a] \mathbf{Y}^a = \mathbf{b} \\ & && \mathbf{Y}^a \geq \mathbf{0} \quad \forall a \in A \end{aligned} \quad (8.18)$$

which is equivalent to:

$$\begin{aligned} & \underset{Y_s^a}{\text{minimize}} && \sum_{s \in S} \sum_{a \in A} e^{\lambda C_{s,a}} P(g|s, a) Y_s^a \\ & \text{s.t.} && \sum_{a \in A(s)} Y_s^a - \sum_{s' \in S \setminus G} \sum_{a \in A(s')} e^{\lambda C_{s',a}} P(s'|s, a) Y_{s'}^a = b(s) \quad \forall s \in S \setminus G \\ & && Y_s^a \geq 0 \quad \forall a \in A, \forall s \in S \setminus G \end{aligned} \quad (8.19)$$

Note: For RS-MDP one can only have an explicit goal formulation.

Given the value exponential function:

$$V(s) = \min_{a \in A(s)} \left\{ e^{\lambda C(s,a)} P(s_g|s, a) + \sum_{s' \in S \setminus G} e^{\lambda C(s,a)} P(s'|s, a) V(s') \right\}$$

it can be rewrite as:

$$\begin{aligned} & \max \sum_{s \in S} V_s \quad \text{s.t} \\ & V_s - \left\{ \sum_{s' \in S \setminus G} e^{\lambda C_{s,a}} P_{s,a}^{s'} V_{s'} \right\} \leq e^{\lambda C_{s,a}} P_{s,a}^{s_g} \quad \forall s, a \end{aligned}$$

the primal linear problem is:

$$\begin{aligned}
 & \max \quad \begin{bmatrix} b_0 & \cdots & b_n \end{bmatrix} \begin{bmatrix} V_{s_0} \\ \vdots \\ V_{s_n} \end{bmatrix} \quad s.t. \\
 & \begin{bmatrix} 1 - e^{\lambda C_{s_0,a_0}} P_{s_0,a_0}^{s_0} & -e^{\lambda C_{s_0,a_0}} P_{s_0,a_0}^{s_1} & \cdots & -e^{\lambda C_{s_0,a_0}} P_{s_0,a_0}^{s_n} \\ \vdots & \vdots & & \vdots \\ 1 - e^{\lambda C_{s_0,a_k}} P_{s_0,a_k}^{s_0} & -e^{\lambda C_{s_0,a_k}} P_{s_0,a_k}^{s_1} & \cdots & -e^{\lambda C_{s_0,a_k}} P_{s_0,a_k}^{s_n} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ -e^{\lambda C_{s_n,a_0}} P_{s_n,a_0}^{s_0} & -e^{\lambda C_{s_n,a_0}} P_{s_n,a_0}^{s_1} & \cdots & 1 - e^{\lambda C_{s_n,a_0}} P_{s_n,a_0}^{s_n} \\ \vdots & \vdots & & \vdots \\ -e^{\lambda C_{s_n,a_k}} P_{s_n,a_k}^{s_0} & -e^{\lambda C_{s_n,a_k}} P_{s_n,a_k}^{s_1} & \cdots & 1 - e^{\lambda C_{s_n,a_k}} P_{s_n,a_k}^{s_n} \end{bmatrix} \begin{bmatrix} V_{s_0} \\ \vdots \\ V_{s_n} \end{bmatrix} \leq \begin{bmatrix} e^{\lambda C_{s_0,a_0}} P_{s_0,a_0}^{s_g} \\ \vdots \\ e^{\lambda C_{s_0,a_k}} P_{s_0,a_k}^{s_g} \\ \vdots \\ \vdots \\ e^{\lambda C_{s_n,a_0}} P_{s_n,a_0}^{s_g} \\ \vdots \\ e^{\lambda C_{s_n,a_k}} P_{s_n,a_k}^{s_g} \end{bmatrix}
 \end{aligned}$$

the corresponding dual formulation

$$\begin{aligned}
 & \min \quad c^T x \quad s.t. \\
 & A^T x \geq b, x \geq 0
 \end{aligned}$$

is

$$\min \left[e^{\lambda C_{s_0,a_0}} P_{s_0,a_0}^{s_g} \quad \cdots \quad e^{\lambda C_{s_0,a_k}} P_{s_0,a_k}^{s_g} \quad \cdots \quad e^{\lambda C_{s_n,a_0}} P_{s_n,a_0}^{s_g} \quad \cdots \quad e^{\lambda C_{s_n,a_k}} P_{s_n,a_k}^{s_g} \right] \begin{bmatrix} x_{s_0,a_0} \\ \vdots \\ x_{s_0,a_k} \\ \vdots \\ \vdots \\ x_{s_n,a_0} \\ \vdots \\ x_{s_n,a_k} \end{bmatrix} \quad s.t.$$

$$\begin{bmatrix} 1 - e^{\lambda C_{s_0,a_0}} P_{s_0,a_0}^{s_0} & \cdots & 1 - e^{\lambda C_{s_0,a_k}} P_{s_0,a_k}^{s_0} & \cdots & -e^{\lambda C_{s_n,a_0}} P_{s_n,a_0}^{s_0} & \cdots & -e^{\lambda C_{s_n,a_k}} P_{s_n,a_k}^{s_0} \\ -e^{\lambda C_{s_0,a_0}} P_{s_0,a_0}^{s_1} & \cdots & -e^{\lambda C_{s_0,a_k}} P_{s_0,a_k}^{s_1} & \cdots & -e^{\lambda C_{s_n,a_0}} P_{s_n,a_0}^{s_1} & \cdots & -e^{\lambda C_{s_n,a_k}} P_{s_n,a_k}^{s_1} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ -e^{\lambda C_{s_0,a_0}} P_{s_0,a_0}^{s_n} & \cdots & -e^{\lambda C_{s_0,a_k}} P_{s_0,a_k}^{s_n} & \cdots & 1 - e^{\lambda C_{s_n,a_0}} P_{s_n,a_0}^{s_n} & \cdots & 1 - e^{\lambda C_{s_n,a_k}} P_{s_n,a_k}^{s_n} \end{bmatrix} \begin{bmatrix} x_{s_0,a_0} \\ \vdots \\ x_{s_0,a_k} \\ \vdots \\ \vdots \\ x_{s_n,a_k} \\ \vdots \\ x_{s_n,a_n} \end{bmatrix} \geq \begin{bmatrix} b_0 \\ \vdots \\ b_n \end{bmatrix}$$

$$\sum_{a \in A(s_i)} x_{s_i,a} - \sum_{s' \in S \setminus G} \sum_{a \in A(s')} x_{s',a} e^{\lambda C_{s_i,a}} P_{s',a}^{s_i} \geq b_i \quad s.t. \quad i \in \{0, \dots, n\}$$

it is equivalent to:

$$\min \sum_{s \in S} \sum_{a \in A} x_{s,a} e^{\lambda C_{s,a}} P_{s,a}^g \quad s.t.$$

$$x_{s,a} \geq 0$$

$$out(s_i) = \sum_{a \in A(s_i)} x_{s_i,a}$$

$$in(s_i) = \sum_{s' \in S \setminus G} \sum_{a \in A(s')} x_{s',a} e^{\lambda C_{s',a}} P_{s',a}^{s_i}$$

$$out(s_i) - in(s_i) \geq b_i$$

Appendix E

Finding λ -feasible for extreme risk policies

In this appendix we show different algorithms to find a extreme feasible λ value to be used in RS-SSP problems. Since this computation is not the main focus of this dissertation, but useful to replicate our experiments and results, we added it as an appendix. The algorithms analysed here are:

- Sequential Search with Policy Iteration (SEQSEARCH + PI). This solution (Algorithm 7) was proposed by [Freire, 2016] and is explained in more details in Section 5.3.3 of Chapter 5, and served as inspiration for the other solutions we propose and analyse.
- Sequential Search with Dual Linear Program (SEQSEARCH + DLP) (Section 8.2.1)
- Sequential Search with LRTDP (seqsearch + LRTDP) (Section 9)
- Binary Search with Dual Linear Program (BINSEARCH + DLP) (Section 8.2.5)

8.2.1 Sequential Search with Dual Linear Program (seqsearch + DLP)

The Algorithm 8 is a variation of Algorithm 7 (SEQSEARCH + PI) where, instead of PI we call a solver to find a solution for the dual formulation of an RS-SSP (LP5 of Section 6.2). It calls the Dual Linear Program formulation of a RS-SSP proposed in Chapter 5, using the Gurobi solver to return a solution or fail. It is used to generate a new policy in each iteration with a new λ . The increase of λ in each iteration is calculated in the same way as in Algorithm 7.

Algorithm 8 SEQSEARCH + DLP : Compute the maximum λ -feasible value

Input: RS-SSP, ϵ , β

Output: Optimal policy π and return the maximum λ -feasible

```
1: Choose an initial policy  $\pi_0$  arbitrarily
2:  $i \leftarrow 1$ 
3:  $\lambda \leftarrow -1$ 
4:  $R \leftarrow \text{DLP\_RS\_SSP}(\text{RS-SSP}, \lambda)$ 
5:  $\pi_i \leftarrow R.\text{policy}$ 
6: while  $\pi_i \neq \pi_{i-1}$  do
7:   while  $\rho((\mathbf{D}^\pi)^\lambda \mathbf{P}_{\mathcal{G}^c}^\pi) \leq (1 - \beta)$  do
8:      $\lambda \leftarrow \lambda + \frac{\ln(1-\epsilon) - \ln(\rho((\mathbf{D}^\pi)^\lambda \mathbf{P}_{\mathcal{G}^c}^\pi))}{\max_{s \in S, a \in A} C(s, a)}$ 
9:   end while
10:   $i \leftarrow i + 1$ 
11:   $R \leftarrow \text{DLP\_RS\_SSP}(\text{RS-SSP}, \lambda)$ 
12:   $\pi_i \leftarrow R.\text{policy}$ 
13: end while
14: return  $\pi_i, \lambda$ 
```

8.2.2 Sequential Search with LRTDP (seqsearch + LRTDP)

The Algorithm 9 is also a variation of Algorithm 7 (SEQSEARCH + PI) where, instead of PI we use the LRTDP to compute a policy π_i in each iteration (line 11). To compute the spectral radius (line 7) we use the matrix $\mathbf{P}_{\mathcal{G}^c}^\pi$ where each cell $(\mathbf{P}_{ij}^\pi)$ takes the value of the transition probability from state i to state j when following policy π , i.e., $(\mathbf{P}_{ij}^\pi = P(j|\pi(i), i))$.

Algorithm 9 SEQSEARCH + LRTDP: Compute the maximum λ -feasible value

Input: RS-SSP, ϵ , β

Output: Optimal policy π and return the maximum λ -feasible

```

1: Choose an initial policy  $\pi_0$  arbitrarily
2:  $i \leftarrow 1$ 
3:  $\lambda \leftarrow -1$ 
4:  $R \leftarrow \text{DLP\_RS\_SSP}(\text{RS-SSP}, \lambda)$ 
5:  $\pi_i \leftarrow R.\text{policy}$ 
6: while  $\pi_i \neq \pi_{i-1}$  do
7:   while  $\rho((\mathbf{D}^\pi)^\lambda \mathbf{P}_{\mathcal{G}^c}^\pi) \leq (1 - \beta)$  do
8:      $\lambda \leftarrow \lambda + \frac{\ln(1-\epsilon) - \ln(\rho((\mathbf{D}^\pi)^\lambda \mathbf{P}_{\mathcal{G}^c}^\pi))}{\max_{s \in S, a \in A} C(s, a)}$ 
9:   end while
10:   $i \leftarrow i + 1$ 
11:   $R \leftarrow \text{RS-LRTDP}(\text{RS-SSP}, \lambda)$ 
12:   $\pi_i \leftarrow R.\text{policy}$ 
13: end while
14: return  $\pi_i, \lambda$ 

```

Algorithm 10 BINSEARCH + DLP

Input: RS-SSP, error

Output: return the maximum λ -feasible

```

1:  $bottom \leftarrow 0$ 
2:  $top \leftarrow error$ 
3:  $R \leftarrow \text{DLP\_RS\_SSP}(\text{RS-SSP}, top)$ 
4: while  $R.\text{isFeasible}$  do
5:    $top \leftarrow top * 2$ 
6:    $bottom \leftarrow top$ 
7:    $R \leftarrow \text{DLP\_RS\_SSP}(\text{SSP}, top)$ 
8: end while
9: while  $top - bottom > error$  do
10:   $f \leftarrow (top - bottom)/2$ 
11:   $R \leftarrow \text{DLP\_RS\_SSP}(\text{SSP}, bottom)$ 
12:  if  $R.\text{isFeasible}$  then
13:     $bottom \leftarrow f$ 
14:  else
15:     $top \leftarrow f$ 
16:  end if
17: end while
18:  $\lambda \leftarrow bottom$ 
19:  $\pi \leftarrow R.\text{policy}$ 
20: return  $\pi, \lambda$ 

```

8.2.3 Binary Search with Dual Linear Program (binsearch + DLP)

This is a variation of Algorithm 8, where instead of a sequential search we use a binary search. I.e., Algorithm 10 returns an approximate extreme λ -feasible using a binary search method. It also calls the Dual Linear Program formulation of a RS-SSP proposed in Chapter 5, using the Gurobi solver to return a solution or fail. The *top* and *bottom* variables define the search range in the binary search. From line

4 to line 8, we look for the first λ value that is unfeasible, i.e., for which the DLP of an RS-SSP has no solution. This value is then assigned to the variable *top*. Then a binary search is applied in this reduced range (from line 9 to line 17) until the range interval be smaller than *error*. Finally, the λ -feasible value is returned in the variable *botton*.

8.2.4 A Comparative analyzes of the Binary and Sequential Search for the extreme λ -feasible

We compare the four algorithms to find the extreme λ -feasible in instances of three planning domains: Triangle Tire World, River and Navigation but we changed the original benchmark with a random dynamics. This is because, as we explain in the next section, the benchmark domains described in PPDDL, have a well defined structure and therefore, all instances of a domain would have the same extreme feasible λ . The results are shown in Tables 8.3, 8.2 and 8.1. Note that "-" indicates the instances that were not solved due to numerical precision error.

The experiments show that the use of the dual linear formulation (BINSEARCH + DLP and SEQSEARCH + DLP) are faster than SEQSEARCH + PI and SEQSEARCH + LRTDP in small instances. However they can not scale up efficiently due to the problem of numerical imprecision. On the other hand, SEQSEARCH + PI and SEQSEARCH + LRTDP can use the *LogSumExp* strategy (Appendix A) which allow them to solve larger instances.

Instance	states	BINSEARCH		SEQSEARCH					
		DLP		PI		DLP		LRTDP	
		time	error	time	error	time	error	time	error
2	15	0.12	0.008	0.07	0	0.01	0	0.01	0
4	66	0.33	0.005	0.24	0	0.07	0	0.16	0
6	153	0.79	0.006	0.96	0	0.12	0	0.54	0
8	276	1.62	0.003	1.94	0	0.25	0	0.61	0
10	435	2.83	0.008	4.05	0	0.32	0	0.79	0
12	630	5.10	0.008	7.74	0	0.98	0	1.26	0
14	861	7.95	0.004	13.62	0	1.75	0	2.61	0
16	1128	12.45	0.008	21.64	0	2.48	0	3.98	0
18	1431	17.42	0.006	35.42	0	3.37	0	6.40	0
20	1770	25.90	0.005	57.64	0	5.21	0	7.35	0
22	2145	31.25	0.009	67.13	0	6.12	0	8.91	0
24	2556	-	-	85.03	0	-	-	10.45	0
26	3003	-	-	115.94	0	-	-	12.95	0
28	3486	-	-	134.54	0	-	-	15.90	0
30	4005	-	-	180.56	0	-	-	18.42	0
32	4560	-	-	237.28	0	-	-	22.03	0
34	5151	-	-	289.07	0	-	-	28.61	0
36	5778	-	-	324.55	0	-	-	31.46	0
38	6441	-	-	375.86	0	-	-	33.45	0
40	7140	-	-	434.64	0	-	-	40.26	0

Table 8.1: Extreme λ -feasible in the Triangle Tire World Domain: random transitions. Time (s) and error (%) for 20 instances computed with randomly transition functions, the error is calculated in relation with the theoretical extreme λ -feasible

Instance	states	BINSEARCH		SEQSEARCH					
		DLP		PI		DLP		LRTDP	
		time	error	time	error	time	error	time	error
5x6	30	0.33	0.008	113.56	0	0.06	0	0.09	0
5x8	40	0.48	0.005	159.16	0	0.16	0	1.65	0
5x10	50	0.69	0.006	237.60	0	0.38	0	2.15	0
5x12	60	0.82	0.003	360.48	0	0.72	0	3.75	0
5x14	70	1.05	0.032	488.07	0	1.35	0	4.91	0
5x16	80	1.13	0.094	591.05	0	1.76	0	6.04	0
5x18	90	1.29	0.054	643.85	0	2.09	0	7.49	0
5x20	100	1.45	0.008	780.96	0	2.98	0	8.83	0
5x22	110	2.36	0.091	862.35	0	4.09	0	9.71	0
5x24	120	3.78	0.061	978.06	0	5.26	0	11.48	0
5x26	130	5.90	0.018	1062.45	0	6.71	0	15.08	0
5x28	140	7.94	0.045	1196.45	0	8.49	0	19.67	0
5x30	150	-	-	1325.76	0	-	0	28.19	0
5x32	160	-	-	1483.65	0	-	0	34.94	0
5x34	170	-	-	1605.45	0	-	0	41.08	0
5x36	180	-	-	1763.65	0	-	0	47.98	0
5x38	190	-	-	1912.65	0	-	0	55.19	0
5x40	200	-	-	2506.45	0	-	0	61.98	0

Table 8.2: Extreme λ -feasible in the River Domain: random transitions. Time (s) and error (%) for 20 instances computed with randomly transition functions, the error is calculated in relation with the theoretical extreme λ -feasible

Instance	states	BINSEARCH		SEQSEARCH					
		DLP		PI		DLP		LRTDP	
		time	error	time	error	time	error	time	error
6x5	30	0.27	0.003	38.45	0	0.05	0	0.13	0
8x5	40	0.34	0.005	56.83	0	0.09	0	0.34	0
10x5	50	0.54	0.033	76.14	0	1.13	0	0.79	0
12x5	60	0.78	0.098	121.36	0	1.26	0	1.38	0
14x5	70	1.17	0.044	172.60	0	1.47	0	1.90	0
16x5	80	1.28	0.003	214.64	0	1.65	0	2.76	0
18x5	90	1.37	0.009	268.48	0	2.46	0	3.07	0
20x5	100	1.59	0.007	315.47	0	3.01	0	3.58	0
22x5	110	2.56	0.011	375.14	0	3.45	0	4.87	0
24x5	120	3.19	0.010	425.86	0	3.95	0	5.98	0
26x5	130	4.89	0.011	493.45	0	4.78	0	6.04	0
28x5	140	5.64	0.004	519.65	0	5.21	0	7.56	0
30x5	150	6.98	0.020	594.82	0	6.09	0	8.90	0
32x5	160	7.95	0.030	648.15	0	6.94	0	11.73	0
34x5	170	11.65	0.110	705.65	0	7.14	0	13.95	0
36x5	180	-	-	779.08	0	-	-	16.48	0
38x5	190	-	-	816.54	0	-	-	19.24	0
40x5	200	-	-	892.61	0	-	-	21.08	0
42x5	210	-	-	1022.76	0	-	-	23.89	0
44x5	220	-	-	1134.02	0	-	-	27.46	0

Table 8.3: Extreme λ -feasible in the Navigation Domain: random transitions. Time (secs) and error (%) for 20 instances computed with randomly transition functions, the error is calculated in relation with the theoretical extreme λ -feasible

8.2.5 Maximum λ -feasible for the domains tested in this work

The domains described in Section 7.2 of Chapter 7, River, Triangle Tire World and Navigation, instead of a random state transition, like the ones analysed in this Appendix, they have a well-structured dynamics. So, when we run these algorithms to find the maximum feasible λ feasible, ***we found the same value for all instances***. Table 8.4 show the maximum feasible λ values used in the experiments of Section 7.2 as an input for the tested algorithms and heuristics. What we have learned is that these benchmark domains have the same extreme risk-averse factor for all their instances. So, we can use any of the algorithms from this appendix to find the maximum λ feasible, by solving only a small instance of these domains, without worrying with efficiency.

Domain	Extreme λ -feasible
Triangle-tire World	1.608
River	2.301
Navigation	2.301

Table 8.4: Maximum feasible λ feasible for Triangle-Tire World, River and Navigation Domain

Bibliography

- [Aberdeen et al., 2004] Aberdeen, D., Thiébaux, S., and Zhang, L. (2004). Decision-theoretic military operations planning. In *ICAPS*, pages 402–412. AAAI.
- [Altman, 1999] Altman, E. (1999). *Constrained Markov Decision Processes*. Chapman & Hall/CRC.
- [Barto et al., 1995] Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138.
- [Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.
- [Bertsekas and Tsitsiklis, 1991] Bertsekas, D. P. and Tsitsiklis, J. N. (1991). An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3):580–595.
- [Blum and Furst, 1997] Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300.
- [Bonet and Geffner, 2001] Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33.
- [Bonet and Geffner, 2003a] Bonet, B. and Geffner, H. (2003a). Faster heuristic search algorithms for planning with uncertainty and full feedback. *IJCAI International Joint Conference on Artificial Intelligence*, pages 1233–1238.
- [Bonet and Geffner, 2003b] Bonet, B. and Geffner, H. (2003b). Labeled RTDP : Improving the Convergence of Real-Time Dynamic Programming. *ICAPS 2003*, (Section 2):12–21.
- [Bonet and Geffner, 2005] Bonet, B. and Geffner, H. (2005). MGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, 24:933–944.
- [Bryce and Kim, 2006] Bryce, D. and Kim, S. (2006). Planning for gene regulatory network intervention. In *2006 IEEE/NLM Life Science Systems and Applications Workshop*, pages 1–2.
- [Chung and Sobel, 1987] Chung, K.-J. and Sobel, M. J. (1987). Discounted MDP’s: distribution functions and exponential utility maximization. *SIAM J. Control Optim.*, 25:49–62.
- [Crites and Barto, 1995] Crites, R. H. and Barto, A. G. (1995). Improving elevator performance using reinforcement learning. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS’95, pages 1017–1023, Cambridge, MA, USA. MIT Press.
- [Denardo and Rothblum, 1979] Denardo, E. V. and Rothblum, U. G. (1979). Optimal stopping, exponential utility, and linear programming. *Mathematical Programming*, 16(1):228–244.
- [d’Epenoux, 1963] d’Epenoux, F. (1963). A probabilistic production and inventory problem. *Management Science*, 10(1):98–108.

- [Edelkamp, 2000] Edelkamp, S. (2000). Heuristic search planning with BDDs. ECAI-Workshop: PuK.
- [Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI’71, pages 608–620, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Filar et al., 1989] Filar, J. A., Kallenberg, L. C. M., and Lee, H.-M. (1989). Variance-penalized Markov decision processes. *Mathematics of Operations Research*, 14(1):147–161.
- [Filar et al., 1995] Filar, J. A., Krass, D., Ross, K. W., and Ross, K. W. (1995). Percentile performance criteria for limiting average Markov decision processes. *IEEE Transactions on Automatic Control*, 40(1):2–10.
- [Freire, 2016] Freire, V. (2016). The role of discount factor in Risk Sensitive Markov Decision Processes. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 480–485.
- [Freire and Delgado, 2016] Freire, V. and Delgado, K. V. (2016). Extreme Risk Averse Policy for Goal-Directed Risk-Sensitive Markov Decision Process. *Bracis*.
- [Freire and Delgado, 2017] Freire, V. and Delgado, K. V. (2017). GUBS: a utility-based semantic for Goal-Directed Markov Decision Processes. In *Sixteenth International Conference on Autonomous Agents & Multiagent Systems*, pages 741–749.
- [Freitas, 2019] Freitas, E. (2019). *Planejamento probabilístico sensível a risco com ILAO e função utilidade exponencial*. PhD thesis, UNIVERSIDADE DE SÃO PAULO.
- [García and Fernández, 2015] García, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.*, 16(1):1437–1480.
- [Ghallab et al., 1998] Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D., Sun, Y., and Weld, D. (1998). Pddl - the planning domain definition language.
- [Hansen and Zilberstein, 2001] Hansen, E. A. and Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. 129:35–62.
- [Haslum and Geffner, 2000] Haslum, P. and Geffner, H. (2000). Admissible Heuristics for Optimal Planning. *Aips*, pages 140–149.
- [Helmert, 2006] Helmert, M. (2006). The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246.
- [Helmert, 2009] Helmert, M. (2009). Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5):503 – 535. Advances in Automated Plan Generation.
- [Hoffmann, 2001] Hoffmann, J. (2001). FF: The fast-forward planning system. *AI magazine*, 22:57–62.
- [Hou et al., 2014] Hou, P., Yeoh, W., and Varakantham, P. (2014). Revisiting risk-sensitive MDPs: New algorithms and results. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.
- [Hou et al., 2016] Hou, P., Yeoh, W., and Varakantham, P. (2016). Solving risk-sensitive POMDPs with and without cost observations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 3138–3144.

- [Howard and Matheson, 1972] Howard, R. A. and Matheson, J. E. (1972). Risk-sensitive Markov decision processes. *Management science*, 18(7):356–369.
- [Huang and Zhang, 2012] Huang, R. and Zhang, W. (2012). Sas+ planning as satisfiability. *Journal of Artificial Intelligence Research*, 43:1–150.
- [Jaquette, 1976] Jaquette, S. C. (1976). A utility criterion for Markov decision processes. *Management Science*, 23(1):43–49.
- [Keeney et al., 1979] Keeney, R., Raiffa, H., and W. Rajala, D. (1979). Decisions with Multiple Objectives: Preferences and Value Trade-Offs. *Systems, Man and Cybernetics, IEEE Transactions on*, 9:403.
- [Keyder and Geffner, 2008] Keyder, E. and Geffner, H. (2008). The HMDPP planner for planning with probabilities. *Planning Competition at ICAPS*.
- [Kolobov et al., 2012] Kolobov, A., Mausam, and Weld, D. S. (2012). A theory of goal-oriented mdps with dead ends. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, UAI’12, pages 438–447, Arlington, Virginia, United States. AUAI Press.
- [Marcus, 1997] Marcus, S. (1997). Risk sensitive Markov decision processes. *Systems and Control in the 21st Century*.
- [Mausam et al., 2005] Mausam, Benazera, E., Brafman, R., Meuleau, N., and Hansen, E. A. (2005). Planning with continuous resources in stochastic domains. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI’05, pages 1244–1251, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Mausam and Kolobov, 2012] Mausam and Kolobov, A. (2012). *Planning with Markov Decision Processes: An AI Perspective*, volume 6.
- [Mihatsch and Neuneier, 2002] Mihatsch, O. and Neuneier, R. (2002). Risk-sensitive reinforcement learning. *Machine Learning*, 49(2):267–290.
- [Patek, 2001] Patek, S. D. (2001). On terminating Markov decision processes with a risk-averse objective function. *Automatica*, 37(9):1379–1386.
- [Popyack, 2009] Popyack, J. L. (2009). Blackjack-playing agents in an advanced ai course. *SIGCSE Bull.*, 41(3):208–212.
- [Puterman, 1994a] Puterman, M. L. (1994a). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.
- [Puterman, 1994b] Puterman, M. L. (1994b). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.
- [Rojers et al., 2013] Roijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48(11):67–113.
- [Rothblum, 1984] Rothblum, U. G. (1984). Multiplicative Markov decision chains. *Mathematics of Operations Research*, 9(1):6–24.
- [Russell and Norvig, 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.
- [Shen et al., 2014] Shen, Y., Tobia, M. J., Sommer, T., and Obermayer, K. (2014). Risk-sensitive reinforcement learning. *Neural computation*, 26(7):1298–1328.

- [Simão, 2017] Simão, T. D. (2017). Planejamento Probabilístico com Becos sem Saida.
- [Sobel, 1982] Sobel, M. J. (1982). The variance of discounted Markov decision processes. *Journal of Applied Probability*, 19(4):794–802.
- [Steinmetz et al., 2016] Steinmetz, M., Hoffmann, J., and Buffet, O. (2016). Goal probability analysis in MDP probabilistic planning: Exploring and enhancing the state of the art. *Journal of Artificial Intelligence Research*.
- [Stone et al., 2005] Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188.
- [Teichteil-Königsbuch, 2012] Teichteil-Königsbuch, F. (2012). Stochastic Safest and Shortest Path Problems. *Association for the Advancement of Artificial Intelligence*, pages 1825–1831.
- [Tisovec et al., 2015] Tisovec, F. A., de Barros, L. N., Delgado, K. V., da Silva, C. F., and Hashimoto, R. F. (2015). Robust intervention on genetic regulatory networks using symbolic dynamic programming. In *Proceedings of the IJCAI Workshop Advances in Bioinformatics and Artificial Intelligence: bridging the Gaps*.
- [Trevizan et al., 2017a] Trevizan, F., Teichteil-Königsbuch, F., and Thiébaux, S. (2017a). Efficient Solutions for Stochastic Shortest Path Problems with Dead Ends.
- [Trevizan et al., 2017b] Trevizan, F., Thiébaux, S., and Haslum, P. (2017b). Occupation Measure Heuristics for Probabilistic Planning Background : SSPs. *International Conference on Automated Planning and Scheduling*, (Icaps):306–315.
- [William et al., 2001] William, N., Ross, D., and Lu, S. (, 2001,). Non-linear Optimization System And Method For Wire Length And Delay Optimization For An Automatic Electric Circuit Placer. (US 6301693 B1).
- [Yu et al., 1998] Yu, S. X., Lin, Y., and Yan, P. (1998). Optimization models for the first arrival target distribution function in discrete time. *Journal of Mathematical Analysis and Applications*, 225(1):193 – 223.