WASHINGTON UNIVERSITY IN ST.LOUIS

School of Engineering & Applied Science
Department of Computer Science and Engineering

Dissertation Examination Committee:
William Yeoh, Chair
Chien-Ju Ho
Erez Karpas
Alvitta Ottley
Yevgeniy Vorobeychik

Stochastic Goal Recognition Design

by
Christabel Wayllace

A dissertation presented to
The Graduate School
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

April 2021
St. Louis, Missouri

Table of Contents

List of Figures

List of Tables

Acknowledgments

This thesis would not be possible without the support, guide, and encouragement of many important people in my life.

First, I want to thank William Yeoh for his constant support and guidance. He is not only a great and generous advisor but also the best human being I know. William, it has been a privilege to be in your lab, and I can only pay you back following your example (as much as possible) and "aiming higher," as you always say to me. Thank you for changing my life in so many aspects and making this the best time of my life!

Next, I want to acknowledge my collaborators, from whom I learned a lot. Thanks to Ping Hou and Son Tran. Writing this document, I remembered every meeting and discussion we had to figure out all intricacies of S-GRD (fun times!). To Sarah Keren, who captivated me with her first paper and continually amazed me with her accomplishments. To Avigdor Gal (Avi), my mentor for the last three versions of our paper. Without Avi's perseverance and countless hours of work, Harry Potter wouldn't have entered into the S-GRD world! To Erez Karpas, who writes beautiful and straightforward math, thank you for your willingness to help, even if that means working late or very early in the morning! To Shlomo Silverstein, who, despite his busy schedule, provided valuable feedback to my work. To Alvitta Ottley for being a role model in all activities we shared. It was fun to work on the simulator and

the visualization projects. Thanks to her, I discovered that deadlines for data visualization projects are therapeutic instead of stressful. To Sunwoo Ha (Jennifer) for working so hard and so well. To Yuchen Han and Jiaming Hu (James) for teaching me how to be a better mentor. Special thanks to James, who said the correct words that made me decide to go to academia. To Shayan Monadjemi, with whom it was always pleasant discussing all sorts of topics. To Khoi Hoang, it took us a while to work well together, but we did it! It is surprising how working late hours to meet a deadline brings people together. It was a pleasure to suffer, get disappointed by the reviews, and finally celebrate with all of you!

I want to mention my fellow researchers, with whom I shared the ups and downs of the Ph.D. life. Sandhya Saisubramanian, I am grateful to count you as a friend; there is always something to learn from you. Ishani Chatterjee, talking to you is always refreshing; robots, papers, art, and life are not usually part of the same conversation, except for ours. Stylianos (Stelios) Vasileiou, thanks for being my pal! I am so glad we took the time to share our fears and interests; we have to get those papers out! Athena M.Tabakhi, we indeed behaved like sisters: sharing, fighting, and sharing again. Thank you for your friendship and kindness. Gan Xu and Ashwin Kumar: we had few but fun interactions, either savoring wine in Napa or looking for a quality procrastination time.

I reserve the final lines for my family. I will not be here without their constant support. My sister brought me to the U.S. and pushed or pulled me whenever I needed to move to the next step; thank you, Ale! My mother always supported me and, through her example, taught me to never give up. My father invariably cheers me up and blindly believes I can reach any goal I want. Thank you both for your unconditional love! I thank my aunts and uncles, whom I consider second moms and dads, who were discretely asking (for a while now) if my thesis was near completion. Thank you all for encouraging me to keep going.

Last but not least, I want to acknowledge my husband's support. Oscar: thank you for being my life partner and for letting me fly. I know how much you gave up when you decided to come with me. I hope this is a worthy journey for both of us because it is just starting :) I love you!

Christabel Wayllace

Washington University in Saint Louis

April 2021

Dedicated to my family.

ABSTRACT OF THE DISSERTATION

Stochastic Goal Recognition Design

by

Christabel Wayllace

Doctor of Philosophy in Computer Science

Washington University in St. Louis, 2021

Professor William Yeoh, Chair

Goal Recognition Design (GRD) is the problem of finding the least amount of environment modifications to force an acting agent to reveal its goal as early as possible. Figuring out an agent's goal by observing its behavior is a problem studied in Psychology, Economics, and Artificial Intelligence, where it is known as *goal recognition*. Contrary to most common approaches where the focus is on finding faster algorithms to detect the goal, GRD takes an offline approach and focuses on environment design to facilitate goal recognition. This thesis investigates GRD problems when action outcomes are stochastic, which is the case of most physical world interactions. I propose the Stochastic GRD (S-GRD) problem and study its specific characteristics, challenges, and limitations. Under this umbrella, we analyze partially-observable and suboptimal cases and provide a novel way to redesign the environment for partially-observable settings. This thesis presents the problem formulation and novel algorithms to solve the problem. Additionally, empirical evaluations show that S-GRD helps reduce the complexity of a goal recognition problem in all cases.

Chapter 1

Introduction

"Our goals can only be reached through a vehicle of a plan,

in which we must fervently believe,

and upon which we must vigorously act.

There is no other route to success."

– Pablo Picasso

Our ability to recognize other people's plans and goals relies on the assumption that most human behavior is goal-oriented. It enables us to understand other people's motivations and expedites human communication. Nowadays, our interactions are not limited to humans but also artificial agents. As human-machine interaction and automated systems' intelligence continually grow, so does the need to understand each other's objectives. Therefore, researchers aim to provide artificial agents with goal recognizing capabilities (Sukthankar et al., 2014). Research in goal recognition studies the problem of determining an agent's goal by observing

its behavior. Since it is desirable to recognize the goal before the agent reaches it, most research focuses on finding fast online algorithms.

One characteristic that delays the recognition is observing ambiguous behavior. Consider the Charade[1] game, where the actor collaborates with the observer by making distinctive actions to help with the goal recognition. The more ambiguous the actions, the more the amount of time that is required to guess. Consequently, a natural concern is to know if it is possible to induce a non-ambiguous behavior even when the actor is indifferent or unaware of the observer.

This thesis studies how the design of a stochastic environment can minimize the ambiguous policies of an acting agent to facilitate goal recognition. It is a more general case of Goal Recognition Design (GRD) (Keren, A. Gal, et al., 2014) that assumes deterministic action outcomes. We propose the Stochastic Goal Recognition Design (S-GRD) framework, which presumes a keyhole goal recognition and stochastic agent action outcomes. Initially, we consider the problem for optimal actors and observers with full observability; later, these assumptions are relaxed to model more realistic scenarios.

Our specific contributions consist of formalizing S-GRD problems, providing algorithms to solve them, and a novel type of modifications for partially-observable settings. Given an original stochastic environment, a set of possible goals, and an actor behavior, our algorithms find the minimal set of modifications that minimize the non-distinctive (ambiguous) policies and therefore facilitate goal recognition. We empirically evaluate the algorithms and the usefulness of the approach.

[1]Charade is a word guessing game. It requires the actors to mime their hints without using any spoken words.

1.1 Stochastic Goal Recognition Design in Context

Plan and goal recognition problems aim to identify an agent's plan or goal from observing its behavior. Researchers have made significant progress within the last decade through synergistic integrations of techniques ranging from logic (Hobbs et al., 1988; Kautz and J. F. Allen, 1986) and probability (Charniak and Robert Prescott Goldman, 1991; E. Y. Ha et al., 2011) to natural language processing (Geib and Steedman, 2007; Vilain, 1990) and planning (Ramírez and Geffner, 2009; Ramírez and Geffner, 2010; Ramírez and Geffner, 2011). Plan and goal recognition problems have been used to model applications in a wide range of domains such as software personal assistants and robots that anticipate the needs of the humans (Kelley et al., 2012; Oh et al., 2010; Oh et al., 2011a; Oh et al., 2011b; Tavakkoli et al., 2007); intelligent tutoring systems that recognize sources of confusion or misunderstanding in students through their interactions with the system (Johnson, 2010; S. Lee et al., 2012; McQuiggan et al., 2008; Min, E. Ha, et al., 2014); and security applications that recognize terrorist plans (Jarvis et al., 2005).

Cohen et al., 1981 and Geib and Robert P Goldman, 2001 distinguish plan recognition according to the relationship between observer and observed agent: *keyhole, intended*, and *adversarial*. In *keyhole* recognition, the agent is not aware that it is being observed and is engaged in its own task; In *intended* recognition, the actor cooperates with the observer and chooses actions that can be easily understood by the observer; *adversarial* recognition assumes that de actor actively tries to deceive the observer through its actions.

Most research in goal recognition primarily focuses on developing better and more efficient techniques to recognize the plan or the goal of an acting agent given a sequence of observations of its actions. For example, imagine a simplistic agent-navigation scenario shown in Figure 1.1(a), where an agent is at $E3$, it can move in any of the four cardinal directions, and

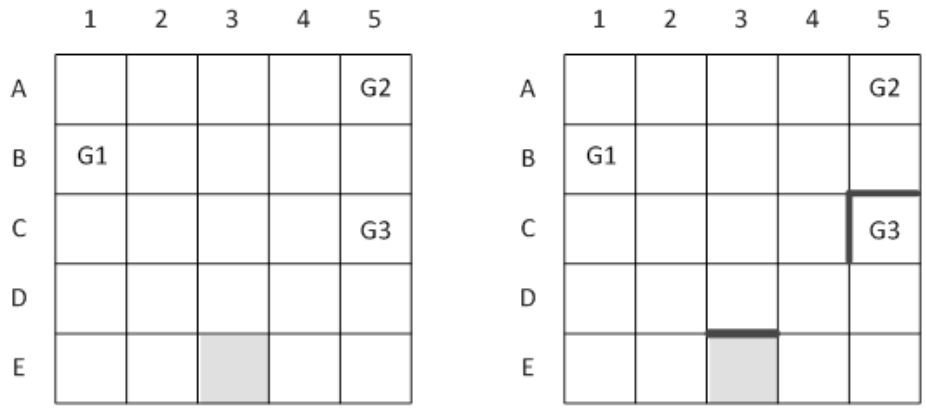


Figure 1.1: Example Problem (a) Original goal recognition problem with starting state at position $E3$ and possible goals G1 (at $B1$), G2 (at $A5$), and G3 (at $C5$). (wcd=4) (b) A design solution that blocks actions $(E3, up), (C4, right), (C5, up)$. ($wcd$=2)

its goal is one of three possible goals G1 (at $B1$), G2 (at $A5$), and G3 (at $C5$). Additionally, assume that it will move along the shortest path to its goal. Then, if it moves left to $E2$, we can deduce that its goal is G1. Similarly, if it moves right to $E4$, then its goal is either G2 or G3. However, if it moves up to $D3$, we cannot make any informed deductions. In fact, if the agent moves along any one of its shortest paths to goal G3, throughout its entire path, which is of length 4, we cannot deduce whether its goal is either G2 or G3! This example illustrates one of the challenges with this approach: there are often many ambiguous observations resulting from similar ways to accomplish different goals. As such, it is difficult to uniquely determine the agent's goal until a long sequence of actions is observed.

Therefore, Keren, A. Gal, et al., 2014 proposed an orthogonal approach to *modify the underlying environment of the agent*, in such a way that *the agent is forced to reveal its goal as early as possible*. They call this problem the Goal Recognition Design (GRD) problem. For example, if we block the actions $(E3, up), (C4, right), (C5, up)$ in our example problem, where we use tuples (s, a) to denote that action a is blocked from cell s, then the agent can make at most 2 actions (i.e., right to $E4$ then up to $D4$) before its goal is conclusively

revealed. Figure 1.1(b) shows the blocked actions. This problem finds itself relevant in many of the same applications of goal recognition because, typically, the underlying environment can be easily modified.

The transformation performed to the model through modifications is an instance of design optimization; thus, we require measuring and comparing models to identify the optimal solution. The seminal work (Keren, A. Gal, et al., 2014) introduced the notion of *worst-case distinctiveness* (*wcd*), a measure that assesses the ease of performing goal recognition in an environment. The *wcd* of a problem is the longest sequence of actions an agent can take without revealing its goal. The objective is to find a subset of feasible actions to make it infeasible such that the resulting *wcd* is minimized. In this problem, they make three explicit assumptions: (*i*) the agents in the system will act optimally (i.e., agents will move along one of the shortest paths to its goal); (*ii*) the outcomes of the actions of agents are deterministic; and (*iii*) the environment is fully observable (e.g., agent and observer have access to all variables). Other implicit assumptions are that all goals are equally important or possible and that all types of modifications have the same cost.

Goal Recognition Design and most of the work derived from it (Ang et al., 2017; Harman and Simoens, 2019; Keren, A. Gal, et al., 2015; Keren, A. Gal, et al., 2016a; Keren, A. Gal, et al., 2016b; Keren, A. Gal, et al., 2018; Keren, Xu, et al., 2020; Mirsky et al., 2017; Son et al., 2016) assume agents with deterministic action outcomes. In this thesis, I study how design can facilitate keyhole goal recognition in stochastic environments. Our approach uses the idea to formulate goal recognition as planning (Ramírez and Geffner, 2010) and uses Markov Decision Processes (MDPs) Mausam and Kolobov, 2012 to formulate the planning problem of the agents within the GRD problem. This choice is motivated by the fact that MDPs are often de facto models for representing planning problems with uncertainties. Real-world

applications inspire the assumptions made for each proposed setting. They aim to extend the applicability of GRD to cases where agents and observers interact with the physical world.

1.2 Overview of Contributions

This thesis presents a general model for the S-GRD framework, which supports four different settings or problems:

1. The **Optimal Stochastic Goal Recognition Design (OS-GRD)** problem, where we assume optimal agents, stochastic action outcomes, and full observability for agents and observer.

2. The **Partially Observable Stochastic Goal Recognition Design (POS-GRD)** problem, with optimal agents, stochastic action outcomes, and observers affected by sensor limitations in two ways: actions are no longer observable, and due to sensor resolution, states are only partially observable. Acting agents have full observability.

3. The **Suboptimal Stochastic Goal Recognition Design (SS-GRD)** problem, which assumes boundedly rational agents, stochastic action outcomes, and full observability for agents and observer.

4. The **Partially Observable Suboptimal Stochastic Goal Recognition Design (POSS-GRD)** problem, which combines the assumptions of POS-GRD and SS-GRD settings.

For each problem, we introduce new algorithms and thoroughly discuss related properties.

1.3 Research Output

Some of the material in this thesis appeared in the following publications:

- **Christabel Wayllace**, Sarah Keren, Avigdor Gal, Erez Karpas, William Yeoh, and Shlomo Zilberstein. Accounting for Observer's Partial Observability in Stochastic Goal Recognition Design. In Proceedings of the European Conference on Artificial Intelligence (ECAI), pages 2394-2401, 2020.
- **Christabel Wayllace**, Ping Hou, and William Yeoh. New Metrics and Algorithms for Stochastic Goal Recognition Design Problems. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 4455-4462, 2017.
- **Christabel Wayllace**, Ping Hou, William Yeoh, and Tran Cao Son. Goal Recognition Design with Stochastic Agent Action Outcomes. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 3279–3285, 2016.

1.4 Thesis Outline

The rest of the thesis is organized as follows:

In Chapter 2, we provide the background information that reviews classical and probabilistic planning, relevant work on goal recognition, design optimization, and GRD. Chapter 3 presents a description of the proposed S-GRD framework, its mathematical model, and the settings that supports: Optimal S-GRD, Partially-Observable S-GRD, Suboptimal S-GRD, and Partially-Observable Suboptimal S-GRD. Chapter 4 analyzes the properties of each setting and presents algorithms to compute *wcd* in each case. Chapter 5 focuses on the design stage and presents algorithms to minimize *wcd*. Chapter 6 empirically evaluates all proposed

algorithms as well as the usefulness of the approach. In Chapter 7, we discuss related work and Chapter 8 presents the conclusions and future work.

Chapter 2

Background

"Without leaps of imagination or dreaming,

we lose the excitement of possibilities.

Dreaming, after all is a form of planning."

– Gloria Steinem

Goal Recognition Design (GRD) stands at the intersection of goal recognition, design optimization, and classical planning. This thesis is a generalization of GRD problems and uses probabilistic planning, particularly Markov Decision Processes (MDPs), at its core. This chapter provides basic background information of classical and probabilistic planning with an emphasis on MDPs and common algorithms to solve them. Next, it presents a review of some relevant work on goal recognition using planning, a background on design optimization, and a more detailed description of GRD.

2.1 Classical Planning

Planning is a branch of AI that consists on devising a plan of action to achieve an agent's goals. The agent selects the next action to take based on a model of the problem that specifies how actions and sensors work, the agent's current situation, and the goal to be achieved. A planning problem is formed by the *model*, the *language* that expresses the model, and the *algorithms* that use the model representation to generate the behavior. In this section, we will review these concepts for *classical planning*.

Classical planning is the simplest form of planning where the actions have deterministic outcomes and the initial state is fully known. A classical planning model (Geffner and Bonet, 2013), can be represented as a tuple $\langle \mathbf{S}, s_0, \mathbf{A}, f, \mathcal{C}, \mathbf{G} \rangle$, where:

- $\mathbf{S}$ is a finite and discrete state space.
- $s_0 \in \mathbf{S}$ is a known initial state of the agent.
- $\mathbf{A}$ is the set of actions and $\mathbf{A}(s) \subseteq \mathbf{A}$ is the set of actions applicable in each state $s \in \mathbf{S}$.
- $f : \mathbf{S} \times \mathbf{A} \to \mathbf{S}$ is a deterministic state transition function, where $s' = f(s, a)$ is the successor state after applying action $a \in \mathbf{A}(s)$ in state s.
- $\mathcal{C} : \mathbf{A} \to \mathbb{R}^+$ defines the cost for each action.
- $\mathbf{G}$ is a set of goal states.

A plan $\pi = \langle a_1, \ldots, a_n \rangle$ is a sequence of applicable actions that brings an agent from the starting state s_0 to a goal state $g \in \mathbf{G}$. The cost of a plan $\mathcal{C}(\pi) = \sum_i \mathcal{C}(a_i)$ is the sum of the cost of each individual action in the plan. The goal is typically to find a cost-minimal plan $\pi^* = \text{argmin}_\pi \mathcal{C}(\pi)$.

The main challenge in planning is to achieve both generality and scalability. Thus, a solution cannot be improved by using any domain-specific strategy. A classical planner must accept any problem whose states and actions are defined in terms of a set of problem variables called *state variables*; any domain-specific information must be irrelevant and in any case the planner must provide a set of actions to take in order to achieve the goal.

An algorithm to solve a classical planning problem should take advantage of its structure. The challenge is to find a language to represent planning problems in compact form and expressive enough to represent a wide variety of problems. We will review two languages: STanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson, 1971), a simple and possibly the oldest classical planning language, and Planning Domain Definition Language (PDDL) (Ghallab et al., 1998), a language and syntax that has been used as the standard language in the planning competitions since 1998.

2.1.1 Planning Languages

STanford Research Institute Problem Solver (STRIPS)

STanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson, 1971) is a language based on Boolean state variables. A planning problem expressed in STRIPS is a tuple $P = \langle \mathbf{F}, \mathbf{I}, \mathbf{O}, \mathbf{S_G} \rangle$ (Geffner and Bonet, 2013) where:

- $\mathbf{F}$ represents the set of atoms or propositions of interest (usually called fluents).
- $\mathbf{I} \subseteq \mathbf{F}$ represents the initial state.
- $\mathbf{O}$ represents the set of actions. The actions $o \in \mathbf{O}$ are represented by three sets of atoms over $\mathbf{F}$ called the Add ($Add(o)$), Delete ($Del(o)$), and Precondition ($Pre(o)$)

lists. $Add(o)$ describes the atoms that the action o makes true, $Del(o)$ the atoms that o makes false, and $Pre(o)$ the atoms that must be true for the action to be applicable.

- $\mathbf{S_G} \subseteq \mathbf{F}$ represents the goal.

A STRIPS problem $P = \langle \mathbf{F}, \mathbf{I}, \mathbf{O}, \mathbf{S_G} \rangle$ encodes the classical state model $\langle \mathbf{S}, s_0, \mathbf{A}, f, \mathcal{C}, \mathbf{G} \rangle$ in compact form:

- The states $s \in \mathbf{S}$ are the possible collections of atoms over $\mathbf{F}$ where an atom $p \in \mathbf{F}$ is true in $s \iff p \in s$.
- The initial state s_0 is $\mathbf{I}$.
- The actions $a \in \mathbf{A}(s)$ are the ones in $\mathbf{O}$ with $Pre(a) \subseteq s$.
- The state transition function is $f(a, s) = (s \setminus Del(a)) \cup Add(a)$.
- The action costs $\mathcal{C}(a), \forall a \in \mathbf{A}$, are equal to 1 by default.
- The set $\mathbf{G}$ of goal states comprises the states s for which $\mathbf{S_G} \subseteq \mathbf{S}$.

Since the states in $\mathbf{S}(P)$ are represented as collections of atoms from $\mathbf{F}$, the number of states in $\mathbf{S}(P) = 2^{|\mathbf{F}|}$, where $|\mathbf{F}|$ is the number of atoms in P.

Planning Domain Definition Language (PDDL)

The *Planning Domain Definition Language* (PDDL) (Geffner and Bonet, 2013; Ghallab et al., 1998) supports STRIPS-style actions as well as a number of additional syntactic constructs in a notation originated in the Lisp programming language. PDDL planning tasks are expressed in two parts: the *general domain* and a specific *domain instance*. The general domain

describes predicates and actions while the specific domain creates object names that will replace the variables defined in the general domain and specifies the initial and goal states. A "requirement" flag in the general domain describes the PDDL fragment used by the encoding, which can include STRIPS and other notations as well as extensions for conditional effects, quantification, etc. PDDL has evolved over time adding minor and major changes to cover new requirements; however, most planners do not support the entire PDDL and can even interpret some constructs in a different way according to the requirements of the algorithms used to solve the problem.

2.2 Probabilistic Planning

Classical planning assumes a fully known initial state and deterministic action outcomes. Those assumptions are inappropriate for many domains, the world dynamics can affect the action effects in many ways that cannot be completely modeled. When things do not work as expected due to insufficient knowledge or when a situation requires a complex model, it is easier to use a probabilistic model. For instance, when a key sometimes fails to start an old car, it is easier to assign a success probability than to consider all possible combinations of thing that could cause the failure. The initial world states are also source of uncertainty: will the highway be crowded?

Probabilistic planning is an extension of classical planning that assumes uncertainty over the initial world state, action effects, and events. It presumes the agent's model and goals, as well as the domain dynamics, are known and focuses on finding the behavior of the agent to achieve its objectives. Unlike classical planning, there is not a unique formulation for a probabilistic planning problem; the formulation differs according to the approach to solving the problem. For instance, Kushmerick et al., 1995 use a probability distribution over possible

world states to model imperfect information about the initial world state and assumes that actions have a conditional probability distribution over changes to the world. However, most researchers agree on formulating it as a Markov Decision Process (Lago Pereira et al., 2008; Littman et al., 1998; Majercik and Littman, 1998; Yoon, Fern, and Givan, 2007; Yoon, Fern, Givan, and Kambhampati, 2008).

2.2.1 Markov Decision Process (MDP)

Markov Decision Processes (MDPs) provide a mathematical formalism to model decisions with uncertain outcomes of actions that an agent must take. In general, MDPs consist of a set of states, a set of actions, a transition model that describes the outcome of each action in each state, i.e, the probability of changing to a new state after taking one action in the current state, and an objective function. The solution of an MDP determines the action to take in each one of the states where the agent might arrive. A solution of this type is called a *policy*. An optimal solution optimizes the objective function that could be for instance, minimizing the incurred cost when traversing a path. MDPs can model problems in numerous domains, especially when they present the following characteristics (Mausam and Kolobov, 2012):

- Uncertain Domain Dynamics: Even though MDPs can model deterministic domains, their more valuable contribution is in stochastic domains.
- Sequential Decision Making: MDPs are useful when the agent needs to plan a sequence of actions to obtain maximum benefits in the future.
- Cyclic Domain Structures: The algorithms used to solve MDPs optimally handle situations where the agent has the possibility to revisit a state, for example, if after taking one action the state of the agent does not change.

- Nature is Fair: Domains where the outcomes are random instead of caused by another agent's decisions are best modeled by MDPs.
- Full Observability and Perfect Sensors: This characteristic implies that the agent has access to the whole state space and that its observations are correct.

Since MDPs can model such a broad spectrum of domains, proposing efficient solution techniques becomes difficult. As a result, researchers conceived specialized MDPs with new restrictions added to the general model. These restrictions try to model more specific types of problems while still covering significant classes of real-world scenarios. This thesis focuses on MDPs modeling scenarios where the agent has a limited but unknown time to make decisions, which implies that the system will eventually reach a terminal state or goal. In particular, it uses *Stochastic Shortest-Path* (SSP) MDPs (Mausam and Kolobov, 2012), which are more general than other well known classes such as finite-horizon and infinite-horizon discounted reward MDPs.

An SSP-MDP is represented as a tuple $\langle \mathbf{S}, s_0, \mathbf{A}, \mathcal{T}, \mathcal{C}, \mathbf{G} \rangle$ where:

- $\mathbf{S}$ is a finite set of all possible states of the system.
- $s_0 \in \mathbf{S}$ is a start state.
- $\mathbf{A}$ is a finite set of all actions an agent can take.
- $\mathcal{T} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to [0, 1]$ is a transition function that specifies the probability $\mathcal{T}(s, a, s')$ of transitioning from state s to s' when action a is executed
- $\mathbf{G} \subseteq \mathbf{S}$ is the set of all goal states s.t. $\forall g \in \mathbf{G}$, $a \in \mathbf{A}, \mathcal{T}(g, a, g) = 1$ and $\mathcal{C}(g, a, g) = 0$, that is, the goal states are terminal.

- $\mathcal{C} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to \mathbb{R}^+$ is a cost function that gives the cost $\mathcal{C}(s, a, s')$ of executing action a in state s and arriving to state s' under two conditions:

    1. There exists at least one proper policy, which is a policy that is guaranteed to reach a terminal state.
    2. Every improper policy must incur an accumulated cost of ∞ from all states from which it cannot reach the goal with probability 1.

These two conditions imply that some proper policy for the SSP-MDP is preferable to all improper ones. Under the SSP-MDP definition conditions, in expectation, an optimal policy is the "shortest" way of reaching the goal from any given state. At a higher level, making the rewards for all actions negative means the agent is effectively paying a cost every time it executes an action; thus, it is motivated to reach a goal state as fast as possible. If we consider an SSP-MDP with one goal and a deterministic transition function, that is, $\mathbf{G} = \{g\}$ and $\mathcal{T} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to 1$, the problem turns into the classical problem of finding a weighted shortest path in a graph. This property is the reason for its name.

As stated before, a solution of an MDP is a policy that maps states to actions. In the next subsection, we will present a well-known algorithm called Value Iteration for finding an optimal policy.

2.2.2 Value Iteration (VI)

The Value Iteration (VI) algorithm was proposed by Bellman, 1957; the basic idea is to compute the utility value for each state and use those values to select an optimal action for each state. VI is based on the Bellman equations (Bellman, 1957) that have a unique solution

and mathematically represents the optimal solution of an MDP:

$$\begin{aligned} V^*(s) &= 0 && \text{if} \quad s \in \mathbf{G} \\ V^*(s) &= \min_{a \in \mathbf{A}} Q^*(s,a) && \text{if} \quad s \notin \mathbf{G} \\ Q^*(s,a) &= \sum_{s' \in \mathbf{S}} \mathcal{T}(s,a,s') \big[\mathcal{C}(s,a,s') + V^*(s') \big] \end{aligned} \tag{2.1}$$

Where $V^*(s)$ is the minimum expected cost to reach a goal starting from a state s and $Q^*(s,a)$ is defined as the minimum expected cost to reach a goal starting in state s if the first action is a. VI first initializes all values V_0 arbitrarily; in the n^{th} iteration it computes a new approximation V_n for each state value using the successor values from the previous iteration V_{n-1}:

$$V_n(s) \leftarrow \min_{a \in \mathbf{A}} \sum_{s' \in \mathbf{S}} \mathcal{T}(s,a,s') \big[\mathcal{C}(s,a,s') + V_{n-1}(s') \big] \tag{2.2}$$

VI successively approaches to $V^*(s)$ with a $V_n(s)$ function that converges to $V^*(s)$ in the limit as n tends to infinity. The action chosen by the policy for each state s is then the one that minimizes $V^*(s)$. VI is an iterative algorithm that performs a Bellman update on each state in each iteration and there is one Bellman equation per state. The difference between the expected cost of a state in two consecutive iterations is called the residual of that state and the largest residual is called the residual error. The algorithm terminates when the values converge, that is, the residual error is less than a user-defined threshold ϵ.

Theoretical Properties of VI

As any iterative algorithm, VI is characterized by four properties: convergence, optimality, termination, and running time.

- VI **converges** in the limit, without initial restrictions, to the **optimal** values, that is, for the SSP-MDP: $\forall s \in \mathbf{S}, \lim_{n\to\infty} V_n(s) = V^*(s)$, irrespective of the initialization V_0.

- The *residual* of a state is the amount of change in the values between two successive iterations. In an SSP-MDP $Res^V(s) = |V_{n-1}(s) - V_n(s)|$ with $V_n(s)$ as defined in Eq. 2.2. The residual w.r.t. a value function V is $Res^V = \max_{s\in\mathbf{S}} Res^V(s)$ and it is called $\epsilon - consistent$ if $Res^V < \epsilon$. Since VI is *monotonic*, once all residuals are less than ϵ, they will remain less than ϵ for all subsequent iterations, and so, it guarantees **termination**.

- In the worst case, a singe Bellman update runs in $O(|\mathbf{S}||\mathbf{A}|)$ (checking all actions and all successor states) and a single iteration requires $O(|\mathbf{S}|)$ updates, so the total **running time** is $O(|\mathbf{S}^2||\mathbf{A}|)$. Bonet, 2007 established a polynomial bound for SSP-MDPs when costs are positive and the initial values are *admissible*, i.e., a lower bound of the optimal cost.

2.2.3 Topological VI (TVI)

VI suffers from a limitation that it updates each state in every iteration even if its expected cost has converged. Topological VI (TVI) (Dai, D. S. Weld, et al., 2011) addresses this limitation by detecting the MDP structure and updating states grouped in topological sequences. It first divides the MDP into *strongly connected components* (SCCs) and repeatedly updates the states in only one SCC until their values converge before updating the states in another SCC. Since the SCCs form a directed acyclic graph, states in an SCC only affect the states in upstream SCCs. Thus, by choosing the SCCs in reverse topological sort order, it no longer needs to consider SCCs whose states have converged in a previous iteration.

TVI is guaranteed to terminate and to converge to an optimal value function. Essentially, TVI decomposes an MDP into sub-problems or components and solves each one of them using VI. The solution of a sub-problem depends only either on its states or on converged ones (treated as sink states).

2.2.4 MDPs with Dead Ends

A dead end is a state $s \in \mathbf{S}$ such that no policy can reach a terminal state from s in any number of steps. SSP-MDPs assume that there exists a way for the agent to reach a terminal state or goal with probability 1 from each state. Therefore, SSP-MDPs do not handle dead ends.

Since dead ends model failures in real-world applications, the inability to control them reduces the applicability of SSP-MDPs. Hence, it is essential to extend their solution techniques to handle these kinds of states. One consequence of considering dead ends is that the VI algorithm will never converge. An option to avoid this problem is to transform an MDP with dead ends (called finite penalty stochastic shortest-path MDP with dead-ends) to an SSP-MDP by adding a new action with a high cost that causes a transition to a goal state with probability 1 (Mausam and Kolobov, 2012). The penalty should be high enough to force the agent to prefer a better policy or, in case of no alternative options, the start state's value becomes large and triggers the process to stop.

The option discussed in the previous paragraph assumes that a dead end is easy to identify; this is possible if no action applies to a non-goal state (*explicit* dead end). However, if the available actions do not lead to a goal, we face a case of *implicit* dead ends that are costly to detect. MDP solvers use different techniques, for instance, Kolobov, D. S. Weld, et al., 2009 classify states to find states with known properties, explicit dead ends, and others. When

a state s is in the third group, the solver uses a determinized version of the domain and a classical planner to find a plan; failure indicates that s is a dead end. A different faster approach uses machine learning to identify implicit dead ends (Kolobov, D. Weld, et al., 2010).

While our work assumes SSP-MDP agents, the process to redesign the model causes dead ends, especially with the use of action removal.

2.2.5 Probabilistic PDDL (PPDDL)

Although the definition of an SSP-MDP is short, it is not easy to describe MDP *instances*. A flat representation assigns IDs to each state, but in this case, the states are not informative, which prevents algorithms from taking advantage of the structure of a particular problem. Further, some domains could contain a vast number of states, causing the transition function to use more space and time than the state space itself. A factored representation specifies a state as a combination of *state variables*, which helps to reduce the description of the state space and other components such as the transition function. Similar to the planning domain definition language (PDDL) presented in Subsection 2.1.1 (p. 12), the probabilistic PDDL language (PPDDL) uses a factored representation and takes advantage of the problem structure. PPDDL is convenient when actions have correlated effects and few outcomes. This subsection reviews PPDDL in more detail.

PPDDL, specifically PPDDL1.0, is an extension of PDDL2.1 that expresses Planning Domains with Probabilistic Effects (Younes and Littman, 2004). The main extension is the support for probabilistic effects where rewards are modeled by fluents to model MDPs. The syntax for probabilistic effects is `(probabilistic` $p_1 e_1 \ldots p_k e_k$ `)` where effect e_i occurs with probability p_i having $p_i \geq 0$ and $\sum_{i=1}^{k} p_i = 1$. For instance, in a BLOCKWORLD domain, the probabilistic

effect that models the effect of stacking one block on top of another block can be represented in PPDDL as:

```
:effect( (probabilistic 0.9
         (and (not (holding ?x)
              (not (clear ?y))
              (clear ?x)
              (handempty
              (on ?x ?y))))))
```

The previous expression means that after executing the action, the block `x` will be on top of block `y` with probability 0.9. Note that it is implicit that with probability 0.1, the state remains unchanged. Other statements are also required to define the state; for example, the agent is no longer holding block `x`, or the top of block `y` is no longer clear. A new flag in the requirements section, `:probabilistic-effects`, signals the support of probabilistic effects.

As stated earlier, Markovian rewards can be encoded using fluents. The reserved word *reward* is used to represent the total accumulated reward due to the execution of the action and it is restricted to action effects of the form $(\langle additive - op \rangle \langle rewardfluent \rangle \langle f - exp \rangle)$ where:

- $\langle additive - op \rangle$ can be `increase` or `decrease`.
- $\langle f - exp \rangle)$ is a numeric expression not involving *reward*.

The requirement flag, `:rewards`, signals that Markovian rewards are utilized. Domains that use both probabilistic effects and rewards can use instead the requirement flag `:mdp` which implies `:probabilistic-effects` and `:rewards`.

Goals statements use the same syntax as in PDDL, `(:goal` ϕ`)`. In the probabilistic context, the probability of achieving ϕ is maximized by default but it can be changed by explicitly specifying an optimization metric. When a planning problem declares the `:rewards` requirement, the plan objective by default is to maximize the expected reward. It is also possible to specify a one-time reward for entering to the goal state by using `(:goal-reward` f`)`, where f is a numeric expression.

2.2.6 The Problem of Finding Best-k Policies

In many real-world scenarios, providing only one optimal solution is not enough. For example, in situations where the optimal solution gives a trajectory where some state has a high risk of failure, it might be preferred to take a less risky, slightly suboptimal path. Even if the reward function incorporates the risk factor, it is not always easy to account for all world dynamics in the model. The criteria to optimize a solution can also be diverse and multi-objective planning is slow. Therefore, a good alternative is to look for multiple solutions optimal for a single criterion and later choose one that best accommodates all requirements.

Solving the *k best policies* problem naïvely is exponential in k. The main idea of the algorithm proposed by Dai and Goldsmith, 2009 is to first define a metric and a lexicographic ordering to compare and rank policies by (1) The expected value at the starting state; (2) Their "distance" to a better policy (the number of states for which two policies differ); and (3) Their lexicographical order. Then, starting from an optimal policy, the next $k-1$ policies can be found one at a time. To find the next best policy, the algorithm prunes the search

space of policies based on a theorem that essentially states that among the best k policies, there is at least one that differs from the best k policy in one state.

We now explain the method in more detail. Dynamic programming finds optimal policies because it assumes that every subproblem is optimal; this assumption is no longer valid for suboptimal policies. However, finding the next best policy is possible by reducing the problem to many optimal planning problems. For instance, given the best and second-best policies π_1 and π_2, π_3 either differs from π_1 on s_0 and from π_2 on s_0, or from π_1 on s_0 and from π_2 on s_1, and so on. In fact, comparing all states requires to solve $\mid \mathbf{S} \mid^2$ optimal planning problems, in general $\mid \mathbf{S} \mid^k$ many. The complexity of this approach when Value Iteration is used becomes $\mid \mathbf{S} \mid^k O(VI)$.

To improve the algorithm, the authors propose a theorem that formally states that "among the *top-k* policies, there exists π_m with $m < k$, such that the *k-th* best policy differs from π_m on exactly one state." The new algorithm keeps an ordered set of candidate policies $\mathcal{P}$ initially empty. The first policy is an optimal policy computed using VI. To find the *i-th* best policy, the algorithm generates $k-i+1$ distinct policies as candidates. These candidates (1) must not be duplicates of any policy in $\mathcal{P}$, and (2) each differs from π_{i-1} on exactly one state. The authors also prove that $\pi_i \in \mathcal{P}$, in fact, π_i is the best policy in $\mathcal{P}$.

There are $(\mid \mathbf{A} \mid -1)\times \mid \mathbf{S} \mid$ policies that have exactly one state different from π. Finding the best $k-i+1$ of them has a complexity of $k\times \mid \mathbf{A} \mid \times \mid \mathbf{S} \mid \times O(\textit{policy evaluation})$. Note that the complexity of keeping the list $\mathcal{P}$ sorted is $O(k^2 log(k))$ so it does not increase the total

complexity. Hence, the complexity of the new algorithm using VI is $k\times \mid \mathbf{A} \mid \times \mid \mathbf{S} \mid \times O(VI)$, which is a linear function of k.

A later paper (Dai and Goldsmith, 2010) proposes an approximate algorithm that skips many "less useful" or trivial policies, which are policies that differ from other policies only in unreachable states. In this case, any trivially extended policies from the current best list $\mathcal{P}$ are excluded.

2.3 Plan and Goal Recognition

Plan recognition (PR) is the problem of inferring an agent's plan and goal given observations of its behavior. On the other hand, *goal recognition* (GR) focuses only on the agent's goal inference, therefore, GR is considered a subproblem of PR. Both problems fall within the scope of plan, activity, and intend recognition (PAIR), where activity recognition focuses on problems that deal directly with data obtained from physical sensors and plan and intend recognition concentrate on identifying the high-level goals and intends of an acting agent (Sukthankar et al., 2014).

There is considerable interest in goal and plan recognition not only in the AI community but also in Psychology, where prediction of agent's behavior is of interest. More than four decades ago, Schmidt et al., 1978 studied the relevance of PR to both Psychology and AI. Nowadays, research in cognitive sciences use AI models to understand human actions or to model human plan recognition (Baker, R. Saxe, et al., 2009; Baker and Tenenbaum, 2014; Bonchek-Dokow

and Kaminka, 2014; Tauber and Steyvers, 2011). At the same time, researchers in AI use concepts of mirroring (the empathetic human response to observations) to solve GR problems (Vered et al., 2018) or add human skills like empathy to PR models (Shvo, 2019).

The intersection of Psychology and AI with respect to plan and goal recognition lies on their shared interest of multi-agent interaction. Applications using PR or GR are diverse and typically involve interactions between human and artificial agents. PR has been applied in education, analyzing student's interaction with exploratory learning environments (Amir and Y. Gal, 2013; Uzan et al., 2015); in computer games, to give computer-controlled players the ability to recognize the opponent's strategies (Kabanza et al., 2010) or in the case of games with multiple solution paths, to detect the opponent's goals (GR) (E. Y. Ha et al., 2011; Min, Mott, et al., 2016). Other fields include security, where it is used to correlate and analyze security alerts (Qin and W. Lee, 2004) or as part of intrusion prevention systems (G. Chen et al., 2010), and human-robot interaction (HRI), for human-robot collaborative teams (Levine and Williams, 2018).

Most applications in plan and goal recognition require the following properties (Sukthankar et al., 2014):

- **Speed:** Plan and goal recognition are usually performed online and the process must finish before the agent arrives to the goal. Ideally, it should take a fraction of the time required by the agent to take the next action.

- **Precision and Recall:** It is desirable that the predictions are correct (precision) and at every opportunity (recall).
- **Early Prediction:** Applications need early and accurate predictions, the problem must be solved *as early as possible*, that is, with the minimum amount of observations.
- **Partial Prediction:** When full prediction is not immediately available, applications should be able to use partial information.

Nevertheless, not all solutions have all the properties, usually systems will sacrifice one for another.

2.3.1 Plan Recognition as Planning

The general idea of solving a PR problem is to match a given sequence of observations to a plan and goal (or only to a goal in the case of GR). The problem requires as input a set of actions together with a means to encode the correct use of those actions. They can be provided through plan libraries (Geib, 2009; Geib and Kantharaju, 2018; Kautz and J. F. Allen, 1986; Maraist, 2017; Massardi et al., 2019; Singla and Mooney, 2011) or via planning domains (domain theory) (Baker, R. Saxe, et al., 2009; Ramírez and Geffner, 2009; Ramírez and Geffner, 2010; Ramírez and Geffner, 2011). An alternative input for GR problems is data (corpora) that is used to train the model of a recognizer (Blaylock and J. Allen, 2003; E. Y. Ha et al., 2011). In this thesis, we are interested in techniques using domain theory that

rely on the idea that PR is *planning in reverse* and use *planning algorithms* and *representation languages* to solve it.

Ramírez and Geffner, 2009 proposed a mapping of plan recognition to a planning problem where the original domain (called theory) is transformed into a new planning domain that accounts for the observations (actions) and can be solved using planning algorithms. They use a heuristic estimation based on the cost from the current state to discard candidate goals. Later, Ramírez and Geffner, 2010 used off-the-shelf classical planners and Bayes theory to find a probability distribution over the goals given a sequence of observations. Researchers in cognitive sciences (Baker, R. Saxe, et al., 2009) independently used a similar technique with MDP models to explain human actions. Ramírez and Geffner, 2011 extend the approach for GR assuming partially-observable MDP (POMDP) settings where the acting agent has partial observability of its current state.

This work has inspired many researchers to use inverse planning as a technique to solve PR problems. Sohrabi et al., 2016 and Riabov et al., 2020 extend the work in classical domains to account for unreliable observations that a plan's actions cannot explain a particular goal. Additionally, they assume that the (possibly noisy) observations are over fluents instead of actions, as in most PR cases, and find probability distributions over the plans instead of goals. Since the observations may not be accurate, the approach does not discard a possible goal in case of an optimal path for it (given the current set of observations) is not available; instead, the probability of that goal decreases to be the lowest. The experiments use two

planers, one of which is a top-k planner that showed better performance in the presence of missing or noisy observations.

Pereira et al., 2017 provide two landmark-based heuristics for GR approaches that use inverse planning. Landmarks are properties or actions that every plan must have in every plan execution towards a goal. The proposed heuristics remove the need to run the planner twice per goal as required in previous work (Ramírez and Geffner, 2009; Ramírez and Geffner, 2010). The use of landmarks, however, does not support unreliable observations as it compares them with landmarks for all possible goals to estimate the correct goal.

Most approaches explicitly provide a set of possible goals from which the recognizer will choose one or assign a probability distribution reflecting their likelihood of being the actual goal. Pattison and Long, 2010 consider that goals should be any subset of fluents reachable from the start state. The authors assign probability distributions to subsets of fluents and call them *goal hypothesis*. Since this assumption creates a huge goal hypothesis space, they suggest an approximation that works in domains with independent goals, i.e., goals that are not strongly correlated. They analyze the domain to find useful properties to reduce the goal hypothesis space and update it after each observation. The heuristic assigns lower probabilities to fluents further away from the current state, favoring close to optimal plans. A later work (Pattison and Long, 2011) does not assume any rationality and finds subgoals using a Bayesian approach; it also claims to scale better than the approach by Ramírez and Geffner, 2010 but still requires mutually-exclusive goals. In this approach, candidate goals are never eliminated and assume the agent can revisit search space areas.

Having subgoals is not the only reason for an agent to revisit states or have a seemingly erratic behavior: deception, where an agent tries to hide or disguise its real goal, is a common reason found in adversarial GR. Masters, 2019 study GR and deception in path planning and consider deceptive agents that, instead of acting erratically, find other more optimal ways and achieve either the same or a better result. A deceptive agent's objective is to confuse a goal recognizer to the point where either none of the goal candidates are possible anymore or when the most probable goal keeps changing with more sequences of observations. The authors work on both sides: on the one hand, they propose a self-modulating formula that reduces the confidence of the goal recognizer when observing erratic behavior; and on the other, they maximize deceptive planning at the lowest cost. They consider two forms of deception: simulation (showing the false) and dissimulation (hiding the real). One important contribution used to find a strongly deceptive strategy with the lowest cost, is the finding that in path planning, probabilistically ranking the goals to infer the true goal is independent of the observations. Therefore, it is possible to build a map of probabilities for the whole state space before the agent acts in the environment (Masters and Sardina, 2017a). This is an alternative offline approach to evaluate a GR problem, equivalent to the first stage of GRD, reviewed later in Section 2.5.

While the reviewed work so far presumes deterministic environments, there has been work within this trend assuming stochastic action outcomes. Baker, Tenenbaum, and R. R. Saxe, 2007 uses inverse planning to infer the goals of people. The paper analyzes three distinct goal configurations that help to recognize goals in different contexts. The first configuration

assumes only one goal and interprets deviations as noise or bounded rationality; the second configuration models agents with complex (i.e., with subgoals); the last option assumes that the agent changes goals over time. In their work, Baker, Tenenbaum, and R. R. Saxe, 2007 assume agent's states and actions can change over time, whereas world's states are stationary. The authors consider an SSP-MDP (Mausam and Kolobov, 2012) agent whose degree of rationality is modeled with a softmax Bellman operator that provides a probability distribution over actions. A temperature parameter models the way agents choose an action, which could range from optimal to random behavior. The probability distribution over the candidate goals is obtained using Bayesian inference.

Ramírez and Geffner, 2011 propose to infer a probability distribution over all possible goals when the agent has partially-observable states and stochastic action outcomes. The observer receives a subset of the observations available to the agent and can observe actions only partially, so she must fill the gaps. Same as Baker, Tenenbaum, and R. R. Saxe, 2007, their approach uses the softmax Bellman operator (or Boltzmann policy) to model the likelihood of choosing an action given that the agent is pursuing a particular goal G.

2.4 Design Optimization

Design optimization is a design methodology used in engineering where a mathematical optimization problem supports the selection of an optimal design among multiple alternatives. The online dictionary Lexico (OxfordUniversityPress, 2019) defines *designer* as a person who

plans the form, looks, or application of something before it is built. If it is possible to define the mathematical model of the designed object, then it is possible to modify its parameters and generate different alternatives. The latter implies that the designer must select the *most desirable* alternative. A rational choice requires a *criterion* to assess alternatives and to rank them (Papalambros and Wilde, 2000).

The evaluating criterion is rarely unique; it depends on the application, point of view, and the designer's judgment; a criterion can also change over time. Like all components of the model, the criterion is an approximation of reality, and it is useful only under the model assumptions. A design model that includes an evaluation criterion is called an *optimization model*, where the selected design is the optimal design, and the criterion is the *objective* of the model. Since a model is only an approximation of a system or design, there are different degrees of "success". A successful model that is also supported by accumulated empirical evidence often becomes a *law*.

A new design model can be used to generate alternatives by manipulating the values of the design variables. Also, changes in the design parameters can show the effect of environmental factors. In the case of product enhancement or *redesign*, we are usually interested in *small* changes that improve the product's performance. In such situations, the model is used to predict the effects of the changes. In addition to the criterion used and the design variables, design optimization should also consider the limitations or available resources, such as natural laws or user preferences. Limitations define the design requirements, and their selection is intimately related to the first two points.

In general, the formal mathematical model of design optimization is:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ & x \in \mathcal{X} \subseteq \mathbb{R} \end{aligned}$$

Where f is the objective criterion, $\mathbf{h}, \mathbf{g}$ are the set or system of (functional) constraints, and $\mathcal{X}$ is the set constraint.

Optimizing the design for large and complicated systems may be costly in computer resources, time, or both. In those cases, the usual practice is to use approximations and exploit characteristics of the model. Some methods decompose the problem into successively smaller problems, or stop the iterations when achieved sufficient improvement (J. Gero, 2012). Popular techniques use pair-wise evaluations of variables, follow a specific order to change variable values, or use heuristics to prune the search space.

Archer, 1968 and Rosenman and J. S. Gero, 1985 describe design as "a goal-directed problem-solving activity and basically a decision-making process" and so, taking the best design as a goal, planning methods are also a valid solution approach. Particularly, Rosenman and J. S. Gero, 1985 decompose a complex problem into subproblems, and the designer finds a set of feasible alternatives for each subsystem. The model of the problem represents each subsystem as a *stage* and the feasible alternatives per subsystem as *states*. The connections

among states are related to the objective function, i.e., if their interaction affects the objective, they are connected. The solution to the problem modeled as a graph is then the shortest path, and both the whole problem and the subproblems can use the same solution approach. In this model, the optimization procedure uses the stages as variables and the states as their alternative values.

2.5 Goal Recognition Design (GRD)

A Goal Recognition Design (GRD) problem (Keren, A. Gal, et al., 2014) is represented as a tuple $T = \langle P_0, \mathcal{D} \rangle$, where P_0 is an initial goal recognition model and $\mathcal{D}$ is a design model. The initial model P_0, in turn, is represented by the tuple $\langle D, \mathbf{G} \rangle$, where $D = \langle \mathbf{S}, s_0, \mathbf{A}, f, \mathcal{C} \rangle$ captures the domain information and $\mathbf{G}$ is a set of possible goal states of the agent. The elements in the tuple D are as they are described in classical planning except that all actions have the same cost of 1. The *worst-case distinctiveness* (*wcd*) is a measure to assess P_0 and represents the length of a longest sequence of actions $\pi = \langle a_1, \ldots, a_k \rangle$ that is the prefix in *cost-minimal* plans $\pi^*_{g_1}$ and $\pi^*_{g_2}$ to distinct goals $g_1, g_2 \in \mathbf{G}$. Intuitively, as long as the agent executes π, it does not reveal its goal to be either g_1 or g_2.

A design model $\mathcal{D} = \langle \mathcal{M}, \delta, \phi \rangle$ (Keren, A. Gal, et al., 2018) includes three components: The set $\mathcal{M}$ of modifications that can be applied to a model; a modification function δ that specifies the effect each modification $m \in \mathcal{M}$ has on the goal recognition setting to which it is applied; and a constraint function ϕ that specifies the modification sequences

that can be applied to a goal recognition model. In the original GRD problem definition, action removal is the only type of modifications allowed in the design model. An action is disallowed or removed from the model only if the cost to reach any of the possible goals does not increase. In later GRD definitions, the model supports arbitrary modifications; Keren, A. Gal, et al., 2019 describe three additional modifications (suitable for GRD extensions described in Section 2.6, p. 37): action conditioning, sensor placement, and single-action sensor refinement. Action conditioning forces ordered sequences of actions, which, similar to action removal, is a mechanism to eliminate legal plans. Sensor placement exposes a previously non-observed action, and single-action sensor refinement maps an action to a *single* (possibly noisy) observation.

The objective in GRD is to find a feasible modification sequence that, when applied to the initial goal recognition model P_0, will minimize the *wcd* of the problem.

2.5.1 Computing *wcd*

The baseline approach to compute *wcd* is to enumerate all possible legal plans (optimal plans in this case) and mark prefixes common to more than one goal to find the largest non-distinctive prefix. The allowed plans result from an exhaustive exploration of the state space using BFS, and the search stops at the level of the most distant goal. A backward search of the tree, level by level, starting from the most distant leaves, and keeping track of each node's goals, could find the largest path common to two goals.

A more optimized version constructs a graph where each node represents sub-paths, and edges represent available actions (the root node is the initial state). The algorithm traverses the graph in a BFS fashion but does not expand nodes representing distinctive paths, i.e., paths with no shared goals, and the *wcd* value is the length of the sub-paths expanded in the last iteration. Finding distinctive paths requires planner calls per node to check whether a sub-path is part of an optimal plan; therefore, even after pruning some of the search space with an upper bound, this method does not scale well.

Finally, Keren, A. Gal, et al., 2014 present a third method based on the problem's compilation into a planning problem. This approach models one agent per candidate goal and finds the *wcd* by looking for the longest path that those agents may share. Their compilation assumes two agents; in case of more candidate goals, they find the maximum *wcd* among all pairs. In the planning problem, each agent tries to achieve its target, but they get a higher reward for "working together". The reward is bounded to avoid an agent diverging from its optimal path. This approach is solved using off-the-shelf classical planners.

2.5.2 Reducing *wcd*

To reduce *wcd*, any modification to the model needs to affect the optimal policies reaching one or more goals. Therefore, the authors chose to remove actions from the original model. Note that although not explicitly stated, action removal refers to removing state-action pairs. For example, in Figure 2.1(b), used in the introduction and copied here for convenience, the

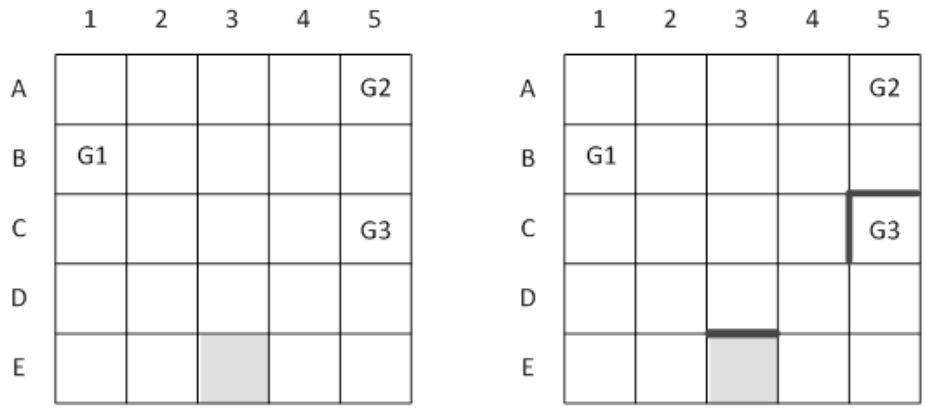


Figure 2.1: Example Problem (a) Original goal recognition problem with starting state at position $E3$ and possible goals G1 (at $B1$), G2 (at $A5$), and G3 (at $C5$). (wcd=4) (b) A design solution that blocks actions $(E3, up), (C4, right), (C5, up)$. ($wcd$=2)

removed actions are $(E3, up), (C4, right)$, and $(C5, up)$, not just up or $right$. For simplicity, we will use actions in this section to refer to state-action pairs.

The basic approach to reduce the wcd is an exhaustive search in the space of combinations of state-action pairs modeled as a tree. Each node contains removed actions. The root is an empty set representing the original goal recognition problem, i.e., no removed actions. Direct edges connect nodes where parents have one element less than their children.

The algorithm traverses the tree using BFS and explores a node only if the original cost to reach a goal does not increase. The search stops when $wcd = 0$ or when there are no more actions to remove. The output contains the smallest set of removed actions for the minimum wcd. While complete, this algorithm needs to explore all actions in the model. The authors observed that removing an action not used to compute the original wcd does not change the

wcd for the new problem. Therefore, a pruning method consists of not considering nodes with actions that were not optimal in the models represented by their parent nodes.

2.6 GRD Extensions

Keren, A. Gal, et al., 2015 extend the GRD model to account for suboptimal agents where the agent has a budget to deviate from its optimal path. They consider two types of suboptimal agents: a naïve agent and a deceptive agent, who tries to mislead the observer by following the longest path towards a fake goal while still being able to achieve its real goal within the allowed budget. Keren, A. Gal, et al., 2016a assume partial observability of agent actions; the *wcd* represents, in this case, the longest sequence of actions an agent can take before the *observed* portion of the trajectory reveals its true goal. Additionally, the paper presents a new type of modification that allows the exposure of non-observable actions. In a subsequent paper, Keren, A. Gal, et al., 2016b generalize the model to also consider non-deterministic sensors. These types of sensors cause the observer to have partial and possible noisy observability of agent actions. The modification used to improve the observer's sensor model chooses actions to make it fully observable.

Other limitations can also affect the actor; a recent paper (Keren, Xu, et al., 2020) assumes actors with incomplete knowledge about the environment. Therefore, they make assumptions about unknown variables but choose plans with minimum assumptions. The new setting models the observer or recognizer as an agent that knows everything and provides some

information to the actor to figure out the real goal faster. The high-level idea is to help the actor reduce extra steps to acquire knowledge to reach its goal. The challenge is to determine the information to provide, as the space is vast and more information is not necessarily better for goal recognition. This type of modification is called information shaping and is applied offline as in previous approaches.

Keren, A. Gal, et al., 2018 redefined the GRD model as a tuple containing an initial goal recognition model and a design model. The design model includes a set of modifications, a function specifying each modification effect, and a constraint function that determines the allowed modification sequences. The model presented in Section 2.5 (p. 33) used this formulation. Further, the authors present in this paper a class of modifications that allows pruning in the modification space by using strong stubborn sets (Valmari, 1989).

Finally, Keren, A. Gal, et al., 2019 generalize the GRD model once more for deterministic environments by adding a set of observations, a set of legal policies, and a sensor function to the initial goal recognition model. The new model accounts for all extensions in a deterministic setting. In contrast to goal recognition models, the set of observations contains *all* possible observation sequences that *could* be observed. While previous papers assumed unitary costs, this journal generalizes it to arbitrary action costs.

Chapter 3

Stochastic Goal Recognition Design (S-GRD)

"It is the existence of goals which makes design purposeful

and necessitates decisions about the best ways to achieve those goals."

– John Gero

No artificial or living creature can recognize goals with a hundred percent of certainty. Even in the most favorable conditions, there are situations where ambiguity is unavoidable; otherwise, playing charades would not be fun! Indeed, in those intended GR scenarios, ambiguous observations can result from incomplete models of the acting agent and the world. Actions

can have different meanings for the actor and the observer, or players can consistently get the most challenging words to guess.

The problem becomes more challenging when the actor is not aware of the observer, and, therefore, it does not help her intentionally. Consider the grid in Figure 3.1(a) and imagine that you observe an agent (represented by a dot) shifting to the left. Assuming full observability and an optimal agent that only moves in one of the four cardinal directions, can you conclude its goal is $G1$? There may be many more variables to consider: is the agent a robot or a human? Is it summer or freezing winter? Common to both scenarios is that the dynamics of the world may affect the action outcomes. Previous work within the GRD framework fails to capture this information, so it will conclude that going to the left is not an ambiguous action. A design based on this result might not be useful in these cases.

(a) (b)

Figure 3.1: Example of GR. (a) A simplified scenario with a dot-agent moving to the left (b) Knowledge of the type of agent (human or robot) and dynamics of the world (slippery ground)[2]

[2]Ice walking in Beijing. Attribution: The Erica Chang. Changes: Circular mask applied.

In this chapter, we model the GRD problem for environments that cause stochastic agent action outcomes. We study scenarios where agents reach a state different from the one intended due to, for instance, slippery or malfunction. Our system might consider that observing the dot-agent moving to the left of its original position is ambiguous and, hence, the design decisions could change. Since our intention on generalizing the GRD framework

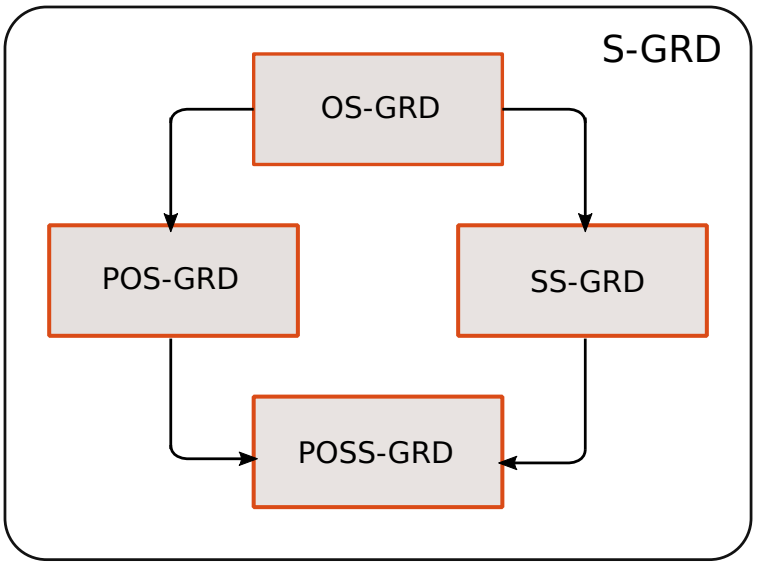


Figure 3.2: S-GRD: Model Structure

stems from its applicability to physical world scenarios, it is only logical to consider partial observability and suboptimal agents. The models we propose observe *four* main assumptions; the agent has full observability, shares its model with the observer, its action outcomes are stochastic, and is unaware of any observer. Figure 3.2 shows a diagram of the proposed models where each block has one characteristic relaxed from its parent.

The *Optimal S-GR*D (OS-GRD) is the most basic S-GRD model; here, the sensor system provides reliable observations of states and actions of the acting agent, and agents always select the optimal policy. The *Partially-Observable* S-GRD (POS-GRD) accounts for partial observability (of an observer) in stochastic environments. POS-GRD assumes unobservable actions and undistinguishable contiguous states. The *Suboptimal S-GRD* (SS-GRD) relaxes the optimality assumption and considers boundedly rational agents that might follow suboptimal policies. Finally, the POSS-GRD model merges POS-GRD and SS-GRD.

Below we describe the reasoning behind the assumptions for partially-observable and suboptimal settings.

Partially-observable settings: The ability to recognize an agent's goal depends, to a large extent, on the ability of an observer to monitor agent behavior. Implicitly, the quality of the observations depends on the sensor resolution.

Consider a robot tasked to deliver a parcel to a monitored area where a tracking system detects the robot's current position. Depending on the sensors' resolution, the robot's exact coordinates may not be available, just an approximation. Further, note that we do not observe the robot's actions but its states (or action outcomes) in reality. To accurately recognize the robot's goal, a GR system should consider that since actions are not perceived, *two or more actions could generate the same observation*. For instance, any turn observed could be either intentional or a result of slippery wheels.

While in deterministic scenarios, it is possible to infer actions from states and vice versa, it is not feasible in stochastic cases. Even assuming a sensor with excellent resolution, we are still limited to observe the outcomes instead of the actions. Therefore, we envision an environment where an observer may not distinguish among several nearby states. In such a setting, changes in observations are the only indication of activities performed by an agent.

Suboptimal settings: The optimality assumption reduces the complexity of the problem but also its applicability. Indeed, a goal recognizer that accounts for some level of suboptimality deals better with uncertainties (Riabov et al., 2020; Sohrabi et al., 2016). Autonomous agents could execute suboptimal policies for many reasons, including resource limitations or unexpected environmental changes. In our delivery robot example, holes in the ground or people walking nearby could require a trajectory change. Since S-GRD is an offline problem, we should anticipate some degree of suboptimality. Our suboptimal settings assume a boundedly rational agent where the number of suboptimal actions is bounded.

The rest of this chapter is organized as follows. Section 3.1 defines a generalized model for Stochastic Goal Recognition Design (S-GRD) problems. Consistent with the GRD case, we consider the GR model, the design model, and the evaluation measure as well as the general objective of S-GRD as an optimization problem. Section 3.2 discusses differences with the most recent GRD model.

3.1 Model

In this section, we describe a generalized S-GRD model that accounts for an *environment* that causes stochastic action outcomes, the *observer*'s capability of perceiving the actor's behavior, and the degree of *agent*'s suboptimality. We model an S-GRD problem with two elements: the *initial GR* model to be analyzed and the *design* model that finds the modifications to the original GR setting that reduce ambiguity. We formulate each component separately before defining the S-GRD problem formally.

3.1.1 Stochastic Goal Recognition (Stochastic GR)

In general, a GR setting defines how actions work in an environment where an agent acts while being observed. The agent generates a sequence of observations during its quest for one goal from a set of candidate goals. In the GRD case, the agent does not act and, therefore, it does not emit observations. Hence, we need to account for *all* possible policies and observations that an agent could take and generate. Additionally, in the S-GRD problem, the challenge is to handle loops produced by revisited states under the same policy. For instance, someone walking on ice could either move for a while but fall and end up in the same original place or continuously slip and not advance at all. If an observer gains some information from the former situation, the goal recognizer needs to differentiate between the initial and last state (even if it corresponds to the same position).

Following the trend of work from Ramírez and Geffner, 2009 and Baker, R. Saxe, et al., 2009 (discussed in the literature review Subsection 2.3.1, p. 26), we use planning models to define the rules that govern actions applicability.

Definition 1. *A stochastic goal recognition problem P is a tuple $P = \langle M, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$, where:*

- $M = \langle \mathbf{S}, s_0, \mathbf{A}, \mathcal{T}, \mathcal{C} \rangle$ *is an SSP-MDP (Subsection 2.2.1, p. 15) without a goal. The four first elements model the world mechanics, and the cost function $\mathcal{C} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to \mathbb{R}^+$ specifies the agent's cost $\mathcal{C}(s, a, s')$ of taking action a at state s and arriving to state s'.*
- $\mathbf{G}$ *is a set of candidate goals, that is, $\forall g \in \mathbf{G}$, $\mathbf{G} \subseteq \mathbf{S}$; g is a possible goal of the agent.*
- $k \geq 0$ *is the degree of granted suboptimality, with $k = 0$ representing optimal agents. In this thesis, k denotes the number of suboptimal actions allowed.*
- $\mathcal{N} : \mathbf{S} \to \mathbf{S}$ *is a sensor function that defines the observer's degree of observability. For partially-observable models, each state s is associated with an observation $\mathcal{N}(s)$, which we refer as the* "projected observation" *of s. The set $\mathbf{S}$ is partitioned into observation sets $\mathbf{O}_1, ..., \mathbf{O}_n$ such that $\forall s, s' : \mathcal{N}(s) = \mathcal{N}(s') \iff \exists i : s, s' \in \mathbf{O}_i$. For fully-observable models, $\mathcal{N}$ is an identity function.*
- $\mathcal{C}_o : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to \mathbb{R}^+$ *is the observer's cost function that assigns a potentially different cost to each agent's action.*

The model admits different cost functions for the agent and the observer, considering their preferences without affecting each other. For example, consider two ambiguous policies with

the same cost for the agent but with a different number of actions per trajectory; the observer should prefer to observe the shorter one, i.e., the one with fewer actions. In this thesis, we assigned a cost of 1 to all actions in both cases.

3.1.2 Design Model

The design model describes the characteristics of applicable modifications.

Definition 2. *A design model in S-GRD is a tuple* $\mathcal{D} = \langle \mathcal{M}, \delta, \phi, \mathcal{C}_m, \mu \rangle$ *where:*

- $\mathcal{M}$ *is a finite set of applicable modifications. A modification sequence is an ordered set of modifications* $\vec{m} = \langle m_1, m_2, \ldots, m_n \rangle$ *with* $m_i \in \mathcal{M}$*. We refer to* $\vec{\mathcal{M}}$ *as the set of all those sequences.*
- $\delta : \mathcal{M} \times P \to P$ *is a modification function, specifying the effect of modifications on the stochastic GR model.*
- $\phi : \vec{\mathcal{M}} \times P \to \{\bot, \top\}$ *is a constraint function that specifies the allowable modification sequences.*
- $\mathcal{C}_m : \mathcal{M} \to \mathbb{R}^+$ *defines the cost* $\mathcal{C}_m(m)$ *to apply modification* $m \in \mathcal{M}$ *to a stochastic GR model.*
- μ *is a user-defined parameter that limits the size of allowed modification sequences.*

In this thesis, we use $\mathcal{C}_m(m) = 1$ for all proposed modifications. We focus on modifications that are *budget preserving*, limiting the modification cost incurred, and *expected cost preserving*,

restricting a modification not to increase agent's or observer's costs. The constraint function handles both restrictions.

The design model allows the application of changes in a principled way. Definition 3 presents the resultant model after the correct application of a sequence of modifications.

Definition 3. *Given a stochastic goal recognition model P and a modification sequence $\vec{m} \in \vec{\mathcal{M}}$ such that $\vec{m} = \langle m_1, m_2, \dots m_n \rangle$; $m \in \mathcal{M}$; and $\phi(\vec{m}) = \top$; the sequence $\vec{m}$ applied to P gives a new stochastic GR model $P^{\vec{m}} = \delta(m_n, \dots, \delta(m_1, P))$.*

The different ways to modify a stochastic GR model depend on the settings and context. We apply two types of modifications to S-GRD. The first, *action removal* defined as by Keren, A. Gal, et al., 2014, removes an action from the set of applicable actions. The second, *sensor refinement*, allowing one to distinguish between states previously mapped to the same observation (Definition 6).

Definition 4. *A modification m is an* action removal modification *if for any stochastic GR model $P = \langle \langle \mathbf{S}, s_0, \mathbf{A}, \mathcal{T}, \mathcal{C} \rangle, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$, $P^m = \langle \langle \mathbf{S}, s_0, \mathbf{A}, \mathcal{T}^m, \mathcal{C} \rangle, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$ is identical to P except that $\exists \mathbf{A}' \subseteq \mathbf{A} \wedge \mathbf{A}' \neq \emptyset, \forall a \in \mathbf{A}', s, s' \in \mathbf{S} : \mathcal{T}(s, a, s') > 0 \implies \mathcal{T}^m(s, a, s') = 0$.*

Definition 5. *A sensor model $\mathcal{N}'$ is a* refinement *of sensor model $\mathcal{N}$ if $\forall s_i, s_j : \mathcal{N}'(s_i) = \mathcal{N}'(s_j) \implies \mathcal{N}(s_i) = \mathcal{N}(s_j)$ (but not necessarily vice versa).*

Let P^m represent the model that results from applying m to P and let $\mathcal{N}^m$ and $\mathcal{N}$ denote the sensor models of P^m and P, respectively. We define sensor refinement as follows.

Definition 6. *A modification m is a* state sensor refinement modification *if for any partially-observable GR model with stochastic action outcomes $P = \langle\langle \mathbf{S}, s_0, \mathbf{A}, \mathcal{T}, \mathcal{C}\rangle, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o\rangle$, $P^m = \langle\langle \mathbf{S}, s_0, \mathbf{A}, \mathcal{T}, \mathcal{C}\rangle, \mathbf{G}, k, \mathcal{N}^m, \mathcal{C}_o\rangle$ is identical to P except that $\mathcal{N}^m$ is a refinement of $\mathcal{N}$.*

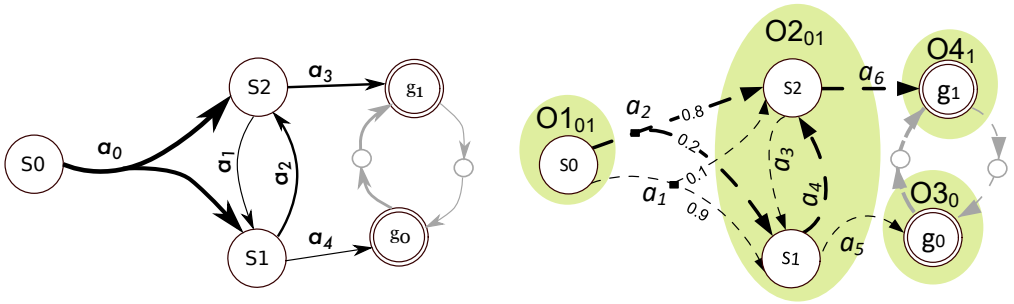
Note that as opposed to the sensor refinement suggested by Keren, A. Gal, et al. (2016b), where the sensor model is defined over tokens emitted by performed actions, the sensor refinement defined here applies to settings where the state of the agent may be only partially observed and the observer has a way to improve its observability by sensing features of the environment.

3.1.3 Evaluating the Goal Recognition Problem

In GRD problems, design optimization serves to minimize ambiguous paths and, consequently, to facilitate GR. Therefore, the approach requires a measure or criterion to assess the difficulty of performing GR in a given model, i.e., environment plus acting agent. The measure used is called the worst-case distinctiveness (*wcd*) (described in Section 2.5, p. 33). In this subsection, we redefine *wcd* for S-GRD problems.

Example 1. *To facilitate the understanding of some definitions, we use Figure 3.3 to illustrate examples of fully-observable (a) and partially-observable (b) S-GRD settings with states (annotated nodes), actions (annotated edges), and observations (annotated shaded areas). Double-lined circles represent goals. A multi-head arrow marks stochastic actions, dashed edges denote unobservable actions, gray edges mark suboptimal actions, and all actions*

have a cost of 1. The thickness of arrows marks optimal actions for a specific goal. Thin arrows represent intended goal g_0, thicker arrows represent intended goal g_1, and the thickest (action a_0 in Figure 3.3(a)) is common for both.



(a) (b)

Figure 3.3: S-GRD Models. (a) OS-GRD considers only black arrows and SS-GRD all. (b) POS-GRD considers only black arrows but POSS-GRD includes all.

Worst-Case Distinctiveness (*wcd*)

Intuitively, the *wcd* (defined for GRD) is the longest sequence of actions that an agent can execute before it reveals its goal. We redefine the *wcd* for S-GRDs taking into account two aspects. First, since transitions are stochastic, the number of actions or, equivalently, the cost incurred can be measured either in the worst-case maximum or in the expectation. The worst-case maximum may be infinite (e.g., when the agent can transition back to its previous state and potentially get stuck in a loop), which prohibits meaningful comparisons. Therefore, we choose to measure the cost in the expectation and use the worst-case distinctiveness (*wcd*) as this measure. Second, note that contrary to Keren, A. Gal, et al., 2019, we propose that

the cost used should be the cost for the observer ($\mathcal{C}_o$ in Definition 1) as the optimization is for the observer's benefit (we provide more details in Section 3.2, p. 57). Hence, at a high level, the *wcd* for S-GRD settings corresponds to the highest expected cost or penalty an observer could experience while the acting agent does not reveal its goal.

Definition 7. *Given a stochastic GR model $P = \langle M, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$, the agent's strategies are the set of all policies Π_g^k of MDP M for goal $g \in \mathbf{G}$ within the limits imposed by k.*

Definition 8. *Given the agent's strategies Π_g^k for goal $g \in \mathbf{G}$, the set $\Pi_{\mathbf{G}} = \bigcup_{g \in \mathbf{G}} \Pi_g^k$ is the set of all legal policies of P for all possible goals.*

In other words, the set of *legal policies* contains *every* policy that an agent can execute for every candidate goal according to the limitations imposed by its model. Models for optimal agents will define $k = 0$. Figure 3.3 represents optimal actions in black arrows; actions $a0, a2$, and $a3$ in Figure 3.3(a) form a *legal policy* in Π_{g1}^0.

We next define *wcd* for S-GRD problems, starting with partial policy containment. The following definitions are constrained to legal policies.

Definition 9. *A partial policy $\hat{\pi}$ is contained in a policy $\pi \in \Pi_{\mathbf{G}}$ (marked $\hat{\pi} \subseteq \pi$) if $\mathbf{S}_{\hat{\pi}} \subseteq \mathbf{S}_{\pi}$ and $\forall s \in \mathbf{S}_{\hat{\pi}}, \hat{\pi}(s) = \pi(s)$.*

$\mathbf{S}_{\pi}$ represent the set of reachable states while following policy π.

Definition 10. *A partial policy $\hat{\pi}$* satisfies *a goal g if $\exists \pi \in \Pi_g^k$ s.t. $\hat{\pi} \subseteq \pi$. The set of goals $\mathbf{G}' \subseteq \mathbf{G}$ satisfied by a partial policy $\hat{\pi}$ is marked by $\mathbf{G}'(\hat{\pi})$.*

A *trajectory* $\vec{\tau} = \langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$ is a realization of an agent's policy, denoted by alternating actions an agent performs and reached states. We use trajectory indices to relate an action with its resulting state. A trajectory $\vec{\tau}$ is *feasible* if $\forall i : T(s_i, a_{i+1}, s_{i+1}) > 0$. We next relate trajectories and goals.

Definition 11. *A feasible trajectory* $\vec{\tau} = \langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$ satisfies *a possible goal* g *if* $\exists \pi_g \in \Pi_{\mathbf{G}}(g)$ *s.t.* $\forall i \in \{0 \ldots n-1\}$, $s_i \in S_{\pi_g}$ *and* $a_{i+1} = \pi_g(s_i)$.

Definition 12. *The* observable projection *of a trajectory* $\vec{\tau} = \langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$ *for OS-GRD and SS-GRD settings is:* $obs(\vec{\tau}) = \vec{\tau} = \langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$

That is, the observer is able to capture *all* information about agent's visited states and executed actions.

Partial observability in POS-GRD and POSS-GRD cases is materialized by limiting the observer to only see changes in emitted observations. To model this, we define an observable projection of a trajectory, where $\cdot$ denotes the concatenation of two sequences.

Definition 13. *The* observable projection *of a trajectory* $\vec{\tau} = \langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$ *is:* $obs(\vec{\tau}) = obs(\langle s_0 \ldots s_n \rangle) =$

$$\begin{cases} \langle \mathcal{N}(s_0) \rangle & n{=}0 \quad (3.1) \\ obs(\langle s_0 \ldots s_{n-1} \rangle) & n > 0 \wedge \mathcal{N}(s_{n-1}) = \mathcal{N}(s_n) \quad (3.2) \\ obs(\langle s_0 \ldots s_{n-1} \rangle) \cdot \langle \mathcal{N}(s_n) \rangle & n > 0 \wedge \mathcal{N}(s_{n-1}) \neq \mathcal{N}(s_n) \quad (3.3) \end{cases}$$

For example, consider Figure 3.3(b). The *observable projection* of trajectory $\vec{\tau} = \langle S0, a_1, S2, a_3, S1, a_5, g_0 \rangle$ is $obs(\vec{\tau}) = \langle \mathcal{N}(S0), \mathcal{N}(S1), \mathcal{N}(g_0) \rangle = \langle O1, O2, O3 \rangle$.

Definition 14. *An observable projection o* satisfies *a possible goal g if there exists a feasible trajectory $\vec{\tau}$ that satisfies g and $o = obs(\vec{\tau})$.*

We denote by $\mathbf{G}(\hat{\pi})$ and $\mathbf{G}(o)$ the set of goals satisfied by a partial policy $\hat{\pi}$ and observation sequence o, respectively.

Definition 15. *The set of goals satisfied by the observed sequence of trajectory $\vec{\tau} = \langle s_0, \pi(s_0), s_1, \ldots, s_n \rangle$ is:*

$$\mathbf{G}(obs(\langle s_0, \pi(s_0), \ldots, s_n \rangle)) =$$

$$\begin{cases} \mathbf{G} & i{=}0 \quad (3.4) \\ \mathbf{G}(obs(\langle s_0, \pi(s_0), \ldots, s_{i-1} \rangle)) & 0 < i \leq n \wedge \mathcal{N}(s_{i-1}) = \mathcal{N}(s_i) \quad (3.5) \\ \mathbf{G}(obs(\langle s_0, \pi(s_0), \ldots, s_{i-1} \rangle)) \cap \mathbf{G}' & 0 < i \leq n \wedge \mathcal{N}(s_{i-1}) = \mathcal{N}(s_i) \quad (3.6) \end{cases}$$

where:

$$\mathbf{G}' = \mathbf{G}(\pi(s_{i-1})) \qquad \textit{for FO models} \quad (3.7)$$

$$\mathbf{G}' = \bigcup_{\pi(s) | \exists s' : \mathcal{T}(s, \pi(s), s') > 0 \wedge \mathcal{N}(s) = \mathcal{N}(s_{i-1}) \wedge \mathcal{N}(s') = \mathcal{N}(s_i)} \mathbf{G}(\pi(s)) \qquad \textit{for PO models.} \quad (3.8)$$

In a fully-observable setting, actions are observable and a goal g becomes no longer possible if the executed action a does not satisfy g. In the partially-observable case, actions cannot be observed, thus g is discarded only if there exist an action a that can be executed at any state projecting $\mathcal{N}(s_{i-1})$, transitions to state s' with projected observation equal to $\mathcal{N}(s_i)$, and does not satisfy goal g.

To illustrate, consider Figure 3.3(b) (p. 49), where $obs(\langle S0\rangle) = \mathcal{N}(S0) = O1$ and $\mathbf{G}(obs(\langle S0\rangle)) = \{g_0, g_1\} = \mathbf{G}$. In a fully-observable case, $\mathbf{G}(obs(\langle S0, a_2, S2\rangle)) = \{g_1\}$. However, in a partially-observable setting, $\mathbf{G}(obs(\langle S0, a_2, S2\rangle)) = \{g_0, g_1\} = \big(\mathbf{G}(\langle obs(S0)\rangle = \mathbf{G}\big) \cap \big(\mathbf{G}(a_1) = \{g_0\} \cup \mathbf{G}(a_2) = \{g_1\}\big)$ because $\mathbf{G}(obs(\langle S0, a_2.S2\rangle)) = \mathbf{G}(obs(\langle S0, a_1.S2\rangle)) = \langle \mathcal{N}(S0), \mathcal{N}(S2)\rangle = \langle \mathcal{N}(S0), \mathcal{N}(S1)\rangle$.

Finally, we are ready to define non-distinctiveness.

Definition 16. *A partial policy $\hat{\pi}$ (respectively trajectory $\vec{\tau}$ or observation sequence o) is* non-distinctive *if it satisfies more than one goal ($|\mathbf{G}(\hat{\pi})| > 1$ (respectively $|\mathbf{G}(obs(\vec{\tau}))| > 1$)). If a partial policy (respectively, observation sequence) is not non-distinctive we say it is* distinctive. *Note that an empty policy $\hat{\pi}$ is non-distinctive as it is legal for all candidate goals.*

The core of our analysis consists of identifying and characterizing non-distinctive (ambiguous) behavior possible in the model. For this purpose we will use the following observations.

Lemma 1. *Given two partial policies $\hat{\pi}$ and $\hat{\pi}'$, if $\hat{\pi} \subseteq \hat{\pi}'$ and $\hat{\pi}$ is distinctive, then $\hat{\pi}'$ is distinctive.*

Proof. Assume to the contrary that $\hat{\pi} \subseteq \hat{\pi}'$, $\hat{\pi}$ is distinctive but $\hat{\pi}'$ is non-distinctive. Since $\hat{\pi}'$ is non-distinctive $\exists g_0, g_1$ that are both satisfied by $\hat{\pi}'$ (Definition 16). This in turn means that $\exists \pi_{g_0} \in \Pi^k_{g_0}$ and $\pi_{g_1} \in \Pi^k_{g_1}$ s.t. $\hat{\pi}' \subseteq \pi_{g_0}$ and $\hat{\pi}' \subseteq \pi_{g_1}$ (Definition 10). According to the containment relation described in Definition 9, the fact $\hat{\pi}'$ is contained in both π_{g_0} and π_{g_1} means that $\forall s \in S_{\hat{\pi}'}\ \hat{\pi}'(s) = \pi_t(s) = \pi_{t'}(s)$. Since $\hat{\pi} \subseteq \hat{\pi}'$ then $\forall s \in S_{\hat{\pi}'}$, $s \in S_{\hat{\pi}}$ and $\hat{\pi}(s) = \hat{\pi}'(s) = \pi_t(s) = \pi_{t'}(s)$. Thus, $\hat{\pi} \subseteq \pi_{t_0}$ and $\hat{\pi} \subseteq \pi_{t_1}$ thus satisfying both t_0 and t_1 and contradicting our assumption that $\hat{\pi}$ is distinctive. ■

Lemma 2. *Given a distinctive trajectory $\vec{\tau} = \langle s_0, a_0, s_1, a_1 \ldots, s_n, a_n, s_{n+1} \rangle$, any trajectory for which $\vec{\tau}$ is a prefix is distinctive .*

Proof. Assume by contradiction that $\exists \vec{\tau}' = \langle s_0, a_0, s_1, a_1 ..., s_n, a_n, s_{n+1}, ..., a_m, s_{m+1} \rangle$ $(n \leq m)$ for which $\vec{\tau}$ is a prefix that is non-distinctive. According to Definition 16 since $\vec{\tau}'$ is non distinctive, then $\exists g, g' \in \mathbf{G}$ $(g \neq g')$ both satisfied by $\vec{\tau}'$. This, according to Definition 11, means that $\exists \hat{\pi}_g \in \hat{\Pi}^k_g$ and $\hat{\pi}_{g'} \in \hat{\Pi}^k_{g'}$ s.t. $\forall i \leq m$, $s_i \in S_{\hat{\pi}_g}$, $s_i \in S_{\hat{\pi}_{g'}}$ and $a_i = \hat{\pi}_g(s_i) = \hat{\pi}_{g'}(s_i)$. In particular, this is true for any $i \leq n$, contradicting our assumption that $\vec{\tau}$ is distinctive. ■

We now (re)define the worst case distinctiveness (*wcd*) for models with stochastic actions.

Distinctiveness cost is the total cost of the maximal prefix of a trajectory whose observable projection is non-distinctive. A partial policy $\hat{\pi}$ induces a *distribution* on trajectories, in which the probability of trajectory $\vec{\tau} = \langle s_0, a_1, s_1 \ldots, a_n, s_n \rangle$ is $P_{\hat{\pi}}(\vec{\tau}) = \Pi_{i=1}^n \mathcal{I}_{\hat{\pi}(s_{i-1})=a_i} P(s_i | s_{i-1}, a_i)$, where $\mathcal{I}$ is the indicator function that takes value 1 when $\hat{\pi}(s_{i-1}) = a_i$ and 0 otherwise.

Therefore, the *wcd* of a stochastic GR model is the (legal) partial policy with the maximal expected distinctiveness.

Definition 17. *The* distinctiveness cost $DC(\vec{\tau})$ *of a trajectory* $\vec{\tau} = \langle s_0, a_1, s_1 \ldots, a_n, s_n \rangle$ *is*

$$\max_{i \in \{0 \ldots n\} \ s.t. \ |\mathbf{G}(obs(\langle s_0, \ldots, a_i, s_i \rangle))| > 1} \sum_{j=1}^{i} \mathcal{C}_o(s_{j-1}, a_j, s_j) \tag{3.9}$$

The expected distinctiveness $ED(\hat{\pi})$ *of a partial policy* $\hat{\pi}$ *is the expected distinctiveness cost of its trajectories,* $\sum_{\vec{\tau}} P_{\hat{\pi}}(\vec{\tau}) DC(\vec{\tau})$.

The worst case distinctiveness *of a stochastic GR problem* P *is:*

$$\mathrm{wcd}(P) = \max_{\hat{\pi} \in \hat{\Pi}_{\mathbf{G}}} ED(\hat{\pi}) \ \textit{where} \ \hat{\Pi}_{\mathbf{G}} = \bigcup_{g \in \mathbf{G}} \hat{\Pi}_g^k \tag{3.10}$$

Two notes are in order here. First, for an empty trajectory ($i = 0$), distinctiveness cost is 0. Second, $DC(\vec{\tau})$ and $ED(\hat{\pi})$ are well-defined for proper policies (Section 2.2.1, p. 15).

Example 2. *To illustrate the influence of each setting in the value of* wcd*, consider Figure 3.4 where all action costs are 1. If the states and actions are fully observable, and the agent is optimal,* $\mathrm{wcd} = 0$ *because there is only one optimal policy for each goal (the arrows' thickness distinguishes individual policies). POS-GRD will have* $\mathrm{wcd} = 1.2$ *as it is not possible to know the action taken from* $O1$ *to* $O2$ *nor the executed actions in* $O2$*. The agent reveals its real goal only at its arrival (the observations' subindices mark possible goals), and goal* g_1 *demands the highest expected cost. Note that by Definition 17 (Eq. 3.9), the cost of the last action* (a_6)

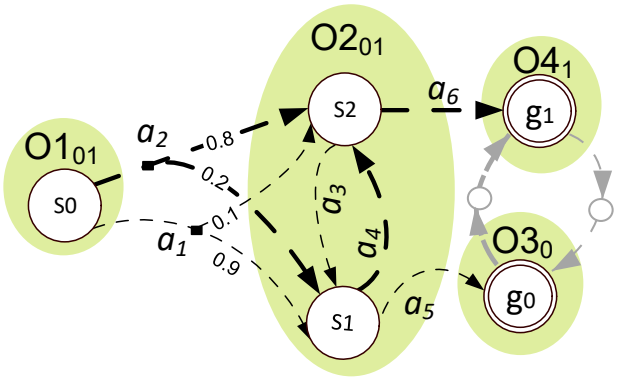


Figure 3.4: Assumptions of S-GRD Models. OS-GRD considers only optimal policies (black edges) and observes all states (white nodes). SS-GRD suboptimal policies (all edges) and full observability (white nodes). POS-GRD considers only optimal policies (black arrows) and obfuscated observations (green shades, dashed arrows). POSS-GRD joins POS-GRD and SS-GRD assumptions.

does not add to the value of wcd. *The value of* wcd *increases to* 2.9 *for SS-GRD models with* $k = 3$ *when the agent chooses actions* a_1, a_4, a_6 *since the only distinctive actions, in this case, are the gray arrows. Likewise, in the case of POSS-GRD with* $k = 3$*, there could be two projected sequences:* $\langle O1, O2, O4, O3\rangle$*, or* $\langle O1, O2, O3, O4\rangle$*. In both cases, the first three observations are non-distinctive and the value* wcd $= 2.9$*. The sequences of observations also show that the distinctiveness of* $O3$ *and* $O4$ depends *on the trajectory.*

3.1.4 Stochastic Goal Recognition Design Model

We model an S-GRD problem with a tuple:

$$T = \langle P_0, \mathcal{D}\rangle \tag{3.11}$$

where P_0 is the *initial stochastic GR* model to be analyzed and $\mathcal{D}$ is the *design model*, which specifies the rules to generate alternative stochastic GR models P by applying modification sequences to P_0.

The objective of a S-GRD problem is to find a sequence of modifications $\vec{m} = \langle m_1 \dots m_n \rangle$, such that $\vec{m}$ is feasible (i.e., $\phi(\vec{m}) = \top$), and which minimizes the *wcd* of the resulting model $P_0^{\Delta} := (P_0^{m_1})^{\dots m_n}$:

$$\vec{m}^o = \underset{\vec{m} \in \vec{\mathcal{M}}}{\operatorname{argmin}} f(P_0) \tag{3.12}$$

$$\text{s.t. } \phi(\vec{m}) = \top \tag{3.13}$$

$$\mid \vec{m} \mid \leq \mu \tag{3.14}$$

where f is the objective criterion or measure chosen for the design optimization. In this thesis, $f(P_0) = wcd(P_0)$. Other terms defined in Section 3.1 maintain their meaning.

3.2 Discussion

In this section, we will compare the equivalences and differences of the general model for GRD in deterministic environments (Keren, A. Gal, et al., 2019) and the S-GRD model proposed in previous section.

Both models have two main components, one for the GR and one for the design problem. Keren, A. Gal, et al., 2019 define a GR problem as $P_D = \langle \mathcal{P}, \mathbf{G}, leg, O, S \rangle$, where:

- $\mathcal{P}$ is a classical planning problem without goals.
- $\mathbf{G}$ is the set of possible goals.
- $leg : \vec{\Pi} \times \mathbf{G} \rightarrow \{0, 1\}$ is an indicator that specifies the legal paths to each possible goal.
- O is a set of observation tokens.
- $S : \mathbf{A} \rightarrow 2^O \setminus \emptyset$ is a sensor model mapping actions to observation tokens.

In both cases, the GR model contains the specification of the environment dynamics and a set of possible goals. There is some discrepancy between models due to differences in the specific assumptions of suboptimality and partial observability. Keren, A. Gal, et al., 2019 have an indicator to validate the set of allowed or *legal* paths instead of the degree of suboptimality (k) for the S-GRD case. To model partial observability, Keren, A. Gal, et al., 2019 define a set of observation tokens representing the set of possible projected observations and a sensor model mapping actions to observation tokens. Our model captures partial observability using the sensor function $\mathcal{N}$. Additionally, S-GRD assumes the cost of observing actions could be different from the execution cost, while the deterministic model P_D assumes the same cost. The argument is that observer and agent might have unrelated priorities. An agent could use a lot of energy executing one action or execute several actions using the same power in total. However, as long as each action takes the same amount of time (non-durative), an

observer would prefer shorter trajectories (i.e., fewer actions) regardless of the cost they have for the agent. This decision will affect the solution when using costs different to 1, and it also affects the *wcd* definition. S-GRD considers the observer cost function $\mathcal{C}_o$. In contrast, GRD considers the agent cost function $\mathcal{C}$.

The design model for S-GRD has two additional components: the cost per modification and the budget of allowed modifications. Keren, A. Gal, et al., 2019 assume uniform modification cost for clarity. However, they discuss changes in the algorithms to tackle different costs. Regarding the budget, Keren, A. Gal, et al., 2019 model it as part of the constraint indicator function ϕ.

Chapter 4

Worst-Case Distinctiveness (*wcd*)

"Take advantage of the ambiguity in the world.

Look at something and think what else it might be."

– Roger von Oech

The previous chapter described and formally modeled the S-GRD framework. Similar to GRD problems, our solution approach to solve S-GRD problems considers two stages: (1) Computing the measure (*wcd*) to evaluate the initial stochastic GR problem; and (2) Optimizing the design by finding the sequence of modifications that minimize *wcd*. This chapter focuses on the first stage and analyzes properties and challenges specific to each setting. Starting with the Optimal S-GRD version, we gradually relax different assumptions to end with the POSS-GRD variant.

4.1 Optimal S-GRD (OS-GRD)

The intuition behind the *worst case distinctiveness* (*wcd*) is that it measures the longest ambiguous trajectory, i.e., a path with the highest observed cost an agent can take without revealing its goal.

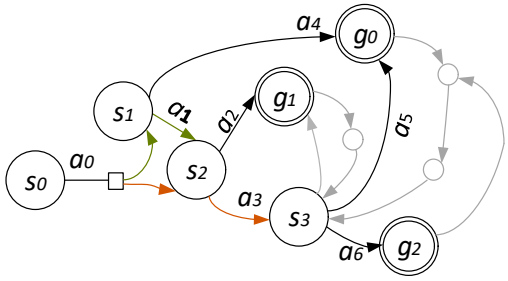


Figure 4.1: OS-GRD example with two possible ambiguous trajectories (green and orange)

Example 3. *Consider Figure 4.1, where the agent starts at state s_0 and has one of three possible goals g_0, g_1, or g_2. All actions have the same cost of 1 for the agent and the observer. All of them are deterministic except for action a_0 out of s_0, which can transition to either s_1 or s_2 with equal probability. In this example, there are two possible trajectories, each with two actions, that the agent can take before revealing its goal with the third action.*

For the first (green) trajectory, starting at s_0, the agent has to take action a_0 as it is the only action available. If it transitions to s_1, then it can take action a_1 to transition to s_2. At this point, its goal can be either g_1 or g_2. From s_2, it reveals its goal to be g_1 if it takes action a_2 and goal g_2 if it takes action a_3. Note that its goal cannot be g_0 as it would otherwise have taken action a_4 instead of a_1 when it was in s_1.

For the second (orange) trajectory, starting at s_0, the agent may transition to s_2 after taking action a_0. Then, it can take action a_3 to transition to s_3, at which point its goal can be either g_0 or g_2. From s_3, it reveals its goal to be g_0 if it takes action a_5 and goal g_2 if it takes action a_6.

Note that the cost of both trajectories is 2 since they both have two actions each. Consequently, the wcd *of this problem should be 2 intuitively.*

4.1.1 Analyzing the GRD Strategy for S-GRD

Keren, A. Gal, et al., 2014 use a pairwise approach to compute *wcd*. They find the largest path common to two goals for all possible combinations of goal pairs, and the maximum value corresponds to the *wcd* of the problem. Since we deal with the expected distinctiveness cost of trajectories, we cannot use the same approach unless all non-distinctive policies have Property 1.

Property 1. *Given a non-distinctive policy $\hat{\pi}$ with trajectories $\vec{\tau}_{\hat{\pi}_i}$.If $\forall \vec{\tau}_{\hat{\pi}_1} \neq \vec{\tau}_{\hat{\pi}_2} : \big(|\mathbf{G}(\vec{\tau}_{\hat{\pi}_1})| \leq |\mathbf{G}(\vec{\tau}_{\hat{\pi}_2})|\big) \rightarrow \big(\mathbf{G}(\vec{\tau}_{\hat{\pi}_1}) \subseteq \mathbf{G}(\vec{\tau}_{\hat{\pi}_2})\big)$, we say that $\hat{\pi}$ satisfies the same subset of goals.*

Theorem 1. *If all non-distinctive policies in a stochastic GR model $P = \langle M, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$ satisfy Property 1, then*

$$\text{wcd}(P) = \max_{g_1, g_2 \in \mathbf{G} | g_1 \neq g_2} \text{wcd}(P') \tag{4.1}$$

where $P' = \langle M, \{g_1, g_2\}, k, \mathcal{N}, \mathcal{C}_o \rangle$.

Proof. By Definition 17 (p. 55), every non-distinctive trajectory has at least two goals. Property 1 guarantees that non-distinctive trajectories for the same number of goals satisfy the same set of goals, and a larger set of goals satisfying any other non-distinctive trajectories are a superset of those goals. Therefore, if a set of goals $\mathbf{G}'$ satisfy a non-distinctive trajectory $\vec{\tau}$ (denoted $\mathbf{G}'(\vec{\tau})$), any combination of pair of goals in $\mathbf{G}'$ satisfy the same trajectory $\vec{\tau}$. Hence, all non-distinctive trajectories that contribute to $wcd(P)$ correspond to non-distinctive trajectories used to compute the largest *wcd* for some pair of goals. ■

Unfortunately, Property 1 does not apply to all scenarios. For instance, in Figure 4.1, the green trajectory satisfies $\mathbf{G}' = \{g_1, g_2\}$ while the orange trajectory satisfies $\mathbf{G}'' = \{g_0, g_2\}$. Since $\mathbf{G}' \neq \mathbf{G}''$, $wcd(P)$ using Eq. 4.1 fails to capture the intuitive trajectories of cost 2. The longest non-distinctive policy for goals $\langle g_0, g_2 \rangle$ has two trajectories $\langle s_0, a_0, s_1 \rangle$ and $\langle s_0, a_0, s_2, a_3, s_3 \rangle$, and the expected cost of this policy prefix is 1.5 (= 0.5 *1+0.5*(1 + 1)). The expected cost for goals $\langle g_0, g_1 \rangle$ is 1 (with trajectories $\langle s_0, a_0, s_1 \rangle$ and $\langle s_0, a_0, s_2 \rangle$). Lastly, the longest ambiguous trajectories for goals $\langle g_1, g_2 \rangle$ are $\langle s_0, a_0, s_1, a_1, s_3 \rangle$ and $\langle s_0, a_0, s_2, a_3, s_3 \rangle$ with an expected cost of 1.5. The *wcd* of this problem using Equation 4.1 is thus 1.5, the largest expected cost over all *pairs* of goals.

4.1.2 The All-Goals Strategy

We now propose a new method to compute *wcd* that captures the above scenario and is more consistent with its intuitive definition. This method considers all goals at the same time

and finds the largest expected cost among *all possible non-distinctive trajectories*, which is equivalent to the *wcd* when all trajectories of a policy with maximal expected distinctiveness (Definition 17, p. 55) have distinctiveness costs larger or equal than trajectories of any other policy [3].

In addition to the need to consider all goals simultaneously instead of pairs of goals, we also make another key observation: that the set of possible goals for a particular state can differ based on the observed path to reach that state. Using Figure 4.1 as an example again, if the agent arrives at state s_3 through the orange trajectory, then its goal is either g_0 or g_2. However, if it arrives at state s_3 through trajectory $\langle s_0, a_0, s_1, a_1, s_2, a_3, s_3 \rangle$, then its goal is definitely g_2. This observation causes a challenge. Unlike the previous method, the set of possible goals of the agent in this new approach is no longer Markovian as it *depends on the entire history of states visited*. We address this challenge by incorporating the history into the state definition and model the problem using augmented MDPs instead of regular MDPs.

4.1.3 Augmented MDP

An augmented MDP adds a Boolean variable pos_g for each possible goal g, to keep track of whether g has been eliminated as a possible goal or not. The terminal states of this MDP are those where all successors have less than two possible goals, with transitions defined according to the original MDP. Formally, given a stochastic GR model

[3] Subsection 4.1.5 (p. 74) presents a method to compute *wcd* for all cases

$P = \langle M = \langle \mathbf{S}, s_0, \mathbf{A}, \mathcal{T}, \mathcal{C} \rangle, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$, with $k = 0$ for OS-GRD, the augmented MDP $\Pi_{aug} = \langle \mathbf{S}', s'_0, \mathbf{A}', \mathcal{T}', \mathcal{C}'_o, \mathbf{G}' \rangle$ is defined as follows:

- $\mathbf{S}' = \mathbf{S} \times \{T, F\}^{|\mathbf{G}|}$: for each $s \in \mathbf{S}$ we create $2^{|\mathbf{G}|}$ possible states, corresponding to all subsets of possible goals.

- $s'_0 = s_0 \cdot \langle T \dots T \rangle$: initially all goals are possible.

- $\mathbf{A}' = \mathbf{A}$, as action labels are the same.

- $\mathcal{T}'(s \cdot \langle pos_1 \dots pos_n \rangle, a, s' \cdot \langle pos'_1 \dots pos'_n \rangle) =$
$$\begin{cases} \mathcal{T}(s, a, s') & (\exists i \neq j : pos'_i = pos'_j = T) \wedge \\ & \forall i \in \{1 \dots n\} pos'_i = \big(pos_i \wedge (\exists \pi \in \Pi^0_{g_i} \mid \pi(s) = a)\big) \\ 0 & \text{otherwise} \end{cases}$$
When executing action a in state s, the flag pos'_i indicating whether goal g_i is possible becomes false if a is not an optimal policy with respect to g_i (or remains false if it was already false). States with less than two possible goals are no longer reachable.

- $\mathcal{C}'(s \cdot \langle pos_1 \dots pos_n \rangle, a, s' \cdot \langle pos'_1 \dots pos'_n \rangle) = \mathcal{C}_o(s, a, s')$: we wish to find policies with maximal cost for the observer.

- $\mathbf{G}' = \big\{ s \cdot \langle pos_1 \dots pos_n \rangle \mid \forall \pi(s) : \pi \in \Pi_{\mathbf{G}} \implies \mathcal{T}'(s \cdot \langle pos_1 \dots pos_n \rangle, \pi(s), s' \cdot \langle pos'_1 \dots pos'_n \rangle) = 0 \big\}$.

  Augmented goal states are those where all original actions from those states will transition to an augmented state with one or no goals.

Lemma 3. *All policies in Π_{aug} from s'_0 to a terminal state are non-distinctive.*

Proof. All states in the original MDP have augmented versions in Π_{aug}, and by the condition of augmented goals, the last reachable terminal states have two or more possible goals. The transition function guarantees that a discarded goal cannot become possible again. Hence, there are not augmented states with less than two goals that are reachable through a legal (optimal) policy , which by Definition 16 makes all policies non-distinctive. ■

Lemma 4. *Let π be a non-distinctive legal policy in the regular MDP. Then $\forall \pi, \exists \hat{\pi} \in \Pi_{aug}$: $V_\pi(s_0) = V_{\hat{\pi}}(s'_0)$ where $V_\pi(s_0), V_{\hat{\pi}}(s'_0)$ are the expected costs of policies π and $\hat{\pi}$ at their starting states.*

Proof. A trajectory $\vec{\tau}_\pi = \langle s_0, a_1, \ldots, a_m, s_m \rangle$ has the same cost of trajectory $\vec{\tau}_{\hat{\pi}} = \langle s'_0, a_1, \ldots, a_m, s'_m \rangle \iff \forall s_i, s'_i : s'_i = s_i \cdot \langle pos_1 \ldots pos_n \rangle$. Let $\vec{\tau}_\pi$ be a valid trajectory in π, i.e., $\forall i \in \{0, \ldots, m\}, s_i \in \mathbf{S} \implies a_i = \pi(s_{i-1})$, then by construction of the augmented MDP and Lemma 3, it is always possible to find $\vec{\tau}_{\hat{\pi}}$ in the augmented MDP. Further, the probability of each trajectory $\vec{\tau}_\pi$ has the same probability of its equivalent $\vec{\tau}_{\hat{\pi}}$. Therefore, policy $\hat{\pi}$ exists in Π_{aug} and has the same expected cost as policy π. ■

Lema 4 ensures that all non-distinctive policies of the regular MDP have equivalent policies in the augmented MDP Π_{aug} with the same expected cost.

Theorem 2. *Given the set $\mathcal{F}$ of all trajectories in the augmented MDP, if $\mathcal{F}\widehat{\pi}_{\mathbf{max}}$ is the set of valid trajectories of $\hat{\pi}_{max} \in \Pi_{aug}$, where $\hat{\pi}_{max}$ is maximal, and if $\forall \vec{\tau} \in \mathcal{F} \setminus \mathcal{F}\widehat{\pi}_{\mathbf{max}}, \vec{\tau}_{\hat{\pi}_{max}} \in$*

$\mathcal{T}\widehat{\pi}_{\mathbf{max}} : Cost(\vec{\tau}) \leq Cost(\vec{\tau}_{\hat{\pi}_{max}})$*, then the expected cost* $V_{\hat{\pi}}(s'_0)$ *of* $\hat{\pi}_{max} \in \Pi_{aug}$ *at the starting state* s'_0*, corresponds to the* wcd *of the original GR model* P*.*

Proof. By Lemma 3, any policy in the augmented MDP is non-distinctive. Hence, the policy $\hat{\pi}_{max} \in \Pi_{aug}$ with the maximum cost is also non-distinctive. Lemma 4 implies that there is a policy $\hat{\pi}' \in \Pi_{aug}$ with the same expected cost as the non-distinctive policy with the highest cost in the regular MDP M. We need to show that $V_{\hat{\pi}_{max}}(s_0) = V_{\hat{\pi}'}(s_0)$. First, $V_{\hat{\pi}_{max}}(s'_0) \not< V_{\hat{\pi}'}(s'_0)$ as both policies exist in Π_{aug}, and $V_{\hat{\pi}_{max}}(s'_0)$ is already the maximum. Since $V_{\hat{\pi}'}(s'_0)$ is equal to the highest non-distinctive cost in M and there are not trajectories with costs larger than any trajectory of π_{max}, π_{max} would need to have trajectories or transition probabilities that do not exist in M. However, the augmented transition function limits the reachable states and actions to those corresponding to valid non-distinctive trajectories in M. Therefore, $V_{\hat{\pi}_{max}}(s_0) = V_{\hat{\pi}'}(s_0)$ and $V_{\hat{\pi}_{max}}(s_0) = wcd(P)$ by Definition 17 (Eq. 3.10). ■

4.1.4 Computing *wcd*: Algorithms

Recall that *wcd* is the maximal cost overall legal (optimal in this case) partial policies that aimed at more than a single goal (Definition 17, Eq. 3.10). The number of augmented states is $O(|S| \times 2^{|G|})$, which is exponential in the number of model goals. However, not all augmented states are reachable, which provides us with an opportunity not to generate all when computing *wcd*. Next, we offer a method to precisely generate the augmented states needed for *wcd* computation.

The proposed method has the following four steps: (1) Find all optimal policies; (2) Construct the augmented MDP for reachable states; and (3) Solve this augmented MDP to compute the *wcd*. We next provide details of each of these steps.

Finding Optimal Policies

To identify $\Pi_{\mathbf{G}}$, we separately solve an MDP for each goal. Using $V^*(s_0)$, the optimal expected cost at the starting state, we identify all optimal policies per goal. Figure 4.2 shows one optimal policy per goal, marked in black, for the original example shown in Figure 4.1 (p. 61).

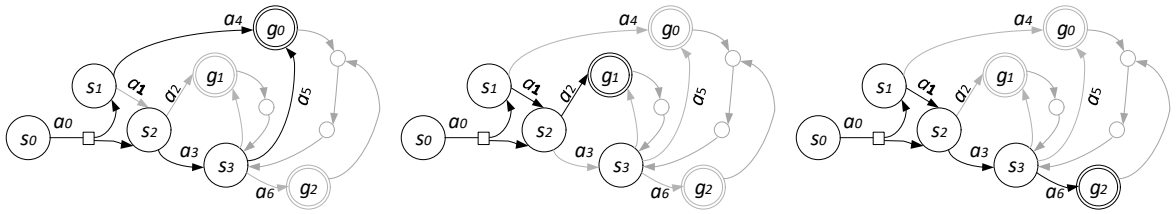


Figure 4.2: Legal Policies for OS-GRD (marked in black). (a) Optimal policy for goal g_0. (b) Optimal policy for goal g_1. (c) Optimal policy for goal g_2.

Constructing Augmented MDP for OS-GRD

To generate the reachable augmented state space and the corresponding augmented transition function, we use an iterative 3-step procedure: (1) Augment the initial state with all possible goals and add it to a stack; (2) For each immediately connected successor: intersect the goals of the predecessor with the set of goals that satisfy the action; if the resultant intersection

contains more than one goal and the augmented state was not created before, augment the successor state with that set; (3) Update the transition function.

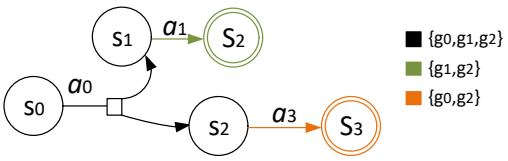


Figure 4.3: Reachable Augmented MDP for the OS-GRD running example.

To illustrate the procedure, consider Figure 4.2. The start sate s_0 is augmented with goals g_0, g_1, and g_2 (Step 1). All actions are also augmented with the set of goals for which they are optimal (Step 2). For example, action a_1 is optimal for goals g_1 and g_2 as shown in Figure 4.2(b) and (c). States s_1, and s_2 are generated and since action a_0 and state s_0 are each augmented with all goals, both states are also augmented with all candidate goals. When s_2 is explored, the procedure generates goal g_1 and state s_3; s_3 is augmented with $\{g_0, g_1, g_2\} \cap \{g_0, g_2\} = \{g_0, g_2\}$. Since g_1 is distinctive, it is left unexplored. Exploring s_3 produces goals g_0 and g_2, which are no longer ambiguous, thus they are no further explored. A similar procedure explores and expands state s_1. Every time a state is explored, the augmented transition function is updated (Steps 2 and 3). Figure 4.3 shows the resultant augmented MDP, where the set of possible goals augmenting states and actions are color-coded and terminal states are marked with double circles.

Algorithm 1 presents a pseudocode for constructing an augmented MDP for OS-GRD that follows the 3-step procedure described before. The algorithm receives as input the regular MDP parameters, the set of candidate goals, a set of optimal policies, and a parameter l that

Algorithm 1: AUGMDP-OS-GRD($s_0, \mathbf{S}, \mathcal{T}, \mathbf{G}, l, \Pi_{\mathbf{G}}$)

1 $\mathbf{G}', \mathbf{S}', Stack \leftarrow \emptyset$; $\mathcal{T}' \leftarrow null$
2 $\mathbf{S}' \leftarrow \mathbf{S}' \cup \{s'_0 \leftarrow s_0 \cdot \langle \mathbf{G} \rangle\}$
3 $Stack.push(s'_0)$
4 **while** $Stack \neq \emptyset$ **do**
5 $\quad s' \leftarrow s \cdot \langle \hat{\mathbf{G}} \rangle \leftarrow Stack.pop()$
6 $\quad Goals \leftarrow \mathbf{A}_{temp} \leftarrow \emptyset$
7 $\quad$ **foreach** $\mathcal{T}(s, \pi(s), s_s) > 0$ *s.t.* $|\hat{\mathbf{G}}'(\pi(s))| > 0$ **do**
8 $\quad\quad \mathbf{A}_{temp} \leftarrow \mathbf{A}_{temp} \cup \{\pi(s)\}$
9 $\quad\quad s'_s \leftarrow s_s \cdot \langle \hat{\mathbf{G}} \cap \hat{\mathbf{G}}' \rangle$
10 $\quad\quad$ **if** $|\hat{\mathbf{G}} \cap \hat{\mathbf{G}}'| > l$ **then**
11 $\quad\quad\quad$ **if** $s'_s \notin \mathbf{S}'$ **then** $Stack.push(s'_s)$;
12 $\quad\quad\quad \mathbf{S}' \leftarrow \mathbf{S}' \cup \{s'_s\}$
13 $\quad\quad\quad T'(s', \pi(s), s'_s) \leftarrow T(s, \pi(s), s_s)$
14 $\quad\quad$ **else**
15 $\quad\quad\quad Goals \leftarrow Goals \cup \{s'\}$
16 $\quad\quad\quad T'(s', \pi(s), s'_s) \leftarrow 0$
17 $\quad$ **if** $\forall \pi(s) \in \mathbf{A}_{temp} : \mathcal{T}(s', \pi(s), \bullet) = 0$ **then** $\mathbf{G}' \leftarrow \mathbf{G}' \cup Goals$;
18 return $(\langle s'_0, \mathbf{S}', \mathcal{T}', \mathbf{G}' \rangle)$

limits the number of possible goals. Note that the parameter l is always 1 for OS-GRD, but may change when used with other models.

The algorithm first initializes the output variables and a stack (line 1). It uses a stack to find successors in a DFS-fashion (lines 4-17). Initially, the start state is augmented with all candidate goals, added to the set of augmented states, and pushed to the stack (lines 2-3). For each augmented state explored, the algorithm analyzes all its successors (lines 7-16) to determine whether it is an augmented goal or not. First, a successor is augmented with the set of possible goals given its trajectory (line 9). If it is not distinctive, push it to the stack if it was not created before, add it to the set of augmented states, and update the transition function (lines 10-13). Otherwise, the explored state is marked as an augmented

goal (lines 14-16). If this explored state has 0 probability of transitioning to any other augmented state, it is added to the set of augmented goals (line 17). Finally, the augmented parameters are returned (line 18).

Determining *wcd*

As stated before, to find the *wcd*, we need to find the maximum expected cost in the augmented MDP, that is:

$$wcd(P) = \max_{\hat{\pi} \in \Pi_{aug}} V_{\hat{\pi}}(s'_0) \tag{4.2}$$

$$V_{\hat{\pi}}(s') = \sum_{s'' \in \mathbf{S}} \mathcal{T}(s', \hat{\pi}(s'), s'') \big[\mathcal{C}'(s', \hat{\pi}(s'), s'') + V_{\hat{\pi}}(s'') \big] \tag{4.3}$$

where Π_{aug} is the set of augmented policies in the augmented MDP, $s'_0 = s_0 \cdot \langle T \dots T \rangle$ is the augmented initial state, $s' = s \cdot \langle pos_1 \dots pos_n \rangle$ is an augmented state, and $V_{\hat{\pi}}(s'_0)$ is the expected cost for s'_0 with augmented policy $\hat{\pi}$ computed recursively using Equation 4.3.

Observe that Equation 4.2 is analogous to the brute force algorithm to solve an MDP (Mausam and Kolobov, 2012) that performs a policy evaluation over all enumerated policies to return the best policy. To avoid policy enumeration, we propose to consider all optimal policies simultaneously by creating a single augmented MDP and maximize the expected cost instead of minimizing it (as in MDPs). Since Value Iteration (VI) is faster than Policy Iteration in regular MDPs, we aim to also optimize over the value function space in augmented MDPs.

This value function optimization can be done using a Bellman-like equation:

$$V^*(s') = \max_{a' \in \mathbf{A}'} \sum_{s'' \in \mathbf{S}'} \mathcal{T}'(s', a', s'') \big[\mathcal{C}'(s', a', s'') + V^*(s'') \big] \tag{4.4}$$

but note that it uses the *maximization* operator instead of the minimization operator for regular MDPs. This difference will cause an issue if there are *infinite-cost cycles*, which are cycles in the graph where the optimal policy is to stay in the cycles and accumulate an infinite cost. Fortunately, our augmented MDP does *not* have infinite-cost cycles, and Equation 4.4 will thus return the correct finite value upon convergence. These properties are formalized in Lemma 5 and Theorem 3.

Lemma 5. *The augmented MDP does not have infinite-cost cycles.*

Proof. We prove that an infinite-cost cycle cannot exist by contradiction. Assume that such a cycle exists. Thus, $\exists s'_a \in \mathbf{S}', \hat{\pi}_a \in \Pi_{aug} : \sum_{i=1}^{m} P(s'_i | \hat{\pi}_a(s'_a), s'_a) = 1 \wedge \exists i = 1, \ldots, m : s'_i \neq s'_a$, also $\forall s'_i \neq s'_a : P(s'_a | \hat{\pi}_i, s'_i) = 1$ and $\hat{\pi}_a \neq \hat{\pi}_i$. In words, an infinite-cost cycle cannot contain only one state as it would mean that $\hat{\pi}_a$ contains an infinite-cost cycle, and that is not possible for optimal policies of SSP-MDPs (Mausam and Kolobov, 2012). It also means that at least two different augmented policies form the infinite cycle.

By condition of the augmented transition function (discussed in Subsection 4.1.3, p. 64), if $\mathcal{T}'(s \cdot \langle pos_1 \ldots pos_n \rangle, \pi(s), s' \cdot \langle pos'_1 \ldots pos'_n \rangle) > 0$ then $\forall i \in \{1, \ldots, n\} : pos_i = F \implies pos'_i = F$, that is, no discarded goal can become possible again. Hence, all states in the cycle have the same set $\mathbf{G}_c \subseteq \mathbf{G}'$ of possible goals, and $\mathbf{G}(\hat{\hat{\pi}}_a) \cap_{i=1,\ldots,m} \mathbf{G}(\hat{\pi}_i) \cap \mathbf{G}_c = \mathbf{G}_c$,

i.e., any policy that form the infinite-cost cycle satisfies those goals. Let $g \in \mathbf{G}_c$ be one of the possible goals. Then, s'_a transitions to s'_i through $\hat{\pi}_a(s'_a)$, optimal for g, and every $\hat{\pi}_i$ $(i = 1, \ldots, m)$, also optimal for g, reaches s'_a with 100% of probability. Therefore, policies optimal for g form an infinite-cost cycle, which is not possible as it contradicts the principle of optimality (Bellman, 1957). ■

Theorem 3. *$wcd = V^*_{\hat{\pi}}(s_0)$, which can be computed recursively via Equation 4.4.*

Proof. Equation 4.4 is, like the original Bellman equation Bellman, 1957, a contracting operator. As such, it will eventually converge to the true optimal value. The only exception is if there are infinite-cost cycles, which will cause the value of some states to converge to infinity. However, since there are no infinite-cost cycles in our augmented MDP (Lemma 5), they will converge to finite values. ■

To compute the *wcd* of each problem, one can use a VI-like algorithm that runs iterations of the Bellman-like update of Eq. 4.4. Additionally, we make the observation that the augmented state space of the augmented MDP can often be segmented into *strongly connected components* (SCCs) — each SCC contains the augmented states with the same set of possible goals, and the set of possible goals is non-increasing. Therefore, we also propose a TVI-like algorithm that uses Tarjan's algorithm (Tarjan, 1972) to segment the augmented state space into SCCs first before running VI on each SCC in reverse topological order. This should significantly speed up the solving time if there are large numbers of SCCs, but may have the opposite effect if there are few SCCs due to the overhead incurred by Tarjan's algorithm.

4.1.5 Using Policy Enumeration

Subsections 4.1.2 to 4.1.4 (p. 63 – 68) presented a method (the All-Goals strategy) to compute *wcd*. The All-Goals strategy is useful because it avoids policy enumeration. However, there are cases where that strategy will not find the *wcd* value, but an upper bound. The next subsections present a method that finds the *wcd* value in all cases.

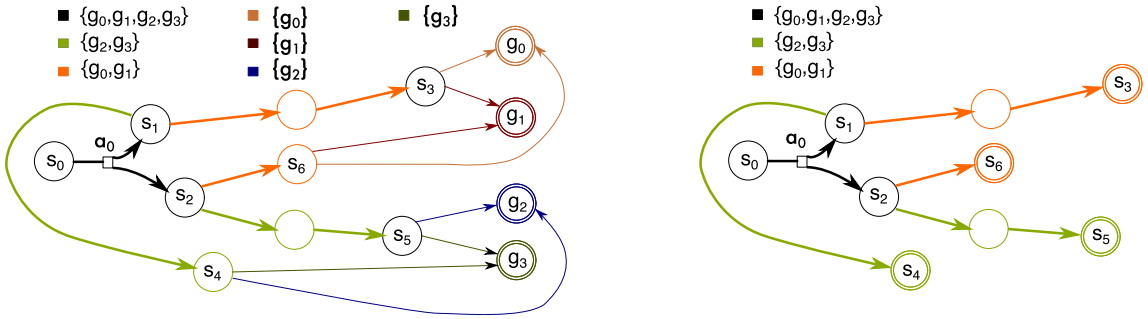


Figure 4.4: OS-GRD Example: all actions (arrows) have unitary costs. Distinctiveness is color-coded (see legend). (a) Original problem. (b) Non-distinctive actions.

Example 4. *Consider Figure 4.4, where bold arrows represent non-distinctive actions. A multi-head bold black arrow denotes the only action with stochastic outcomes (action* a_0*). All actions have unitary costs, and the set of possible goals at each state (node) is color-coded. There are four achievable goals and one proper policy to reach each one of them. Figure 4.4(a) presents the original problem and Figure 4.4(b) the state space reached by non-distinctive actions, which in this case, is equivalent to an augmented MDP as defined in Subsection 4.1.3. Figure 4.5 shows all legal policies identified with red numbers.*

Example 4 violates the All-Goals strategy's condition; that is, not all trajectories of policies with the largest expected distinctiveness (ED) have higher distinctiveness costs (DC) than other trajectories. In our case, all policies share the maximum ED: policies containing actions marked in orange have the same expected cost of policies with actions marked in green. However, not all trajectories with the highest DC belong to the same policy; that is, the All-Goals strategy cannot correctly solve it: Assuming $\mathcal{T}(s_0, a_0, s_1) = \mathcal{T}(s_0, a_0, s_2) = 0.5$, the maximum expected cost at the initial state in Figure 4.4(b) is $V(s_0) = 3$. On the other hand, the largest ED, i.e., the wcd according Definition 17 is $wcd = 2.5$. Note that when the trajectory with the highest DC has significantly higher probability to be executed than others, the value found using the All-Goals strategy is closer to the wcd of the problem. Assuming $\mathcal{T}(s_0, a_0, s_1) = 0.9$ and $\mathcal{T}(s_0, a_0, s_2) = 0.1$, $wcd = 2.9$ and the maximum expected cost at the initial state in Figure 4.4(b) does not change.

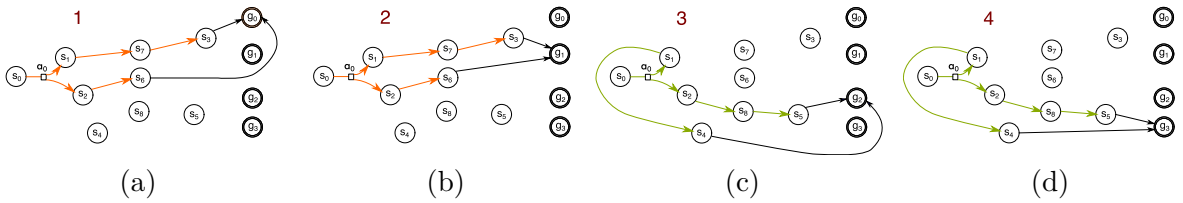
Since the reasons to account for all goals detailed in Subsection 4.1.2 remain true, one solution would be to:

1. Create one augmented MDP per goal g_i similar to the one specified in Subsection 4.1.4 without creating states augmented with sets of goals that do not contain g_i.

2. Find the maximum expected cost $V(s_0)$ for each one of them.

3. Find the maximum above all.

Nevertheless, we decided to enumerate all legal policies and keep track of them. In few cases the number of legal policies will be smaller than the number of possible goals. However, when keeping track of the policies, we can easily generalize the method to account for suboptimal settings and we can exploit more properties to optimize our algorithms for design [4]. The following subsections present the policy enumeration approach. Keeping track of the legal policies requires to define a new MDP augmented with policy IDs. To evaluate the ED of a policy, we consider only the augmented space *reachable* by that policy. In our example, to evaluate the DC of policy 1 (Figure 4.5(a)) we only consider states s_0, s_1, s_2, s_3, s_6, and s_7.



(a) (b) (c) (d)

Figure 4.5: OS-GRD Example: all optimal policies. Red numbers denote the policy IDs. Non-distinctive partial policies are colored. (a) Optimal policy for goal g_0. (b) Optimal policy for goal g_1. (c) Optimal policy for goal g_2. (d) Optimal policy for goal g_3.

4.1.6 Policy-Aware Augmented MDP

The augmented states of a policy-aware augmented MDP for OS-GRD contain a Boolean variable pos^g_ρ corresponding to each legal policy π_ρ for goal g, keeping track of non-discarded policies and, indirectly, goals. The terminal states of this MDP are those where all successors have less than two possible goals, with transitions defined according to the original MDP.

[4] Section 4.3 (p. 105) presents the approach for SS-GRD and Chapter 5 (p. 118) discuses the design methods.

Formally, given a stochastic GR model $P = \langle M = \langle \mathbf{S}, s_0, \mathbf{A}, \mathcal{T}, \mathcal{C} \rangle, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$ (with $k = 0$ restricting to optimal policies), a policy-aware augmented MDP for OS-GRD $\Pi_{aug} = \langle \mathbf{S}', s'_0, \mathbf{A}', \mathcal{T}', \mathcal{C}'_o, \mathbf{G}' \rangle$ is defined as follows:

- $\mathbf{S}' = \mathbf{S} \times \{T, F\}^{|\Pi_\mathbf{G}|}$: for each $s \in \mathbf{S}$ we create $2^{|\Pi_\mathbf{G}|}$ augmented states, corresponding to all subsets of possible legal policies.
- $s'_0 = s_0 \cdot \langle T \dots T \rangle$: initially all legal policies are possible.
- $\mathbf{A}' = \mathbf{A}$, as action labels are the same.
- $\mathcal{T}'(s \cdot \langle pos^1_1 \dots pos^g_\rho \rangle, a, s' \cdot \langle pos'^1_1 \dots pos'^g_\rho \rangle) =$

$$\begin{cases} \mathcal{T}(s,a,s') & (\exists i \neq j, m \neq n : pos'^m_i = pos'^n_j = T) \wedge \\ & \forall \rho \in \{1 \dots |\Pi_\mathbf{G}|\} : pos'^g_\rho = \\ & \big(pos^g_\rho \wedge \pi_\rho \in \Pi^k_g \wedge \pi_\rho(s) = a \wedge id(\pi_\rho) = \rho\big) \\ 0 & \text{otherwise} \end{cases}$$

  where $id : \Pi_\mathbf{G} \to \mathbb{Z}^+$ is a function mapping legal policies to policy IDs. When executing action a in state s, the flag pos'^g_ρ, indicating whether a policy π_ρ is possible, becomes false if a is not part of the legal policy π_ρ for goal g (or remains false if it was already false). States with less than two possible goals are no longer reachable.
- $\mathcal{C}'(s \cdot \langle pos^1_1 \dots pos^g_\rho \rangle, a, s' \cdot \langle pos'^1_1 \dots pos'^g_\rho \rangle) = \mathcal{C}_o(s,a,s')$: we want to find policies with maximal cost for the observer, and

- $\mathbf{G'} = \{s \cdot \langle pos_1^1 \dots pos_\rho^g \rangle \mid \forall \pi(s) : \pi \in \Pi_\mathbf{G} \implies \mathcal{T}'(s \cdot \langle pos_1^1 \dots pos_\rho^g \rangle, \pi(s), s' \cdot \langle pos_1'^1 \dots pos_\rho'^g \rangle) = 0\}$

  Augmented goal states are those where any applicable action will transition to an augmented state with less than two goals.

A resultant policy-aware augmented MDP for Example 4 has the structure shown in Figure 4.4(b), where black states and actions are augmented with policies 1 through 4, orange denotes states and actions augmented with policies 1 and 2, and green indicates states and actions augmented with policies 3 and 4.

4.1.7 Computing *wcd*: Algorithms

We first provide a high-level description of our approach to compute *wcd* for OS-GRD problems using policy enumeration. Initially, all legal policies per goal are identified by finding (and marking) all optimal policies. Next, we evaluate the expected distinctiveness of each policy using its reachable augmented state space, avoiding to recompute policies that share its entire non-distinctive prefix with an already assessed policy. For instance, since policy 1 and 2 share the same non-distinctive prefix (orange portion in Figures 4.5(a) and (b)), the algorithm will only evaluate one of them. Finally, the *wcd* is the largest expected cost found. Since this evaluation is costly, we store all partial results in a priority queue to minimize recomputation during design.

Computing *wcd*: Practical Considerations

As stated before, the highest expected cost at the starting state of an augmented MDP for OS-GRD is not always equivalent to the *wcd* of the problem. Therefore, we need to evaluate the non-distinctive prefix of every legal policy. There is no need to generate all $2^{|\Pi_{\mathbf{G}}|}$ augmented states, just the reachable states using the policy to be evaluated. While building this smaller augmented MDP, we keep track of all policies that share all non-distinctive trajectories and group the ones sharing their maximum non-distinctive prefixes; the *wcd* is equivalent to the maximum expected cost among all groups. In the worst case, we will need to evaluate individually all $\mid \Pi_{\mathbf{G}} \mid$ policies.

Procedure CommonNDPoliciesFO presents the steps we follow to group augmented policies for evaluation and to build the partial augmented MDP. CommonNDPoliciesFO receives a policy ID $\rho \in \mho$, the set of all policy IDs $\mho$, the initial state s_0, the set of states $\mathbf{S}$, the set of marked legal policies $\Pi^{\mho}_{\mathbf{G}}$, and the transition function $\mathcal{T}$ as input parameters; each legal action $\pi(s)$ is augmented with a set of policy IDs corresponding to all policies that use $\pi(s)$. We use π_ρ to identify a marked policy with ID ρ, $\mho(\pi(s))$ to denote the set of policy IDs that use action $\pi(s)$, and $\mathbf{G}(\mho)$ to represent all goals of policies with IDs in $\mho$.

First, all variables are initialized and the starting state is augmented with the set of all policy IDs and pushed to a stack (Lines 19-21). Then, we traverse the original MDP in a DFS fashion to find all reachable states using policy $\hat{\pi}_\rho$ and create the partial augmented MDP (Lines 22-36). We keep track of valid policy IDs by intersecting the set corresponding

Procedure CommonNDPoliciesFO($\rho, \mho, s_0, \mathbf{S}, \Pi^{\mho}_{\mathbf{G}}, \mathcal{T}$)

19 $Stack \leftarrow \emptyset; \mathbf{S}' \leftarrow \emptyset; \mho_i \leftarrow \mho, \mathcal{T}' \leftarrow null, \mathbf{G}' \leftarrow \emptyset$
20 $s_0^{\mho} \leftarrow \langle s_0, \mho \rangle$
21 $Stack.push(s_0^{\mho})$
22 **while** $Stack \neq \emptyset$ **do**
23 $\quad s^{\mho'} = \langle s, \mho' \rangle \leftarrow Stack.pop$
24 $\quad$ **if** $s \notin \mathbf{S}'$ **then**
25 $\quad\quad \mathbf{S}' \leftarrow \mathbf{S}' \cup \{s^{\mho'}\}$
26 $\quad\quad \mho'' \leftarrow \emptyset$
27 $\quad\quad$ **foreach** $\mathcal{T}(s, \hat{\pi}_\rho(s), s_s) > 0 \mid \hat{\pi}_\rho \in \Pi^{\mho}_{\mathbf{G}}$ **do**
28 $\quad\quad\quad \mho'' \leftarrow \mho' \cap \mho(\hat{\pi}_\rho(s))$
29 $\quad\quad\quad$ **if** $|\mathbf{G}(\mho'')| > 2$ **then**
30 $\quad\quad\quad\quad Stack.push(s^{\mho''} \leftarrow \langle s_s, \mho'' \rangle)$
31 $\quad\quad\quad\quad \mathcal{T}'(s^{\mho'}, \hat{\pi}_\rho(s), s_s^{\mho''}) \leftarrow \mathcal{T}(s, \hat{\pi}_\rho(s), s_s)$
32 $\quad\quad\quad$ **else**
33 $\quad\quad\quad\quad \mathcal{T}'(s^{\mho'}, \hat{\pi}_\rho(s), s_s^{\mho''}) \leftarrow 0$
34 $\quad\quad$ **if** $\mho'' = \emptyset \vee \mathcal{T}'(s^{\mho'}, \hat{\pi}_\rho(s), \bullet) = 0$ **then**
35 $\quad\quad\quad \mathbf{G}' \leftarrow \mathbf{G}' \cup \{s^{\mho'}\}$
36 $\quad\quad\quad \mho_i \leftarrow \mho_i \cap \mho'$
37 $\mho_i \leftarrow removeND(\mho_i, \mathbf{G}', \Pi^{\mho}_{\mathbf{G}}, \mathcal{T})$
38 return $\langle \mho_i, \mathbf{S}', \mathcal{T}', \mathbf{G}' \rangle$

to the current state with the action's respective set (Line 28). If the resulting set contains policies for more than two goals, the procedure pushes the augmented successors to the stack and updates the augmented transition function (Lines 29-33). All augmented states that cannot transition to any other state are identified as augmented goals and the policies common to all these goals are saved (Lines 34-36). The function $removeND$ checks if the common policies could have longer non-distinctive prefixes and in case they do, removes them from $\mho_i$ (Line 37). The final set $\mho_i$ signals policies with the same non-distinctive trajectories that can be combined when computing the maximum expected value. Finally, the procedure

returns $\mho_i$ and the components of an augmented MDP useful to evaluate the group of policies with IDs in $\mho_i$ (Line 38).

The solution of the augmented MDP generated using Procedure CommonNDPoliciesFO gives the expected cost of the largest non-distinctive prefix of all policies $\Pi^{\mho_i}$ with IDs indicated in $\mho_i$. The *wcd* is then computed using:

$$wcd(P) = \max_{i=1...n} V_{\Pi^{\mho^i}}(s'_0) \tag{4.5}$$

$$V_{\Pi^{\mho^i}}(s') = \sum_{s'' \in \mathbf{S}'} \mathcal{T}'(s', \pi(s'), s'')[\mathcal{C}'(s', \pi(s'), s'') + V_{\Pi^{\mho^i}}(s'')] \tag{4.6}$$

where: $\pi \in \Pi^{\mho^i}$ and $\bigcup_{i=1}^{n} \mho^i = \mho \wedge \bigcap_{i=1}^{n} \mho^i = \emptyset$

We use a TVI-like algorithm that runs iterations of Eq. 4.6 for groups of policies with the same non-distinctive trajectories, and find the maximum among all using Eq. 4.5.

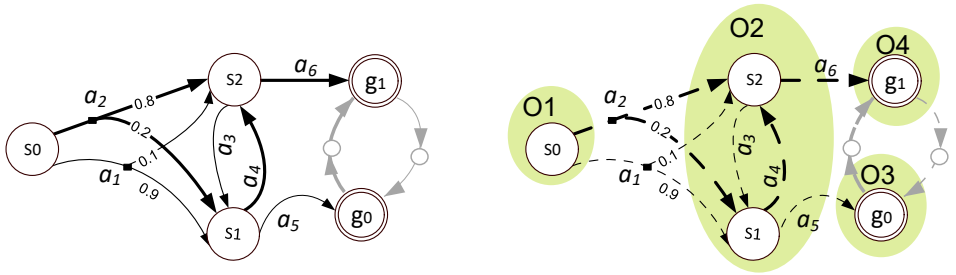
4.2 Partially-Observable S-GRD (POS-GRD)

In this section, we extend the definition of the worst-case distinctiveness measure to account for partial observability (of an observer) in stochastic environments. The model, which we call Partially-Observable S-GRD (POS-GRD), assumes that the agent's actions are no longer observable, and agent's states are partially observable, so that several states may be indistinguishable from one another. The degree of observation uncertainty is related to the resolution of sensors in the problem.

Example 5. *To illustrate the setting of this work, we present an example in Figure 4.6; the fully-observable version is shown in Figure 4.6(a). Due to low sensor resolution, more than one state can be mapped to the same observation, as in Figure 4.6(b), where states $S2$ and $S1$ are perceived as only one. In this example, all actions have cost of 1; we represent states as annotated nodes, unobservable actions with annotated dashed edges, and observations using annotated shaded areas. Goals are marked with double-lined circles. Each edge is labeled with an action name, and an action with a stochastic outcome is represented by a multi-head arrow, with probabilities associated with each arrow head. We also mark the intended goal of an action using arrow width, where thin arrows represent intended goal g_0 and thick arrows represent intended goal g_1.*

The observation model we propose is different from others such as HMM or POMDP as we assume actions are not observable *at all*. The observer only observes a transition between two states that are associated with different observations. Any transition between states that are mapped to the same observation is undetectable. This is suitable for sensors that produce a continuous reading of the observed state and settings in which the agent can spend an arbitrary amount of time in each state.

Note that the OS-GRD version of the example has a wcd=0 as each policy satisfies optimally only one goal, and as soon as the agent executes an action reveals its goal. However, with a sensor configuration as in Figure 4.6(b), an observer is not able to disambiguate the goals until the agent arrives at its target. We now present algorithms to find wcd under these conditions.



(a) wcd=0 (b) wcd=1.2

Figure 4.6: POS-GRD Assumptions. (a) OS-GRD. (b) POS-GRD: Non-observable actions and states are only partially observable (green shades).

Subsections 4.2.1 to 4.2.3 (p. 83 – 94) present algorithms that allow us to find the largest expected cost among *all possible non-distinctive observable trajectories*, which is equivalent to compute *wcd* when all observable trajectories of a policy with maximal expected distinctiveness (Definition 17, p. 55) have distinctiveness costs larger or equal than trajectories of any other policies. Subsections 4.2.4 to (p. 100) present a method to compute *wcd* for all cases.

4.2.1 Augmented MDP for POS-GRD

The sets of possible goals at specific states demonstrate a non-Markovian behavior, which depends not just on the current state but also on what we have observed in the past. Intuitively, once goal g has been eliminated as a possible goal (by observing the agent performing an action that is not part of an optimal policy with respect to g), then g never becomes a possible goal again, even if the agent executed an action that is optimal with respect to g. We, therefore, propose the use of augmented MDPs to capture the non-Markovian behavior in a partial observability setting.

Let $P = \langle M, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$ be a partially-observable goal recognition model with stochastic action outcomes, (Definition 1, p. 45), with $k = 0$ for optimal agents, $M = \langle \mathbf{S}, s_0, \mathbf{A}, \mathcal{T}, \mathcal{C}, \emptyset \rangle$ being an MDP with positive costs $\mathcal{C}$ for the agent and no goal. An augmented MDP adds a Boolean variable pos_g for each possible goal g, to keep track of whether g has been eliminated as a possible goal or not. The terminal states of this MDP are those that have less than two possible goals, with transitions and costs defined according to the original MDP. To comply with Definition 17 (p. 55), the cost of an action transitioning to a terminal state is 0.

We account for partial observability by overlaying a sensor model on the augmented MDP. We first define a notion of connectivity in which an agent can transition from state s to state s', while following a policy that is optimal with respect to some goal $g \in \mathbf{G}$, without being observed.

Definition 18. *State s is* unobservably connected *to state s' with respect to a set of possible goals $\mathbf{G}$ if there exists a policy $\pi \in \cup_{g \in \mathbf{G}} \Pi_g^0$, and a trajectory $\vec{\tau} = \langle s_0, \pi(s_0), s_1, \pi(s_1), \ldots, s_n \rangle$ with $s = s_0$ and $s' = s_n$, such that $\mathcal{N}(s_0) = \mathcal{N}(s_1) = \ldots = \mathcal{N}(s_n)$, and with $\mathcal{T}(s_i, \pi(s_i), s_{i+1}) > 0$ for $0 \leq i \leq n$.*

We denote by $uc_{\mathbf{G}}(s)$ the set of states s' such that s is unobservably connected to s' with respect to $\mathbf{G}$, e.g., in Figure 4.6(b), $uc_{\mathbf{G}=\{g_0,g_1\}}(s_1) = \{s_1, s_2\}$.

In an OS-GRD setting, a transition from s to s' using an action a that is not part of an optimal policy to goal g, results in the removal of g from the set of possible goals. However, if $\mathcal{N}(s) = \mathcal{N}(s')$, the transition cannot be observed and g cannot be eliminated. Moreover, even

when $\mathcal{N}(s) \neq \mathcal{N}(s')$, there may be another transition from $\hat{s} \in uc_{\{g\}}(s)$ to $\hat{s}'$ using action $\hat{a}$, such that $\mathcal{T}(\hat{s}, \hat{a}, \hat{s}') > 0$, $\mathcal{N}(s) = \mathcal{N}(\hat{s})$, $\mathcal{N}(s') = \mathcal{N}(\hat{s}')$, and $\hat{a}$ is an optimal action at $\hat{s}$ with respect to g. In this case, an observer cannot distinguish between the two transitions and as a result g still cannot be eliminated from the set of possible goals. As an example, consider Figure 4.6(a) where actions a_3 and a_4 (compensating actions for the stochasticity of actions a_1 and a_2, respectively) reveal the agent's goal since each one is optimal for only one (different) goal. In the partially-observable scenario of Figure 4.6(b), actions a_3 and a_4 are not observed as $\mathcal{N}(s_2) = \mathcal{N}(s_1)$ and even though $\mathcal{N}(s_0) \neq \mathcal{N}(s_1)$, if a_1 is executed, goal g_1 cannot be eliminated because a_2, optimal for g_1, also transitions from $\mathcal{N}(s_0)$ to $\mathcal{N}(s_1)$ and they cannot be distinguished.

Taking into account the observation above, the augmented MDP for POS-GRD $\Pi_{aug} = \langle \mathbf{S}', s'_0, \mathbf{A}', \mathcal{T}', \mathcal{C}'_o, \mathbf{G}' \rangle$ is defined as follows:

- $\mathbf{S}' = \mathbf{S} \times \{F, T\}^{|\mathbf{G}|}$: for each $s \in \mathbf{S}$ we create $2^{|\mathbf{G}|}$ possible states, corresponding to all subsets of possible goals. We use $w(s') = s$ to denote that s is the state of the world at $s' \in \mathbf{S}'$.
- $s'_0 = s_0 \cdot \langle T \dots T \rangle$: initially all goals are possible.
- $\mathbf{A}' = \mathbf{A}$ (action labels remain unchanged).
- $\mathcal{T}'(s \cdot \langle pos_1 \dots pos_n \rangle, a, s' \cdot \langle pos'_1 \dots pos'_n \rangle) =$

$$
\begin{cases}
\mathcal{T}(s,a,s') & \forall i \in \{1\ldots,n\}(pos'_i = (pos_i \wedge & (4.7)\\
 & \big(\mathcal{N}(s) = \mathcal{N}(s') & (4.8)\\
 & \vee(\exists \pi \in \Pi^0_{g_i} \mid \pi(s) = a) & (4.9)\\
 & \vee(\exists \pi \in \Pi_{\mathbf{G}} \wedge \exists \hat{s} : \mathcal{N}(s) = \mathcal{N}(\hat{s}) \wedge & (4.10)\\
 & \exists \hat{s}' \mid \mathcal{T}(\hat{s}, \pi(\hat{s}), \hat{s}') > 0 \wedge & (4.11)\\
 & \mathcal{N}(s') = \mathcal{N}(\hat{s}'))))) & (4.12)\\
0 & \text{otherwise} & (4.13)
\end{cases}
$$

To compute whether the probability that executing action a when in state s with $\langle pos_1 \ldots pos_n \rangle$ (where pos_i indicates whether goal g_i is possible) leads to state s' with $\langle pos'_1 \ldots pos'_n \rangle$ is equal to $\mathcal{T}(s,a,s')$, we test the observer belief regarding each g_i according to the following cases. A goal cannot become possible (Line 4.7). A goal remains possible if s' is unobservably connected (Definition 18) to s (Line 4.8) or a is optimal with respect to the goal (Line 4.9). Finally, lines 4.10-4.12 cover the case of undistinguishable actions discussed above.

- $\mathcal{C}'(s \cdot \langle pos_1 \ldots pos_n \rangle, a, s' \cdot \langle pos'_1 \ldots pos'_n \rangle) = \begin{cases} \mathcal{C}_o(s,a,s') & \forall (s' \cdot \langle pos'_1 \ldots pos'_n \rangle) \notin \mathbf{G}' \\ 0 & \text{otherwise} \end{cases}$

  We want to find policies with maximal cost for the observer without including the cost that transitions to a terminal state.

- $\mathbf{G}' = \left\{ s \cdot \langle pos_1 \dots pos_n \rangle \mid \exists pos_i \forall j \neq i : pos_j = F \right.$: terminal states are those with less than two possible goals.

Lemma 6. *Let π' be any policy for the augmented MDP Π_{aug}, and define the non-distinctive partial policy $\hat{\pi}$ for P by:*

$$\hat{\pi}(s) = \begin{cases} a & \pi'(s') = a \ \forall s' \in \mathbf{S}' : w(s') = s \\ & \wedge \ \exists s'_s \in \mathbf{S}' : \mathcal{T}'(s', \pi'(s'), s'_s) > 0 \wedge s'_s \notin \mathbf{G}' \\ \bot & \textit{otherwise} \end{cases} \tag{4.14}$$

Then $V_{\pi'}(s'_0) = ED(\hat{\pi})$, that is, the expected value of policy π' in Π_{aug} at s'_0 is equal to the expected distinctiveness of $\hat{\pi}$ in P.

Proof. Let $\mathbf{S}_{\hat{\pi}}$ be the set of states reached by any non-distinctive trajectory of $\hat{\pi}$ and $\mathbf{S}'_{\pi'}$ the set of states reached by policy $\pi' \in \Pi_{aug}$. By Definition 17 (p. 55), the *expected distinctiveness* of a non-distinctive policy $\hat{\pi}$ is its expected cost. From Eq. 4.14, $\hat{\pi}$ starts at $s_0 = w(s'_0)$ and ends at states $s_n = w(s'_n) \mid \forall s'_s \in \mathbf{S}' : \mathcal{T}'(s'_n, \pi'(s'_n), s'_s) > 0 \implies s'_s \in \mathbf{G}'$. Therefore, $ED(\hat{\pi}) = V_{\hat{\pi}}(s_0)$, where:

$$V_{\hat{\pi}}(s) = \begin{cases} 0 & \text{if } \hat{\pi}(s) = \bot \\ \sum_{s_s \in \mathbf{S}_{\hat{\pi}}} \mathcal{T}(s, a, s_s) \left[\mathcal{C}_o(s, a, s_s) + V_{\hat{\pi}}(s_s) \right] & \textit{Otherwise} \end{cases} \tag{4.15}$$

By Eq. 3.9 (p. 55), the expected distinctiveness will not include costs of non distinctive trajectories. Hence, Eq. 4.15 considers only successors reachable by non-distinctive trajectories.

On the other hand, $V_{\pi'}(s'_0)$ can be evaluated using:

$$V_{\pi'}(s') = \begin{cases} 0 & \text{if } s' \in \mathbf{G}' \\ \sum_{s'_s \in \mathbf{S}'_{\pi'}} \mathcal{T}(s', a, s'_s)\big[\mathcal{C}'(s', a, s'_s) + V_{\pi'}(s'_s)\big] & \textit{Otherwise} \end{cases} \tag{4.16}$$

By construction of the augmented MDP, $\forall s', s'_s \in \mathbf{S}'_{\pi'}, \exists s, s_s \in \mathbf{S}_{\hat{\pi}} : s = w(s') \wedge s_s = w(s'_s) \wedge \mathcal{T}(s', a, s'_s) = \mathcal{T}(s, a, s_s)$. Additionally, if $s = w(s') \wedge s_s = w(s'_s) \wedge s'_s \notin \mathbf{G}'$ then $\mathcal{C}'(s', a, s'_s) = \mathcal{C}_o(s, a, s_s)$, else $\mathcal{C}'(s', a, s'_s) = 0$. In words, transitioning to an augmented goal is possible, but the cost of that transition is 0. If $s' \in \mathbf{G}'$ then $s = w(s') \notin \mathbf{S}_{\hat{\pi}}$, that is, s is not part of any non-distinctive trajectory, nor is the action reaching s. Therefore, Eqs. 4.15 and 4.16 provide the same values for all states $s \in \mathbf{S}_{\hat{\pi}}$ and $s' \in \mathbf{S}_{\pi'}$. Hence, $V_{\hat{\pi}}(s_0) = ED(\hat{\pi}) = V_{\pi'}(s'_0)$. ■

Lemma 6 connects the expected distinctiveness cost of a partial policy value to the expected value from a policy in the augmented MDP. Next, we define *legal augmented policies* where $\mathbf{S}$ represent the set of states in the regular MDP and $\mathbf{S}'$ the set of augmented states.

Definition 19. *Let π' be any policy for the augmented MDP Π_{aug} with trajectories $\mathbf{T} = \{\vec{\tau} = \langle s'_0, a_0, \ldots, s'_{n-1}, a_{n-1}, s'_n\rangle \mid \forall i \in \{0, \ldots, n\} : a_i = \pi'(s'_i)\}$. π' is a* legal augmented policy *if and only if there exists a non-distinctive policy $\hat{\pi} \in \Pi_{\mathbf{G}}$ in the regular MDP M*

with trajectories $\widehat{\mathbf{T}} = \{\vec{\tau} = \langle s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n\rangle \mid \forall i \in \{0, \ldots, n\} : a_i = \hat{\pi}(s_i)\}$ *and* $\forall s_i \in \mathbf{S}, \exists s'_i \in \mathbf{S}' : w(s'_i) = s_i \wedge a_i = \hat{\pi}(s_i) = \pi'(s'_i)$

The following corollary establishes the connection to the *wcd*, leading to the algorithm to be detailed next for efficiently computing the *wcd* for optimal agents.

Corollary 1. *Let π' be a maximal legal augmented policy, and let $\hat{\pi}$ be as defined in Eq. 4.14, then $V_{\pi'}(s_0)$ is equal to the* wcd*.*

Proof. By Definition 19, there exists a non-distinctive policy $\hat{\pi}$ in the regular MDP that has trajectories equivalents to π'. Lemma 6 guarantees that both policies have the same expected cost. Since π' is maximal, $\hat{\pi}$ should also be maximal, therefore, the expected cost $V_{\hat{\pi}}(s_0) = wcd = V_{\pi'}(s_0)$. ■

4.2.2 Augmented MDP for POS-GRD: Practical Considerations

Similar to the All-Goals strategy for OS-GRD, we would like to construct an augmented MDP where finding the maximal expected cost does not require to evaluate individual policies over all enumerated augmented policies. To avoid policy enumeration, we propose to consider all augmented policies simultaneously and maximize the expected cost instead of minimizing it (as in MDPs). However, we need to be careful of *infinite-cost cycles*, that is, cycles in the graph where the optimal policy is to stay in the loop and accumulate an infinite cost. Next, we analyze the possible existence of those types of loops.

Infinite-Cost Cycles in the Augmented MDP for POS-GRD

Finding the maximum expected cost in an MDP is equivalent to finding the *wcd* only if there are no infinite-cost cycles. By Lemma 5, an augmented MDP for OS-GRD does not have that type of cycles, thus, it must be the case that they are caused by partial observability. We will characterize those cases and show how to remove infinite-cost cycles.

Definition 20. *We formally define an infinite-cost cycle in the augmented MDP as follows:* $\exists s'_a, s'_i \in \mathbf{S}', \pi_a \in \Pi_{\mathbf{G}} : \sum_i^m \mathcal{T}'(s'_a, \pi_a(s'_a), s'_i) = 1 \wedge \exists i = 1, \ldots, m : s'_i \neq s'_a$ *and* $\forall s'_i \in \mathbf{S}' : \mathcal{T}'(s'_a, \pi_a(s'_a), s'_i) > 0, \exists \Pi_i \subseteq \Pi_{\mathbf{G}} : P(s'_a \mid \Pi_i, s'_i) = 1$.

Note that any state in an infinite loop must reach other states in the loop with a 100% probability. Further, by construction, all actions used in the reachable part of the augmented MDP are optimal actions in the regular MDP (augmented transition, Lines 4.11, and 4.12). Therefore, by Lemma 7 below, infinite-cost cycles in the augmented MDP cannot contain only actions in policies optimal to the same set of goals. Additionally, an infinite-cost cycle cannot have only one state, as that would imply one optimal policy with an infinite loop, which contradicts the principle of optimality (Bellman, 1957).

Lemma 7. *Let* s'_a *be a state in an infinite-cost cycle as defined in Definition 20 and* $\pi_a(s'_a)$ *an applicable legal action. If* $\forall s'_i : \mathcal{T}'(s'_a, \pi_a(s'_a), s'_i) > 0,\ \exists \Pi_i : P(s'_a \mid \Pi_i, s'_i) = 1$, *then* $\big(\exists (s'_i \mid \mathcal{T}'(s'_a, \pi_a(s'_a), s'_i) > 0), \pi \in \Pi_i, g \in \mathbf{G}\big) : \big(g \notin \mathbf{G}(\pi_a) \cap \mathbf{G}(\pi) \wedge g \in \mathbf{G}(\pi_a) \cup \mathbf{G}(\pi)\big)$.

Proof. We prove it by contradiction, that is, assume $\forall(s'_i \mid \mathcal{T}'(s'_a, \pi_a(s'_a), s'_i) > 0), \pi \in \Pi_i, g \in \mathbf{G}' : g \in \mathbf{G}(\pi_a) \cap \mathbf{G}(\pi) \vee g \notin \mathbf{G}(\pi_a) \cup \mathbf{G}(\pi)$. If $g \notin \mathbf{G}(\pi_a) \cup \mathbf{G}(\pi)$, then g is not considered in the infinite-cost cycle, else if $g \in \mathbf{G}(\pi_a) \cap \mathbf{G}(\pi)$, $P(s'_a \mid \{\pi, \pi_a\}, s'_a) = 1$, that is, there is a 100% probability that policies π and π_a have an infinite loop. Also, by Lemma 6, if $P(s'_a \mid \Pi_i, s'_i) = 1$, then $P(w(s'_a) \mid \Pi_i, w(s'_i)) = 1$.Thus, combining both statements, if $w(s'_a) = s_a$, we have that $\forall g \in \hat{\mathbf{G}}, \exists \pi \in \Pi^0_g : P(s_a \mid \{\pi, \pi_a\}, s_a) = 1$; that is, there are policies, both optimal for goal g that form an infinite-cost cycle in the regular MDP. This is not possible as it contradicts the principle of optimality (Bellman, 1957). ■

Moreover, by construction of the augmented transition function for POS-GRD, a goal cannot become possible after it is discarded (Line 4.7). Hence, all states in the infinite cycle ($s' \in \mathbf{S}'_c$) have the same set of possible goals, i.e., $\forall s'_1 \neq s'_2 \in \mathbf{S}_c : s'_1 = s_1 \cdot \langle pos_1 \dots pos_n \rangle \wedge s'_2 = s_2 \cdot \langle pos'_1 \dots pos'_n \rangle \implies \forall i \in \{1, \dots, n\} : pos_i = pos'_i$.

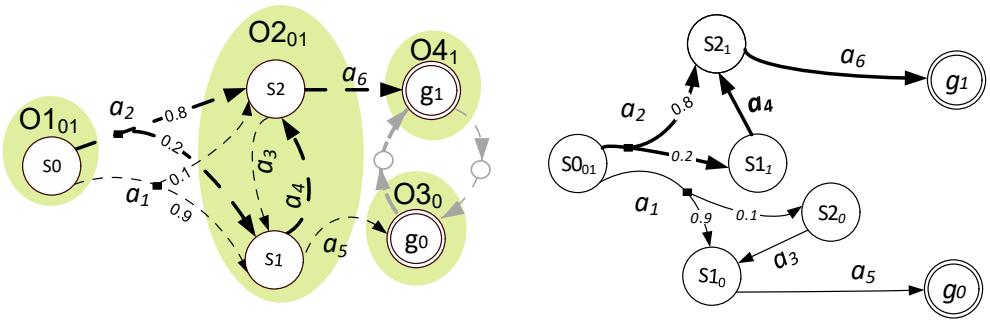
We will now consider the possible cases of partial observability under the POS-GRD assumptions:

1. Fully-observable states and non-observable actions; and

2. Partially-observable states and non-observable actions.

For the first case, let $\mathbf{G}_c$ be the set of possible goals of all states $s' \in \mathbf{S}'_c$, the set of actions applicable in state s'_a is $\Pi_{s'_a} = \{\pi(s'_a) \mid \pi \in \Pi_{\mathbf{G}} \wedge \forall s'_a \in \mathbf{S}'_c : \mathcal{T}'(s'_a, \pi(s'_a), \bullet) > 0\}$, and $\mathbf{G}_{\Pi_{s'_a}} = \bigcup_{\pi(s'_a) \in \Pi_{s'_a}} \mathbf{G}(\pi(s'_a))$ is the set of all goals satisfied by all those applicable actions.

Since all states are fully-observable, we have that $\forall s'_a, s'_b \in \mathbf{S}'_c, \pi(s'_a) \in \Pi_{s'_a} : \mathcal{T}'(s'_a, \pi(s'_a), s'_b) > 0 \implies \mathbf{G}'_c = \mathbf{G}'_c \cap \mathbf{G}_{\Pi_{s'_a}}$, that is, all states in the infinite-cost cycle are reachable through actions satisfying each goal $g \in \mathbf{G}'_c$. This contradicts Lemma 7 as $\mathbf{G}'_c \neq \emptyset$. Hence, infinite-cost cycles must occur in the second case.



(a) (b)

Figure 4.7: POS-GRD (cont.)(a) Infinite-cost cycle in POS-GRD. (b) Cycle-free MDP.

Example 6. *Figure 4.7(a) shows an example of an infinite-cost loop formed by actions* a_3 *and* a_4*, each from policies satisfying different sets of goals. States* $S1$ *and* $S2$ *are* unobservably connected *(Definition 18, p. 84) and visually represented in the same green shape. Also, subindexes in each observation* O_i*, denote the set of possible goals of all unobservably connected states.*

Removing Infinite-Cost Cycles

To remove infinite-cost cycles caused by partially-observable states, we propose to apply first a modified version of the augmented MDP rules for OS-GRDs (Subsection 4.1.3, p. 64), specifically, use the following augmented transition function:

$$\mathcal{T}''(s \cdot \langle pos_1 \dots pos_n \rangle, a, s' \cdot \langle pos'_1 \dots pos'_n \rangle) = \begin{cases} \mathcal{T}(s, a, s') & pos'_i = \big(pos_i \wedge (\exists \pi \in \Pi^0_{g_i} \mid \pi(s) = a)\big) \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

The resultant MDP is then augmented following the rules for partial observability.

Lemma 8. *Eq. 4.17 breaks infinite-cost cycles.*

Proof. Let s' denote a state in the augmented MDP, and $s = w(s')$ represent the corresponding state in the regular MDP. By Lemma 7, there must be at least two policies, with different satisfying sets of goals, that form the infinite-cost cycle as defined in Definition 20. Given a state $s'_a \in \mathbf{S}'_c$, from Lemma 7, $\forall s'_b \in \mathbf{S}'_c : \mathcal{T}'(s'_a, \pi_a(s'_a), s'_b) > 0, \exists \pi_b \in \Pi_b, g \in \mathbf{G} : P(s'_a \mid \Pi_b, s'_b) > 0 \wedge g \in \mathbf{G}(\pi_b) \cup \mathbf{G}(\pi_a) \wedge g \notin \mathbf{G}(\pi_b) \cap \mathbf{G}(\pi_a)$. Eq. 4.17 adds information of the set of goals satisfying policies into the states of a regular MDP. Below, we identify the new states with a circumflex accent.

If $\mathcal{T}(s_a, \pi_a(s_a), s_b) > 0$, then $\forall i = 1, \ldots, |\ \mathbf{G}\ |$: $(\hat{s}_a = s_a \cdot \langle pos_{g_1}, \ldots, pos_{g_{|\mathbf{G}|}} \rangle, \hat{s}_b = s_b \cdot \langle pos'_{g_1}, \ldots, pos'_{g_{|\mathbf{G}|}} \rangle \wedge pos_{g_i} = T \wedge g_i \in \mathbf{G}(\pi_a)) \implies pos'_{g_i} = F$, that is, the set of possible goals of $\hat{s}_b$ does not contain goals in $\hat{s}_a$ that belong to $\mathbf{G}(\pi_a)$; therefore, $P(\hat{s}_a \mid \Pi_b, \hat{s}_b) = 0$. ■

In essence, Eq. 4.17 breaks cycles that the agent will never take as they correspond to policies satisfying different sets of goals.

4.2.3 Computing *wcd*: Algorithms

The number of augmented states of an augmented MDP for POS-GRD is $O(|S| \times 2^{|G|})$, which is exponential in the number of model goals. Since not all augmented states are reachable, there is no need to generate them all when computing *wcd*.

The proposed method generates exactly the augmented states needed for *wcd* computation, solving a single augmented MDP, and has four main steps: (1) Find all optimal policies; (2) Join them and remove infinite cycles to avoid computing the expected distinctiveness cost for each policy; (3) Construct the augmented MDP for reachable states taking partial observability into account; (4) Solve this augmented MDP to compute the *wcd*. We next provide details of each of these steps.

Finding Optimal Policies

To identify the set of legal policies $\Pi_{\mathbf{G}}$, we separately solve an MDP for each goal. Using $V^*(s_0)$, the optimal expected cost at the starting state, we identify all optimal policies per

goal. Figure 4.7(a) shows two optimal policies, one per goal, using two different edge widths (optimal policy for g_1 is marked in bold).

Removing Cycles

Combining all optimal policies into a single augmented MDP may create infinite-cost cycles resulting from joining two or more policies that reach undistinguishable states. Solving such an augmented MDP leads to optimal policies that choose to remain within the cycle to achieve an infinite maximum expected cost. For example, Figure 4.7(a) contains a cycle of actions a_3 and a_4, which belong to different optimal policies and therefore will never be executed together by an optimal agent. Therefore, we eliminate such cycles.

As a first step, we model the agent's true behavior using Eq. 4.17, which is similar to the transition function for OS-GRDs. This step guarantees that there are no infinite loops (Lemma 8, p. 93). For each augmented state, the set of goals becomes part of the state ID to keep the MDP structure free of infinite-cost cycles. In what follows, we refer to this MDP as *cycle-free MDP* and to the sets of possible goals as *FO possible goals*.

For example, consider Figures 4.7(a)-(b). State $S0$ is augmented with goals g_0 and g_1 (denoted as subindex 01), then each action is analyzed to generate successors. When a_1 (optimal for goal g_0) is examined, states $S1$ and $S2$ are augmented with goal g_0, and when action a_2 (optimal for goal g_1) is analyzed, $S1$ and $S2$ are augmented with goal g_1. Later, when action a_3 is analyzed, no new state needs to be generated as $S1$ augmented with goal g_0 was already

created; a similar situation occurs with a_4. It is worth noting that all the augmented states generated from the same state (e.g., $S1$ or $S2$) project to the same observation. Also, the resulting cycle-free MDP has separate distinguished paths for each goal.

Constructing Augmented MDP for POS-GRD

To generate the components of the final augmented MDP from the cycle-free MDP, we use an iterative 7-step procedure: (1) Augment the initial state with all possible goals; (2) Find all unobservably connected states and augment them with the same set of goals; (3) Find all immediately connected states projecting successors not belonging to the unobservably connected states; (4) Group them according to their observations; (5) Augment states in each group; (6) Keep non-duplicated augmented states; and (7) Update the transition function.

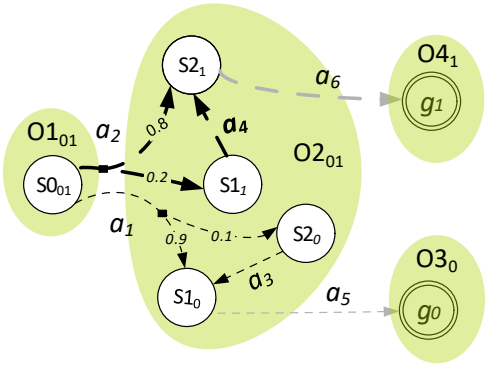


Figure 4.8: POS-GRD (cont.) Reachable Augmented MDP for POS-GRD.

To illustrate the procedure, consider the resulting augmented MDP in Figure 4.8. Possible goals for a state are shown as indices to the ID of their observations. The start sate $S0_{01}$ is

augmented with goals g_0 and g_1 (Step 1). States $S1_0, S2_0, S1_1$, and $S2_1$ are generated and since (1) actions a_0 and a_1 are each optimal for different goals; (2) both of them transition from $O1$ to $O2$; and (3) they are non-observable, then all these states are augmented with goals g_0 and g_1 (Steps 3 to 5). The transition function in this case does not change (Step 7). When $S1$ and $S2$ are examined, all their unobservably connected states should be first generated and augmented with the same set of goals. However, in this case, no new state needs to be created. Later, other connected states not projecting both g_0 and g_1 are generated and augmented following the same procedure. Note that the cost of actions a_6 and a_5 is 0 and is not used to compute the *wcd* since goals g_0 and g_1 are distinctive.

Algorithm 2 presents a pseudocode for constructing an augmented MDP for POS-GRD, receiving as input a goal recognition model and a set of optimal policies. The algorithm initially builds a cycle-free MDP (lines 39-41) by calling Algorithm 1 with limit $l = 0$ to account for distinctive states as well, and initializes the output variables and a stack (line 42). Then, the 7-step procedure starts. Step 1 is executed in line 43 and the stack is used to find successors in a DFS-fashion (lines 44-68). Each augmented state in the stack is explored to generate its immediate successors (lines 46-47). Following steps 2 and 3, the algorithm finds and augments successors (lines 49-50). The set of possible goals to temporarily augment a successor found in step 3 corresponds to the intersection of its predecessors' goals with the set of goals for which the action executed to arrive at it is optimal (line 52). Next, successors are grouped as specified in step 4 (lines 53-54) and the final set of goals per group is generated (lines 57-58). The algorithm uses this set to augment states in that group (step 5, line 60).

Algorithm 2: AUGMDP-PO($s_0, \mathbf{S}, \mathcal{T}, \mathcal{C}_o, \mathbf{G}, N, \Pi_{\mathbf{G}}$)

$\langle \hat{s}_0, \hat{\mathbf{S}}, \hat{\mathcal{T}}, \hat{\mathbf{G}} \rangle \leftarrow AugMDP\text{-}FO(s_0, \mathbf{S}, \mathcal{T}, \mathbf{G}, 0, \Pi_{\mathbf{G}})$
foreach $\hat{s} \in \hat{\mathbf{S}}$ **do**
  if $\hat{s} = s \cdot \langle pos_1 \dots pos_{|\mathbf{G}|} \rangle = s \cdot \langle \mathbf{G}' \rangle$ **then** $\mathcal{N}(\hat{s}) \leftarrow \mathcal{N}(s)$;
$\mathbf{G}', \mathbf{S}', Stack \leftarrow \emptyset$; $\mathcal{T}' \leftarrow null$
$s'_0 \leftarrow \hat{s}_0 \cdot \langle \hat{\mathbf{G}} \rangle$
$Stack.push(s'_0)$
while $Stack \neq \emptyset$ **do**
  $s' \leftarrow \hat{s} \cdot \langle \hat{\mathbf{G}}' \rangle \leftarrow Stack.pop()$
  $Keys \leftarrow \emptyset; Map \leftarrow null$
  foreach $\hat{\mathcal{T}}(\hat{s}, \pi(\hat{s}), \hat{s}') > 0$ **do**
    $\hat{\mathbf{G}}'' \leftarrow \mathbf{G}(\pi)$
    if $\mathcal{N}(\hat{s}) = \mathcal{N}(\hat{s}')$ **then** $s'' \leftarrow \hat{s}' \cdot \langle \hat{\mathbf{G}}' \rangle$;
    else
      $s'' \leftarrow \hat{s}' \cdot \langle \hat{\mathbf{G}}' \cap \hat{\mathbf{G}}'' \rangle$
      $Keys \leftarrow Keys \cup \{\mathcal{N}(s'')\}$
      $Map(\mathcal{N}(s'')) \leftarrow Map(\mathcal{N}(s'')) \cup \{\langle s', \pi(\hat{s}), s'' \rangle\}$
  foreach $k \in Keys$ **do**
    $Goals \leftarrow \emptyset$
    foreach $\langle s', \pi(\hat{s}), \hat{s}'' \cdot \langle \hat{\mathbf{G}}'' \rangle \rangle \in Map(k)$ **do**
      $Goals \leftarrow Goals \cup \hat{\mathbf{G}}''$
    foreach $\langle s', \pi(\hat{s}), \hat{s}'' \cdot \langle , \hat{\mathbf{G}}'' \rangle \rangle \in Map(k)$ **do**
      $s'' \leftarrow \hat{s}'' \cdot \langle Goals \rangle$
      if $|Goals| > 1$ **then**
        if $s'' \notin S'$ **then** $Stack.push(s'')$;
        $\mathbf{S}' \leftarrow \mathbf{S}' \cup \{s''\}$
        $\mathcal{C}'(s', \pi(\hat{s}), s'') \leftarrow \mathcal{C}_o(\hat{s}, \pi(\hat{s}), \hat{s}')$
      else
        $\mathbf{G}' \leftarrow \mathbf{G}' \cup \{s''\}$
        $\mathcal{C}'(s', \pi(\hat{s}), s'') \leftarrow 0$
      $\mathcal{T}'(s', \pi(\hat{s}), s'') \leftarrow \mathcal{T}(\hat{s}, \pi(\hat{s}), \hat{s}')$
return $(\langle s'_0, \mathbf{S}', \mathcal{T}', \mathbf{G}', \mathcal{C}' \rangle)$

If the newly augmented state was not explored before, it is added to the stack for future exploration (step 6, line 62), and to the set of augmented states if it is distinctive (line 63). If the set of goals per group has less than two goals, the explored state is an augmented goal (lines 65-66). The augmented cost function is updated in both cases (lines 64,67). Finally, the augmented transition function is updated (line 68), and once the algorithm explores all reachable expanded states, it returns all augmented parameters. The resultant augmented MDP contains only *legal augmented policies* (Definition 19, p. 88).

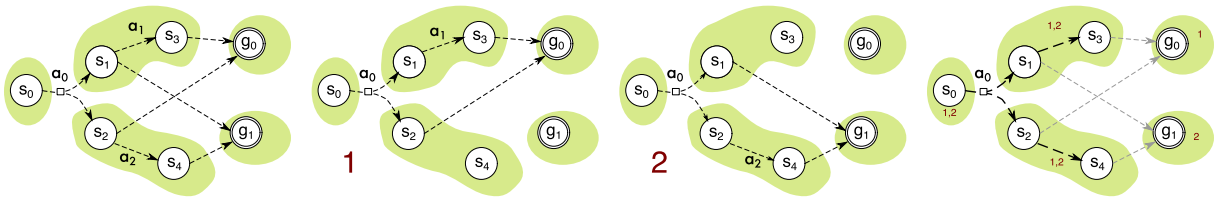
Determining *wcd*

Exactly as in OS-GRDs (Section 4.1, p. 71), the maximum expected cost can be found using a TVI-like algorithm (Dai, D. S. Weld, et al., 2011) to solve the augmented MDP where instead of using Eq. 2.1, we use Eq. 4.4 that replaces the minimization condition with a maximization. Algorithm 2 removes possible *infinite-cost cycles* caused by partial observability; therefore, the resultant augmented MDP does *not* have infinite-cost cycles, and Equation 4.4 will thus return the correct finite value upon convergence. Theorem 3 (p. 73) is valid for POS-GRD as well.

The *wcd* for our example in Figure 4.8 is 1.2, corresponding to the thickest policy satisfying goal g_1.

4.2.4 Using Policy Enumeration in POS-GRD

The last subsections presented a method to compute *wcd* for POS-GRD while avoiding policy enumeration. Although the method is useful in many cases, it can only find an upper bound in cases where the policy with maximal expected distinctiveness has trajectories with lower distinctiveness costs than trajectories of other policies. The following subsections present a method to compute *wcd* in all cases.



(a) (b) (c) (d)

Figure 4.9: POS-GRD Example. (a) Original problem. (b) Optimal policy for goal g_0. (c) Optimal policy for goal g_1. (d) Annotated augmented MDP.

Example 7. *Consider Figure 4.9(a), where dotted arrows represent unobservable actions of unitary costs and nodes denote states. A multi-head arrow depicts the only action with stochastic outcomes and double-lined circles symbolize possible goals. Due to sensor resolution, multiple states can map to one observation, marked as a green shaded area. There is only one policy to reach each goal and each legal policy has two possible trajectories. Figure 4.9(d) highlights non-distinctive observable actions in bold, the graph structure is equivalent to an MDP augmented with goal information as described in Subsection 4.2.1 (p. 83). Gray arrows correspond to actions with cost* 0 *in such augmented MDP.*

Example 7 presents a case where evaluating the maximum expected cost in an augmented MDP provides an upper bound instead of the *wcd* value as specified in Definition 17 (p. 55). Similar to OS-GRD, we decided to enumerate all legal (optimal) policies and keep track of them. Figures 4.9(b) and (c) show policies 1 and 2, optimal for goals g_0 and g_1, respectively. In this case, however, we need to account not only for policies that share the same actions, but also for policies sharing the same *observable trajectories*.

4.2.5 Policy-Aware Augmented MDP for POS-GRD

Let $P = \langle M, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$ with $k = 0$ be a partially-observable goal recognition model with stochastic action outcomes (Definition 1, p. 45), $M = \langle \mathbf{S}, s_0, \mathbf{A}, \mathcal{T}, \mathcal{C}, \emptyset \rangle$ is an MDP with positive costs $\mathcal{C}$ for the agent and no goal. The states of an augmented MDP add a Boolean variable pos^g_ρ per possible policy to keep track of its validity given an observed trajectory. Note that each policy with ID ρ also serves to keep track of possible goals since they are legal for a specific goal g. Terminal states of this MDP are those that have less than two possible goals, with transitions and costs defined according to the original MDP. To comply with Definition 17 (p. 55), the cost of an action transitioning to a terminal state is 0.

The policy-aware augmented MDP for POS-GRD $\Pi_{aug} = \langle \mathbf{S}', s'_0, \mathbf{A}', \mathcal{T}', \mathcal{C}'_o, \mathbf{G}' \rangle$ is defined as follows:

- $\mathbf{S}' = \mathbf{S} \times \{T, F\}^{|\Pi_{\mathbf{G}}|}$: for each $s \in \mathbf{S}$ we create $2^{|\Pi_{\mathbf{G}}|}$ augmented states, corresponding to all subsets of possible legal policies. We use $w(s') = s$ to denote that s is the state of the world at $s' \in \mathbf{S}'$.

- $s'_0 = s_0 \cdot \langle T \dots T \rangle$: initially all policies are possible.

- $\mathbf{A}' = \mathbf{A}$ (action labels remain unchanged).

- $\mathcal{T}'(s \cdot \langle pos_1^1 \dots pos_\rho^g \rangle, a, s' \cdot \langle pos_1'^1 \dots pos_\rho'^g \rangle) =$

$$
\begin{cases}
\mathcal{T}(s,a,s') & (\exists i \neq j, m \neq n : pos_i^m = pos_j^n = T) \wedge & (4.18)\\
& \forall i \in \{1 \dots, |\Pi_{\mathbf{G}}|\}(pos_i'^g = (pos_i^g \wedge & (4.19)\\
& \big(\mathcal{N}(s) = \mathcal{N}(s') & (4.20)\\
& \vee(\pi_i \in \Pi_g^k \wedge \pi_i(s) = a \wedge id(\pi_i) = i) & (4.21)\\
& \vee(\exists \pi \in \Pi_{\mathbf{G}} \wedge \exists \hat{s} : \mathcal{N}(s) = \mathcal{N}(\hat{s}) \wedge & (4.22)\\
& \exists \hat{s}' \mid \mathcal{T}(\hat{s}, \pi(\hat{s}), \hat{s}') > 0 \wedge & (4.23)\\
& \mathcal{N}(s') = \mathcal{N}(\hat{s}'))))) & (4.24)\\
0 & \text{otherwise} & (4.25)
\end{cases}
$$

where $id : \Pi_{\mathbf{G}} \to \mathbb{Z}^+$ is a function mapping legal policies to policy IDs. The probability of transitioning from state s with $\langle pos_1^1 \dots pos_\rho^g \rangle$ to state s' with $\langle pos_1'^1 \dots pos_\rho'^g \rangle$ (where $pos_i^m, pos_i'^m$ indicate whether policy with ID i, legal for goal m is possible) depends on multiple factors. The transition probability from a state where the true goal was revealed is 0 (Line 4.18). Once discarded, a policy cannot become possible (Line 4.19).

A policy remains possible if s' is *unobservably connected* (Definition 18, p. 84) to s (Line 4.20) or action a is part of policy π_i with ID i (Line 4.21). Finally, Lines 4.22-4.24 cover the case of undistinguishable actions discussed for our example and, in more detail, in Subsection 4.2.1 (p. 84).

- $\mathcal{C}'(s \cdot \langle pos_1^1 \dots pos_\rho^g \rangle, a, s' \cdot \langle pos_1'^1 \dots pos_\rho'^g \rangle) = \begin{cases} \mathcal{C}_o(s, a, s') & \forall (s' \cdot \langle pos_1'^1 \dots pos_\rho'^g \rangle) \notin \mathbf{G}' \\ 0 & \text{otherwise} \end{cases}$
  We want to find policies with maximal cost for the observer without including the cost of actions that transition to a terminal state, and

- $\mathbf{G}' = \{ s \cdot \langle pos_1^1 \dots pos_\rho^g \rangle \mid (\exists pos_i^m, \forall j \neq i, n \neq m : pos_j^n = F) \}$: terminal states are those with less than two possible goals.

Figure 4.9(d) shows the resultant policy-aware augmented MDP where states are augmented with policy IDs (signaled by red numbers, as defined in Figures 4.9(b) and (c)).

4.2.6 Computing *wcd*: Algorithms

The computation of *wcd* for POS-GRD implies finding legal policies and accounting for the observer's partial observability as defined at the beginning of this section. We find and mark all legal (optimal) policies per candidate goal. Next, we use Procedure CommonNDPoliciesPO to find groups of policies sharing all their observable non-distinctive trajectories and to build a partial augmented MDP to evaluate them. In this Procedure, $\mathbf{N}$ represents the set of possible observations given a sensor configuration defined by the function $\mathcal{N}$. Function

$\mathcal{O} : \mathbf{N} \rightarrow P(\mathbf{S}')$ is a mapping from $\mathbf{N}$ to the power set of augmented states that models augmented states grouped by their projected observations.

Procedure CommonNDPoliciesPO($\rho, \mho, s_0, \mathbf{S}, \Pi^{\mho}_{\mathbf{G}}, \mathcal{T}, \mathcal{N}$)

```
70 Stack ← ∅; S' ← ∅; ℧_i ← ℧, T' ← null, G' ← ∅; O ← null
71 s_0^℧ ← ⟨s_0, ℧⟩
72 Stack.push({s_0^℧}); S' ← S' ∪ {s_0^℧}
73 while Stack ≠ ∅ do
74     S_0 ← Stack.pop
75     ⟨℧_i, T', G', S', O⟩ ← UnobsConnStates(S_0, T, N, ρ, ℧_i, T', G', S')
76     foreach N(s) ∈ N | O(N(s)) ≠ null do
77         S'' ← O(N(s))
78         Stack ← S''
79 return ⟨℧_i, S', T', G'⟩
```

At a high level, the procedure represents each unobservably connected set of states as a node and traverses these nodes in a DFS fashion. It receives as arguments the policy ID ρ to be evaluated, the set $\mho$ of all policy IDs, the initial state s_0, the set $\mathbf{S}$ of original states, the set $\Pi^{\mho}_{\mathbf{G}}$ of all legal policies marked with their respective IDs, the original transition function $\mathcal{T}$, and the sensor configuration modeled by the sensor function $\mathcal{N}$. First, all variables are initialized and the start state s_0 is augmented with the set of all policy IDs $\mho$ (Lines 70-71). Next, a set containing the augmented initial state is pushed to a stack and the set of augmented states is updated (Line 72). Each stack entry is a set of augmented states emitting the same observation and whose predecessors emit a *different* observation. The Procedure UnobsConnStates updates the augmented MDP components and the set of policies $\mho_i$ that share all observable trajectories of policy with ID ρ while traversing unobservably connected

sates (Lines 73-78). Finally, the procedure returns $\mho_i$ and the components of an augmented MDP useful to evaluate the group of policies with IDs in $\mho_i$ (Line 79).

The solution of the augmented MDP generated using Procedure CommonNDPoliciesPO gives the expected cost of the largest non-distinctive prefix of all policies $\Pi^{\mho_i}$ with IDs indicated by $\mho_i$. The *wcd* is then computed using:

$$wcd(P) = \max_{i=1...n} V_{\Pi^{\mho^i}}(s'_0) \tag{4.26}$$

$$V_{\Pi^{\mho^i}}(s') = \sum_{s'' \in \mathbf{S}'} \mathcal{T}'(s', \pi(s'), s'')[\mathcal{C}'(s', \pi(s'), s'') + V_{\Pi^{\mho^i}}(s'')] \tag{4.27}$$

where: $\pi \in \Pi^{\mho^i}$ and $\bigcup_{i=1}^{n} \mho^i = \mho \wedge \bigcap_{i=1}^{n} \mho^i = \emptyset$

We use a TVI-like algorithm that runs iterations of Eq. 4.27 for groups of policies with the same non-distinctive trajectories, and find the maximum among all using Eq. 4.26.

4.3 Suboptimal S-GRD (SS-GRD)

Previous sections consider the worst-case distinctiveness measure under the assumption of optimal agents. However, even if the agent is rational, it may not act optimally in many real-world scenarios. Moreover, as in any other design optimization problem, the effectiveness of GRD depends on accurately modeling the agent's interaction with the environment. Many intelligent agents use human prediction models, studied in psychology and economics, among other fields (Rosenfeld and Kraus, 2018). Nonetheless, there is no consensus on the degree of

human rationality. There are multiple examples of studies supporting both theories (Rahnev and Denison, 2018), including the work of two Nobel laureates in 2002 showing the opposite results (Rosenfeld and Kraus, 2018).

Assuming an utterly irrational agent might cause an infinite *wcd*. Additionally, it is unlikely that an agent with a goal or purpose presents such behavior. Therefore, we decided to consider a boundedly rational agent. Specifically, we find a set of policies with up to k suboptimal actions that acting agents could take. Like humans, artificial agents cannot account for every aspect of world dynamics, and it is also problematic to optimize a decision for multiple objectives such as utility, risk, or other preferences. Therefore, accounting for slightly suboptimal policies arguably improves the model since there is a higher probability that one of them will be the best for all criteria. This decision also aligns with work on goal recognizers using top-k planners to improve goal recognition in domains with unreliable observations (Riabov et al., 2020; Sohrabi et al., 2016) (reviewed in Subsection 2.3.1, p. 26).

In this section, we propose algorithms to compute *wcd* for settings where the agent is *boundedly rational*. The problem we call Suboptimal S-GRD (SS-GRD) assumes an agent with suboptimal behavior and both agent and observer with full observability.

4.3.1 Handling Suboptimality

Let $P = \langle M, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$ be a suboptimal stochastic GR model (Definition 1, p. 45), where M is an MDP with positive costs and no goal; $\mathbf{G}$ is a set of possible goals of M; $k \geq 0$

represents the maximum number of *suboptimal actions per policy*; $\mathcal{N}$ in this case models an ideal sensor that allows full observability; and $\mathcal{C}_o$ is the cost function that represents costs for the observer every time an agent executes an action.

In Sections 4.1 and 4.2, we observed that the set of possible goals for a particular state in stochastic environments is not Markovian as it depends on the observed trajectory of the agent to that state. When considering a set of allowed or *legal* suboptimal policies, we need to keep track not only of the possible goals, but also of the possible followed policies.

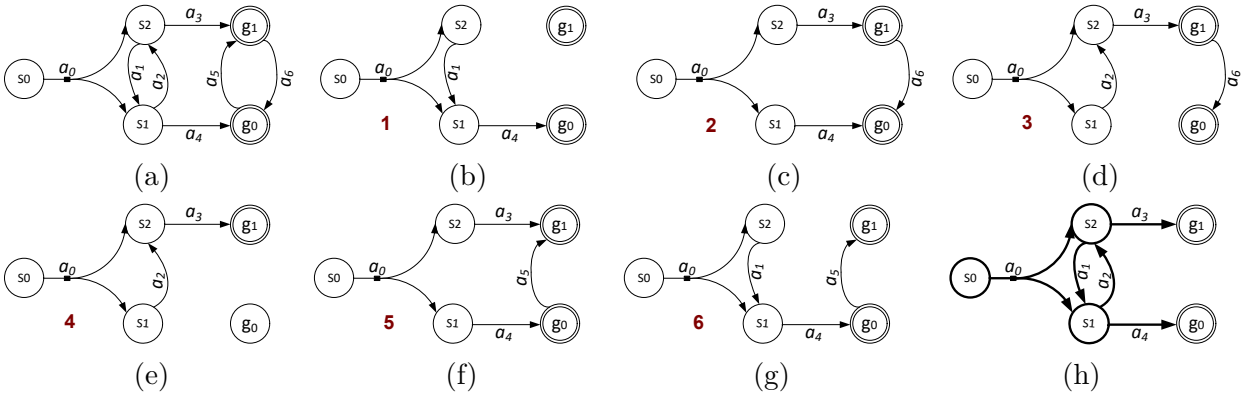


Figure 4.10: Running example for SS-GRD: (a)Original problem. (b - g) All possible proper policies. (b,c) Optimal policies for g_0. (d) Suboptimal policy for g_0. (e,f) Optimal policies for g_1.(g) Suboptimal policy for g_1. Red numbers represent the policy ID. (h) Non-distinctive actions.

Example 8. *To illustrate the SS-GRD setting, consider Figure 4.10(a), where the start state is s_0 and there are two possible goals: g_0 and g_1. All actions are deterministic except for action a_0, which has the same probability to transition from s_0 to states s_1 or s_2. The cost of executing an action is 1. Figures 4.10(b)–4.10(g) enumerate all possible* proper *policies an agent could take to reach either goal g_0 or g_1. A red number identifies each policy. In*

this example, $k = 1$ as there is only one suboptimal policy per goal. Figure 4.10(h) shows all non-distinctive actions, and it also corresponds to an augmented MDP that keeps track of goals as in the OS-GRD case. Note that actions a_1 and a_2 form an infinite loop, where a_1 is optimal for goal g_0 and suboptimal for goal g_1 (Figures 4.10(b) and (g)), and a_2 is optimal for goal g_1 and suboptimal for goal g_0 (Figures 4.10(d)and (e)). Therefore, the policy with the largest expected cost of an augmented MDP as shown in Figure 4.10(h) will not provide the wcd *value.*

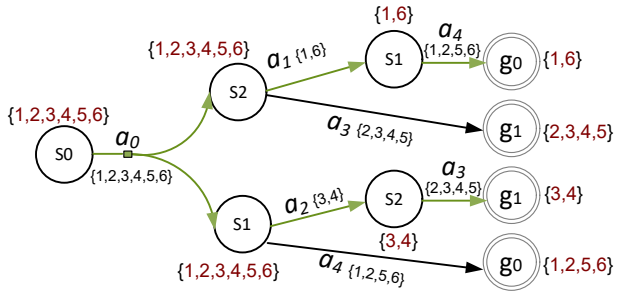


Figure 4.11: Policy Tracking: MDP of running example augmented with policy IDs. Circles in bold represent states with two possible goals and bold arrows denote non-distinctive actions. The largest trajectories marked in green correspond to different sets of policies

Since actions forming an infinite-cost loop cannot be part of any proper static policy, keeping track of the policies breaks those loops. However, augmenting states with policy IDs may create maximal augmented policies that do not correspond to any legal policy. For instance, Figure 4.11 shows the resultant MDP of our running example after augmenting states and actions with policy IDs. The policy marked in green highlights the largest augmented policy; note that each trajectory maps to different pairs of policy IDs ($\{1, 6\}$ and $\{3, 4\}$). Due to this problem, we evaluate the non-distinctive part of a policy only in the augmented space

reachable by that policy. To evaluate policy 1, for instance, we do not consider augmented states $\{\langle S2, \{3,4\}\rangle, \langle g_1, \{3,4\}\rangle, \langle g_1, \{2,3,4,5\}\rangle\}$.

4.3.2 Augmented MDP for SS-GRD

An augmented MDP for SS-GRD is almost identical to the policy-aware augmented MDP described in Subsection 4.1.6 with the only difference that $k \geq 0$ defining the maximum number of suboptimal actions allowed.

4.3.3 Computing *wcd*: Algorithms

We first provide a high-level description of our approach to compute *wcd* for SS-GRD problems. Initially, all legal policies per goal are identified by finding all policies with up to k suboptimal actions. The implied assumption here is that the policies are stationary and proper, that is, the agent will not change policy until its goal is reached. Next, we proceed as in the OS-GRD case with the policy enumeration approach; that is, we evaluate the expected distinctiveness of each policy using its reachable augmented state space, avoiding to recompute policies that share its entire non-distinctive prefix with an already assessed policy. Finally, the *wcd* is the largest expected cost found.

Finding Policies with up to k Suboptimal Actions

Inspired by the algorithm to find the best k policies (Dai and Goldsmith, 2010) (discussed in Subsection 2.2.6, p. 22), we propose Algorithm 3 to find all policies with up to k suboptimal actions. As Dai and Goldsmith (2010) mention, there are usually multiple "trivially extended policies" that differ from another only in a non-reachable state, which is not useful in our case. Therefore, we make sure that policies differ from the previous one in *one* of their reachable states.

Algorithm 3 shows the procedure of finding and marking all legal policies for one goal. At a high level, it first finds all optimal policies and marks the one that has all actions with the lowest lexicographical order and only in the reachable states. Next, it finds and marks a policy that differs in *one reachable state*. Finding (and marking) the next policy from an already marked policy π requires (1) Fixing a prefix that reaches a state s from the starting state s_0 following π; (2) Changing to the next available action a in s (where optimal actions are prioritized), and (3) Following policy π (as much as possible) from there. In case the new policy reaches a state not reached by π, the algorithm chooses an optimal action with the lowest lexicographical order. Note that forcing two policies to differ in exactly one action can cause infinite loops if there are suboptimal actions. The algorithm detects and does not consider policies with infinite loops.

We now describe the pseudocode in more detail. Algorithm 3 receives as parameters the original MDP, one of the possible goals, and the maximum number of suboptimal actions

Algorithm 3: LEGPOL($M = \langle s_0, \mathbf{S}, \mathbf{A}, \mathcal{T}, \mathcal{C} \rangle, g, k$)

```
80 n ← 0; p ← 0; Π_g ← ∅
81 (V(s_0), Π*) ← VI'(M, g)
82 (•, π_g^0 | π_g^0 ∈ Π*, π_g^0(s_0)) ← mark(•, s_0, •, p, n)
83 S_π ← reach(s_0, π_g^0)
84 Q ← (s_0, π_g^0(s_0), p); p++
85 while Q ≠ ∅ do
86     (s, a_i ← π_g^i(s), i) ← Q.dequeue
87     if S_π = ∅ then S_π ← reach(s, π_g^i);
88     foreach s_π ∈ S_π do
89         π̃ ← prefix(s_π, π_g^i)
90         do
91             (n, π_g^p, •) ← mark(π̃, s_π, a_i, p, n)
92             if π_g^p ≠ null then Q ← (s_π, a_i, p); p++ ;
93             a_i ← a_{i+1}
94         while (a_{i+1} ∈ A(s_π));
95     S_π ← ∅
```

allowed. After variable initialization, the algorithm finds all optimal policies and their expected cost using a modified version of VI (Lines 80-81). It then marks the first optimal policy (π_g^0) and retrieves the action at the initial state s_0 used in this policy (Line 82). Next, the algorithm retrieves all reachable states following π_g^0, stores the initial state, the retrieved action, and the current policy ID in a queue, and increases the counter p of policies (Lines 83-84). The function *reach* finds the reachable states starting from the received state. The enqueued information is useful to find the next legal policy: the state-action pair (s, a_i) signals the point from which the new policy can deviate, and the policy ID i signals the policy used as reference. The algorithm then finds the set of reachable states if necessary (Lines 85-87) and each reachable state is used as a next point of deviation (Line 88). For each reachable state s_π, the algorithm finds the prefix of policy π_g^i until s_π and uses function *mark* to (1)find the next state-action pair to consider, (2)find the policy that follows that

state-action pair and optimally reaches g, and (3) mark the new policy with ID p in case it is a proper policy that contains at most n suboptimal actions. If a policy was successfully marked, the algorithm enqueues the state-action pair found in point (1) together with the policy ID used and increases the policy counter (Lines 89-92). New policies are marked using the same prefix while there are available actions applicable in state s_π (Lines 93-94). Finally, the set of reachable states is cleared before reading the next entry of the queue (Line 95). When Algorithm 3 finishes, all actions to reach goal g were marked with the set of legal policies that use them.

Note that the policies found are *proper*, that is, they reach the goal with probability 1. Therefore, these policies do not contain infinite-cost loops. In our example of Figure 4.10(a), the policy formed by actions a_0, a_1, a_2 is not proper as it does not reach a goal.

Computing *wcd*: Practical Considerations

As stated before, the highest expected cost at the starting state of an augmented MDP for SS-GRD is not equivalent to the *wcd* of the problem. Therefore, we need to evaluate the non-distinctive prefix of every legal policy. There is no need to generate all $2^{|\Pi_G|}$ augmented states, just the reachable states using the policy to be evaluated. While building this smaller augmented MDP, we keep track of all policies that share all non-distinctive trajectories and group the ones sharing their maximum non-distinctive prefixes; the *wcd* is equivalent to the maximum expected cost among all groups. For instance, in our running example, Figure 4.10 shows that policies 1 and 6 share all their non-distinctive trajectories, thus, we will have one

computation for both policies. In the worst case, we will need to evaluate individually all $\mid \Pi_{\mathbf{G}} \mid$ policies.

Similar to OS-GRD, we use Procedure CommonNDPoliciesFO (p. 80) to group augmented policies for evaluation and to build the partial augmented MDP. The solution of the augmented MDP generated using this procedure gives the expected cost of the largest non-distinctive prefix of all policies $\Pi^{\mho_i}$ with IDs indicated in $\mho_i$.

We use a TVI-like algorithm that runs iterations of Eq. 4.6 (p. 81) for groups of policies with the same non-distinctive trajectories, and find the maximum among all using Eq. 4.5 (p. 81) .

4.4 Partially-Observable Suboptimal S-GRD (POSS-GRD)

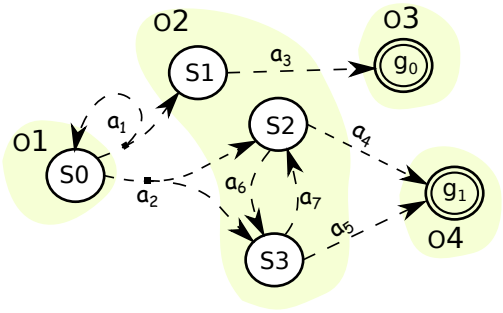


Figure 4.12: POSS-GRD Running Example: Original MDP

POSS-GRD is the most general model of our framework, it combines the assumptions of POS-GRD and SS-GRD models. POSS-GRD assumes a boundedly rational agent and an

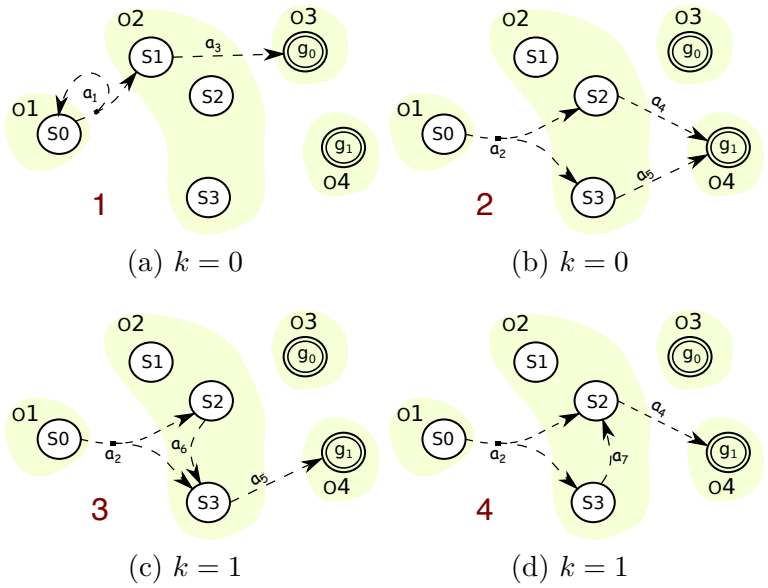
observer that does not perceive executed actions, just current states. Further, due to sensor resolution, some states cannot be distinguished from each other.

Example 9. *To illustrate, consider Figure 4.12, where nodes represent states, dashed arrows denote non-observable actions, and green bubbles stand for observations (*$O1$ *through* $O4$*) that group undistinguishable states. All actions have a cost of execution equal to* 1*; agents executing action* a_1 *have 90% probability to succeed and agents executing action* a_2 *have the same probability of transitioning in states* s_2 *or* s_3*. An observer only perceives sequences of observations, and the acting agent could execute one of the multiple legal trajectories. In our example, the sequence* $\langle O1, O2 \rangle$*, will not provide any new information. An observer will not know whether the agent executed action* a_1 *or* a_2*, nor could she discern whether the agent transitions between states* $S2$ *and* $S3$*.*

Remember that due to the offline property of GRD problems, we need to account for *all* possible sequences of observations generated by a rational agent. Moreover, we require that an agent selects only proper policies, and assume the policies are stationary. Therefore, policies including actions a_3 and a_4 or actions a_7 and a_8 are not allowed.

Figure 4.13 shows all possible proper policies for each goal, which contain up to $k = 1$ suboptimal actions. Policy 1 is optimal for goal g_0 and policy 2 is optimal for goal g_1. Policies with ID 3 and 4 are suboptimal policies for goal g_1.

Similar to other S-GRD models, the set of goals that are still valid at a given state depends on the observed trajectory, thus, we need to keep track of possible goals. However, our



(a) $k = 0$ (b) $k = 0$

(c) $k = 1$ (d) $k = 1$

Figure 4.13: POSS-GRD(cont.): All possible proper policies. (a) Optimal policy for g_0. (b) Optimal policy for g_1. (c, d) Suboptimal policies for g_1. Red numbers represent the policy ID.

assumptions of partial observable states can create conditions of infinite-cost loops when all non-distinctive policies are considered together, thus, eliminating the possibility to compute the *wcd* by evaluating one augmented MDP. Figure 4.14(a) shows all non-distinctive actions and observations, keeping track of the possible goals (shown as subindices). The largest expected cost at the initial state in such MDP is infinite due to the loop formed by actions a_6 and a_7.

Since suboptimality requires to evaluate policies individually, we can proceed exactly as for POS-GRD when using policy enumeration. We need to account not only for policies that share the same actions, but also for policies sharing the same *observable trajectories*. For instance, consider Figure 4.13. When evaluating the ambiguity of policy 1, no legal policy for goal g_1 shares actions a_1 or a_3. However, all other policies share part of its observable

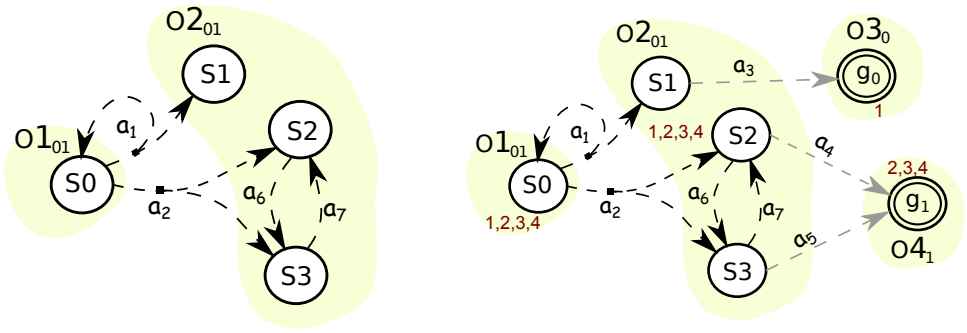


Figure 4.14: POSS-GRD(cont.). (a) Non-distinctive observations. (b) Augmented MDP

trajectory ($\vec{\tau} = \langle O1, O2 \rangle$). To improve the naïve approach, we can find groups of policies that share their largest ambiguous prefixes and, to avoid infinite loops, consider only policies that share actions. For example, in Figure 4.13, we cannot evaluate together policies 3 and 4.

4.4.1 Augmented MDP for POSS-GRD

An augmented MDP for POSS-GRD is almost identical to the policy-aware augmented MDP described in Subsection **??** for POS-GRD. The only difference is that $k \geq 0$ to account for the number of suboptimal actions allowed.

Figure 4.14(b) shows the resultant augmented MDP for $k = 1$ where states are augmented with policy IDs (signaled by red numbers, as defined in Figure 4.13). Augmented goals ($g_0 \cdot \langle T, F, F, F \rangle$ and $g_1 \cdot \langle F, T, T, T \rangle$) are terminal and the cost of the last actions to reach them is 0, indicated by gray arrows.

4.4.2 Computing *wcd*: Algorithms

The computation of *wcd* for POSS-GRD implies finding legal policies and accounting for the observer's partial observability as defined for POS-GRD models. We use Algorithm 3 (p. 111) to find legal policies per candidate goal. Each policy contains at most k suboptimal actions to account for boundedly suboptimal agents. Next, we proceed exactly as in POS-GRD when using policy enumeration:

We use Procedure CommonNDPoliciesPO (p. 104) to find groups of policies sharing all their observable non-distinctive trajectories and to build a partial augmented MDP to evaluate them. The solution of the resultant augmented MDP gives the expected cost of the largest non-distinctive prefix among all policies. A TVI-like algorithm runs iterations to solve Eq. 4.27 (p. 105) for groups of policies with the same non-distinctive trajectories, and the *wcd* value corresponds to the maximum among all (Eq. 4.26, p. 105).

Chapter 5

Design: Minimizing *wcd*

"Design can be art. Design can be simple.

That's why it's so complicated."

– Paul Rand

The solution procedure of an S-GRD problem has two stages: (1) Computing a measure to evaluate the initial stochastic GR problem; and (2) Optimizing the design by finding a minimal set of modifications that minimize the chosen measure. Chapter 4 focused on the first stage and provided methods to compute *wcd* for each model in our framework. This chapter analyzes the optimization problem of changing the environment to support a faster GR, which corresponds to the second stage of the solution.

Keren, A. Gal, et al., 2018 cast the design process as a graph search in the space of modification sequences, where the root denotes the original GR model, each other node in the graph represents a modified GR model. Direct edges connect vertices differing in only one modification, where tail nodes are modified head node versions. A basic approach to solve this search problem is to traverse the graph using BFS as explained in Chapter 2 (p. 35). This process is equivalent to enumerate all sequences of modifications and evaluate the model described by each sequence, that is, compute *wcd* for each case. Our design model $\mathcal{D} = \langle \mathcal{M}, \delta, \phi, \mathcal{C}_m, \mu \rangle$ (Definition 2, p.46) contemplates μ, a user-defined parameter that specifies the maximum number of allowed modifications, and would limit the number of levels reached by the BFS traversal. Similar to GRD problems, we have the constraint that a valid sequence of modifications should not increase the original cost of reaching a goal.

We consider two types of modifications, namely action removal (applicable to all models), and sensor refinement for partially-observable settings. For these types of modifications, the order in a sequence is not important.

5.1 Action Removal (AR)

Action removal was first proposed by Keren, A. Gal, et al., 2014 as a method to modify GRD models. The naïve approach consists of removing each combination of up to μ state-action pairs and computing the measure (*wcd*) for each of the resultant GR models to find the set of removed actions that minimize its value. However, the number of state-action pairs is

usually so large in mid-size instances, that even $\mu = 2$ can generate an unfeasible number of combinations. For simplicity in this Chapter, we will refer to state-action pairs as actions.

Consider Figure 5.1(b), which shows an augmented MDP for the problem represented in Figure 5.1(a) when $k = 1$. The naïve approach for $\mu = 2$ will examine $\binom{7}{1} + \binom{7}{2} = 28$ possibilities. We use Lemmas 9-12 as heuristics to prune the search space of removed actions.

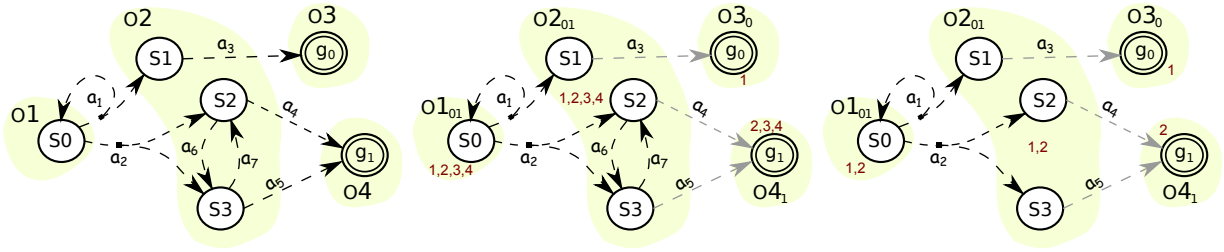


Figure 5.1: Action removal in POSS-GRD: (a) Original problem: action a_1 has 90% probability of success and action a_2 has 50% probability to reach states $S2$ or $S3$. (b) Augmented MDP of the original problem (wcd=1.5). (c) Augmented MDP after removing actions a_6 and a_7 (wcd=1.1). Grey arrows have a cost of 0.

Lemma 9. *Removing actions not used in a legal policy does not reduce* wcd.

Proof. The *wcd* of an S-GRD model is defined for legal policies ($\pi \in \Pi_{\mathbf{G}}$) (Definition 17, p. 55). Therefore, removing action $\pi(s)$ where $s \in \mathbf{S}$ and $\pi \notin \Pi_{\mathbf{G}}$, will not affect its value. ■

Hence, it is safe to consider the removal of only legal actions.

Lemma 10. *If removing action $\pi(s)$ causes an unreachable goal using a legal policy, then removing any modification sequence containing $\pi(s)$ will also cause an unreachable goal.*

Proof. If there is no legal policy that can reach a goal g , then the expected cost c to reach g is infinite. Additionally, removing an action cannot reduce the expected cost to reach a goal. Thus, removing a second action cannot change the infinite cost. ■

The next corollary extends Lemma 10 to a set of removed actions.

Corollary 2. *If removing a set of actions* $\mathbf{A}'$ *causes an unreachable goal, then removing any modification sequence containing all actions in* $\mathbf{A}'$ *will cause an unreachable goal too.*

Lemma 11. *Let* $\hat{\mathbf{S}}$ *be the set of unreachable states after removing action* $\pi(s)$*, and* $\hat{\mathbf{A}} = \{\hat{\pi}(\hat{s}) \mid \hat{s} \in \hat{\mathbf{S}} \wedge \hat{\pi} \in \Pi_{\mathbf{G}}\}$ *be the set of legal actions applicable in those unreachable states. Then, removing* $\pi(s)$ *and* $\hat{\pi}(\hat{s})$ *is not different than removing* $\pi(s)$ *alone.*

Proof. Actions applicable in unreachable states do not contribute to any policy's expected cost; therefore, removing them will not cause any change. ■

Lemma 11 implies that we can prune combinations of one action with any legal action of unreachable states. We extend this Lemma to account for sets of removed actions in the following corollary:

Corollary 3. *Let* $\hat{\mathbf{S}}$ *be the set of unreachable states after removing the set of actions* $\mathbf{A}'$*, and* $\hat{\mathbf{A}} = \{\hat{\pi}(\hat{s}) \mid \hat{s} \in \hat{\mathbf{S}} \wedge \hat{\pi} \in \Pi_{\mathbf{G}}\}$ *be the set of legal actions of those unreachable states. Then, removing* $\mathbf{A}' \cup \{\hat{\pi}(\hat{s})\}$ *is not different than removing* $\mathbf{A}'$ *alone.*

Corollary 3 implies that removing any superset $\tilde{\mathbf{A}}$ of $\mathbf{A}' \cup \{\hat{\pi}(\hat{s})\}$ has the same effect of removing only $\tilde{\mathbf{A}} \backslash \{\hat{\pi}(\hat{s})\}$. Therefore, we can safely prune those supersets.

We can prune the search space a bit further using Lemma 12 below, where we use $\Pi_{dis}(P)$ to represent the set of distinctive policies that do not have ambiguous prefixes and $msr(P)$ to indicate a measure to evaluate P. Lemma 12 states that removing those distinctive policies does not affect the value of that measure, for instance, *wcd*.

Lemma 12. *Let P' and P^n, be two stochastic GR models s.t. $P' = \delta(m_{n-1}, \ldots, \delta(m_1, P_0))$ and $P^n = \delta(m_n, P')$, where P_0 is the original stochastic GR setting and m_n is the action to remove in P'. If $\Pi_{dis}(P') = \{\pi \big| (\pi(s_0)$ is observable $\wedge\, |\mathbf{G}(\pi(s_0))| < 2) \vee (\pi(s_0)$ is not observable $\wedge\, \forall s' \in \mathbf{S}, \vec{\tau} = \langle s_0, \pi(s_0), s' \rangle : |\mathbf{G}(obs(\vec{\tau}))| < 2\}$ and $\Pi_m(P') = \{\hat{\pi} \mid \forall s \in \mathbf{S} : \hat{\pi}(s) = m_n\}$, then $\Pi_m(P') \subseteq \Pi_{dis}(P') \implies msr(P^{\mathbf{M}}) = msr(P')$.*

Proof. By Eq. 3.10 in Definition 17 (p. 55), $wcd(P')$ is the maximum expected cost over non-distinctive policies. Therefore, a policy $\pi \in \Pi_{dis}(P')$ is never considered to compute *wcd*. Further, if all policies using action m_n are distinctive, removing it will not change $wcd(P')$. Any other measure defined only for non-distinctive policies will have the same property. ■

ReduceMeasure-AR uses Lemmas 9-12 and Corollaries 2-3 to prune the search space. The algorithm receives as parameters the original stochastic GR problem P_0 and the maximum number of actions to be removed μ. First, it evaluates the original stochastic GR problem using the selected measure (retrieving in the process the set of legal policies $\Pi_{\mathbf{G}}$ and the set of distinctive policies $\Pi_{dis}(P_0)$) (Line 96). Next, the algorithm initializes some variables: (1) $\bar{\mathcal{M}}$, denoting a set of groups of actions that can be pruned (as well as any set of actions containing any of those groups); (2) $\mathbf{M}^*$, the set of actions that need to be removed to optimize the

Algorithm 4: ReduceMeasure-AR(P_0, μ)

```
96  ⟨msr, Π_G, Π_dis(P_0)⟩ ← computeMsr(P_0)
97  M̄ ← ∅; M* ← ∅; Π ← Π_G \ Π_dis(P_0); Π̄ ← ∅
98  Q ← enqueue(⟨{∅}, Π⟩)
99  S_Π ← reachable(P_0)
100 while Q ≠ ∅ do
101     ⟨M, Π⟩ ← dequeue(Q); Π ← Π \ Π̄
102     foreach s ∈ S_Π, π ∈ Π | π(s) ∉ M do
103         M ← M ∪ {π(s)}
104         if ∀M_f ∈ M̄ : M ⊉ M_f then
105             P_0^M ← δ(M, P_0)
106             S_π̂ ← reachable(P_0^M)
107             if ∃g ∈ G | [g ∉ S_π̂ ∨ unrch(g, P_0^M) = ⊤] then
108                 if | M |= 1 then Π̄ ← Π̄ \ {π | π(s) ∈ M};
109                 else M̄ ← M̄ ∪ {M};
110             else
111                 S_u ← S_Π \ (S_Π ∩ S_π̂)
112                 foreach s ∈ S_u, π̂(s) ∈ A do
113                     M̄ ← M̄ ∪ {M ∪ {π̂(s)}}
114                 ⟨msr_t, •, •, Π_dis(P^M)⟩ ← computeMsr(P_0^M)
115                 if | M |< μ then Q ← enqueue(⟨M, Π_G \ Π_dis(P^M)⟩);
116                 if msr > msr_t then
117                     msr ← msr_t
118                     M* ← M
119 return ⟨msr, M*⟩
```

design, i.e., to minimize the measure; (3) A set Π containing valid policies; (4) A set $\bar{\Pi}$ of policies that can be removed (Line 97); (5) A queue Q initialized with a set containing an empty set representing the original model P_0, and with the set of valid policies (Line 98); and (6) $\mathbf{S}_\Pi$ with the set of reachable states using all valid policies (Line 99). For each set of actions in the queue, ReduceMeasure-AR adds a new action and evaluates whether the resultant set $\mathbf{M}$ is a superset of any set of actions in $\bar{\mathcal{M}}$, i.e., it evaluates if $\mathbf{M}$ should be pruned (Lines 100-104). If $\mathbf{M}$ is valid, then the algorithm removes all actions in $\mathbf{M}$ from the

original GR model (Line 105) and checks whether all goals are reachable (Lines 106-107). Two things may happen if a goal is not reachable: (1) If the removed set contains only one action, then all policies using that action are marked to be removed from the set of valid policies in future iterations (Lines 108 and 101); this is because the number of possible combinations of up to μ actions maybe so large, that even just checking for their validity is impossible when $\mu > 2$. (2) If the removed set contains more than one action, $\mathbf{M}$ and all its supersets can be pruned the next iterations (Line 109). If all goals are reachable in the modified model $P_0^{\mathbf{M}}$, then the algorithm marks all sets containing all actions in $\mathbf{M}$ and one legal action of each unreachable state to be pruned in future iterations (Lines 110-113). Basically, we can prune every combination of actions with actions applicable in unreachable states (Corollary 3). The measure for the modified model is computed (Line 114), then every set containing less than μ actions paired with the set of policies that if removed could reduce the measure are enqueued (Line 115). Then, the minimum value of the measure found so far, together with the set of actions removed to find it are saved (Lines 116-118). Finally, the algorithm returns the minimum value of the selected measure (*wcd*in this case) and the set of removed actions that optimize the design (Line 119).

5.1.1 Optimization for *wcd* Reduction with Action Removal when Using Policy Enumeration

Algorithm 4 makes clear that the time complexity of solving an S-GRD problem is dominated by the design. Therefore, we need to prune the number of state-action pairs as much as possible.

Note that the removal of action a effectively removes all policies that use a. If the largest non-distinctive policy is not a sub-policy of a removed policy, then *wcd* will not reduce. Lemma 13 formalizes this notion using $\Pi_{wcd}(P)$ to represent the set of policies in P that contribute to *wcd*, and $\mathbf{S}_\pi$ to denote the set of reachable states using policy π.

Lemma 13. *Given two stochastic GR models P' and $P^{\mathbf{M}}$ s.t. $P' = \delta(m_{n-1}, \ldots, \delta(m_1, P_0))$ and $P^{\mathbf{M}} = \delta(m_p, \ldots, \delta(m_n, P'))$, where P_0 is the original stochastic GR setting and $\mathbf{M} = \{m_n, \ldots, m_p\}$ is the set of actions to remove from P', we have that $\forall \pi \in \Pi_{wcd}(P'), s \in \mathbf{S}_\pi : \pi(s) \notin \mathbf{M} \implies \text{wcd}(P^{\mathbf{M}}) \geq \text{wcd}(P')$.*

Proof. $P^{\mathbf{M}}$ is a modified model from P', thus, every policy removed from P' does not exist in $P^{\mathbf{M}}$. By Eq. 3.9 in Definition 17 (p. 55), $wcd(P')$ is defined as the maximum expected cost over non-distinctive policies in P'; if none of those policies is removed, the *wcd* will not reduce. ■

Remarks: Lemma 13 considers GR problems derived one from the other. In general, when comparing two modified GR settings, it is only possible to know that $wcd(P')$ will not change if no $\pi \in \Pi_{wcd}(P')$ is removed, but if it changes, we do not know whether the *wcd* value will increase or decrease. Additionally, knowing that the $wcd(P')$ will not reduce *does not* imply that any modified GR problem derived from P' will not have a smaller *wcd*.

To apply Lemma 13, the computation of *wcd* needs to identify the policies that contribute to $wcd(P)$. Algorithm 3, used to find legal policies for suboptimal agents (SS-GRD and POSS-GRD), already identifies each policy and Procedures CommonNDPoliciesFO and CommonNDPoliciesPO evaluate policies with the same non-distinctive prefix, therefore, there is not additional work to return policies in $\Pi_{wcd}(P)$.

Additionally, in the computation of $wcd(P_0)$ we can store the evaluation of each group of policies $\Pi^{\mho^i}$ found using Eq. 4.27 in a priority queue and apply Lemmas 14 and 15 to improve the recomputation of *wcd* for each modified model.

Lemma 14. *Let P' and P^n, be two stochastic GR models s.t. $P' = \delta(m_{n-1}, \ldots, \delta(m_1, P_0))$ and $P^n = \delta(m_n, P')$, where P_0 is the original stochastic GR setting and m_n is the action to remove in P'. If $\pi \in \Pi_{\mathbf{G}}$ is a policy whose expected distinctiveness is $ED_{P'}(\pi)$, then after removing action m_n, its expected distinctiveness $ED_{P'}(\pi)$ will not increase, that is, $ED_{P^n}(\pi) \leq ED_{P'}(\pi)$.*

Proof. Let Π' be the set of policies that share the longest non-distinctive prefix of π and thus, contribute to $ED_{P'}(\pi)$ (Definition 17, p. 55). Removing action m_n is equivalent to remove

all policies Π^n that use m_n. Therefore, if $\Pi'' = \Pi' \cap \Pi^n = \emptyset$ then $ED_{P^n}(\pi) = ED_{P'}(\pi)$ else $ED_{P^n}(\pi) = ED_{P'}(\pi) \iff \exists \pi' \in \Pi' : \pi \neq \pi' \wedge \pi' \in \Pi''$, otherwise, the non-distinctive prefix of π will be shorter and hence $ED_{P^n}(\pi) < ED_{P'}(\pi)$. ■

Next, we prove Lemma 15 using Lemma 14:

Lemma 15. *Removing an action does not increase* wcd.

Proof. Let P' be a stochastic GR model s.t. $P' = modFunc(m, P_0))$, where P_0 is the original stochastic GR setting and m is the action to remove in P_0. We want to prove that $wcd(P_0) \geq wcd(P')$. Assume $\Pi^{P_0}_{wcd} = \{\pi \mid ED_{P_0}(\pi) = wcd(P_0)\}$ is the set of policies whose expected distinctiveness is equal to $wcd(P_0)$ and $\Pi^{P'}_{wcd} = \{\pi \mid ED_{P'}(\pi) = wcd(P')\}$ is the set of policies whose expected distinctiveness is equal to $wcd(P')$. Removing action m is equivalent to remove all policies Π^m that use m. Therefore, if $\Pi'' = \Pi^{P_0}_{wcd} \cap \Pi^m = \emptyset$ then $wcd(P_0) = wcd(P')$, else by Lemma 14, $\forall \pi \in \Pi'' : ED_{P'}(\pi) \leq ED_{P_0}(\pi)$. Therefore, there are two cases:

Case 1: $\exists \pi \in \Pi'' : ED_{P'}(\pi) = ED_{P_0}(\pi)$. In this case, $wcd(P_0) = wcd(P')$.

Case 2: $\forall \pi \in \Pi'' : ED_{P'}(\pi) < ED_{P_0}(\pi)$. In this case, $\forall \pi \in \Pi^{P'}_{wcd} : ED_{P'}(\pi) < wcd(P_0)$, thus, $wcd(P_0) > wcd(P')$. ■

Corollary 4 generalizes Lemma 15.

Corollary 4. *Removing a set of actions does not increase* wcd.

Proof. Let m_i represent the action to remove, P_0 the original stochastic GR model, and $\mathcal{M} = \{m_0, \ldots, m_{|\mathbf{S}|\times|\mathbf{A}|}\}$ the set of all possible modifications, that is, actions to remove. Following Definition 3 (p. 47), we apply one removal at a time, $P^{\mathcal{M}} = \delta(m_{|\mathbf{S}|\times|\mathbf{A}|}, \ldots \delta(m_0, P_0))$ and each time, Lemma 15 guarantees the ED will not increase for any partial policy $\hat{\pi}$ of P_i. ■

Procedure RECOMPUTEWCD applies Lemma 14 and Corollary 4 to avoid evaluating all policies when computing *wcd* for suboptimal settings. Line 113 in Algorithm 4 uses Procedure RECOMPUTEWCD instead of $computeMsr(P_0^{\mathbf{M}})$ and the set of policies enqueued in Line 114 are replaced by the set $\hat{\Pi}$ when $\mid \mathbf{M} - \mu \mid = 1$. $\hat{\Pi}$ contains all policies whose expected distinctiveness is equal to the new *wcd*.

The Procedure receives as arguments a priority queue Q containing groups of policies that share all their longest non-distinctive trajectories and their respective expected distinctiveness values, a set of policies to be removed Π_R, the set of legal policies $\Pi_{\mathbf{G}}$, and the modified GR problem $P^{\mathbf{M}}$. The high-level idea is to take advantage of the information in Q and reevaluate only groups of policies that change after the modification or contain only one member since their expected distinctiveness will not change otherwise. Additionally, groups of policies with lower priorities are pruned; specifically, we stop when the current *wcd* is higher than the top value in Q. First, the variables are initialized and the set of valid policies Π is updated (Lines 120-121). If some goal is unreachable using Π, then return values signaling that it is not possible to perform the proposed modification (Line 122). Otherwise, the Procedure

Procedure RECOMPUTEWCD($Q, \Pi_R, \Pi_{\mathbf{G}}, P^{\mathbf{M}}$)

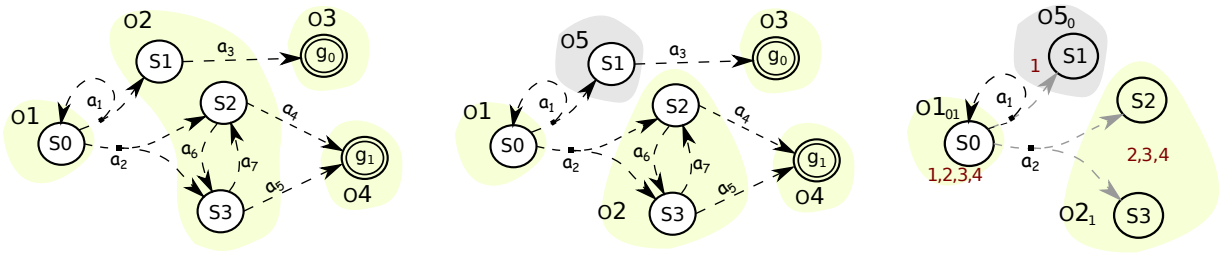
```
120 nwcd ← −1; Π̂ ← ∅; Π_dis ← ∅
121 Π ← Π_G \ Π_R
122 if | G(Π) |≠| G | then return ⟨∞, null, ∅⟩;
123 while Q ≠ ∅ do
124     ⟨cost, Π_c⟩ ← Q.dequeue
125     if Π_c ∩ Π_R ≠ ∅ ∨ | Π_c |= 1 then
126         foreach π ∈ Π_c \ Π_R do
127             ⟨Π_i, cost_i⟩ ← evaluate(π, P^M)
128             if cost_i = 0 then Π_dis ← Π_dis ∪ Π_i;
129             if cost_i > nwcd then nwcd ← cost_i; Π̂ ← Π_i ;
130             else if cost_i = nwcd then Π̂ ← Π̂ ∪ Π_i;
131     else
132         if cost > nwcd then nwcd ← cost; Π̂ ← Π_c;
133         else if cost = nwcd then Π̂ ← Π̂ ∪ Π_c;
134     if Q ≠ ∅ then
135         ⟨cost′, Π′⟩ ← Q.peek
136         if cost′ < nwcd then return ⟨nwcd, Π̂, Π_dis⟩;
137 return ⟨nwcd, Π̂, Π_dis⟩
```

analyzes the entries in Q; each entry contains the expected distinctiveness cost *cost* of all policies in Π_c and the policies themselves (Line 124). If any policy in Π_c was removed or if $|\Pi_c| = 1$ then evaluate each remaining policy and update the set of distinctive policies, new *wcd* (*nwcd*), and policies that contribute to *nwcd* when appropriate (Lines 125-130). If it is not required to reevaluate the policies in Π_c, the Procedure uses the entry in Q to update *nwcd* and $\hat{\Pi}$ (Lines 131-133). If the cost of the next groups of policies in Q is less than *nwcd*, return *nwcd*, the set of policies $\hat{\Pi}$ with expected distinctiveness equal to *nwcd*, and the set of distinctive policies Π_{dis} (Lines 134-136). In case the whole queue is processed, return $nwcd, \hat{\Pi}$ and Π_{dis} (Line 137).

For our example of Figure 5.1(a), (the same analyzed in Section 4.4), only policies 3 and 4 contribute to the *wcd* value (Figure 4.13, p.4.13) and removing actions a_1, a_2 or a_3 causes unreachable goals. Hence, when $\mu = 2$, we need to examine $\binom{7}{1} + \binom{4}{2} = 13$ possibilities, less than half of the possibilities contemplated with a naïve approach.

5.2 Sensor Refinement (SR)

Partial observability reduces an observer's ability to distinguish agent states. The design objective is thus to identify sensors whose refinement yields measure (*wcd*) reduction under budget constraints. At the extreme, refining all sensors leads to fully-observable states, yet we note that in our model, even with all sensors refined actions are still unobservable.



(a) (b) (c)

Figure 5.2: POS-GRD Design: (a) Original problem with 90% probability of success for each stochastic action, $k = 1$ (*wcd*=1.5). (b) Sensor Refinement. (c) Augmented MDP for the refined model (*wcd*=0.1). Gray arrows have a cost of 0.

We consider a sensor refinement modification that refines a *single* state to make it fully observable (Definition 6, p. 48). Hence, if an agent is in any of the *refined states*, its state is known with full certainty. Figure 5.2(a) shows the POSS-GRD problem used in

Section 4.4 (p. 113). Figure 5.2(b) depicts the problem after $S1$ is refined and mapped to a new observation $O5$, allowing the observer to distinguish it from $S2$ and $S3$. Figure 5.2(c) shows the resultant augmented MDP when $k = 1$. Each state is augmented with the set possible policies shown in red numbers and for the reader's facility, the sets of possible goals are shown as subindices of the observation IDs. Actions with cost 0, i.e. actions leading to distinctive states, are represented by gray arrows.

Note that after refinement, all distinctive states ($S1, S2$, and $S3$) are terminal, thus, actions depicted in gray do not contribute to the measure value. Also, the cost of taking an action is given per successor (as defined in Section 4.3.2, p. 109). For example, the cost of action a_1 transitioning to state $S1$ is 0 but the cost of action a_1 transitioning to state $S0$ is 1. The observer's knowledge is represented through the set of possible goals, which may change according to the observed sequence.

5.2.1 Analyzing the Effects of Sensor Refinement

Understanding how trajectories change after refinement provides insight to prune the space of modifications. In this subsection, we study the cases of trajectories affected by sensor refinement. Specifically, we use Definition 15 (p. 52) to analyze the cases where sensor refinement potentially reduces the set of possible goals and then we link it to the reduction of the expected distinctiveness (ED) of all non-distinctive policies.

Let s_r represent the state to refine, $P = \langle M, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$ and $P^m = \langle M, \mathbf{G}, k, \mathcal{N}^m, \mathcal{C}_o \rangle$ denote two stochastic GR models such that m is a sensor refinement modification and $P^m = \delta(P)$. Given a feasible trajectory $\vec{\tau} = \langle s_0, \ldots, s_n \rangle$, we can define sub trajectories $\vec{\tau}_1 = \langle s'_0, \ldots, s'_{i-1} \rangle$ and $\vec{\tau}_2 = \langle s'_0, \ldots, , s'_{i-1}, a_i, s'_i \rangle$ such that $\forall j \leq i : s_j = s'_j$ to identify each case. For ease of reading, we copy below Eq. 3.8 (p. 52), used in the computation of goals that satisfy a trajectory.

$$\mathbf{G}' = \bigcup_{\pi(s) | \exists s' : \mathcal{T}(s, \pi(s), s') > 0 \wedge \mathcal{N}(s) = \mathcal{N}(s_{i-1}) \wedge \mathcal{N}(s') = \mathcal{N}(s_i)} \mathbf{G}(\pi(s)) \tag{5.1}$$

Case 1:$s_i = s_r \wedge \forall (j < i) : \mathcal{N}(s_j) \neq \mathcal{N}(s_r)$. In this case, $\mathbf{G}(obs(\vec{\tau}_1)) = \mathbf{G}_1$ and $\mathbf{G}(obs(\vec{\tau}_2)) = \mathbf{G}_1 \cap \mathbf{G}' = \mathbf{G}_2$, where $\mathbf{G}'$ is given by Eq. 5.1. After refinement, $\mathcal{N}^m(s_i) \neq \mathcal{N}^m(s_{i-1}), \mathbf{G}(obs(\vec{\tau}_1)) = \mathbf{G}_1$, and $\mathbf{G}_3 = \mathbf{G}(obs(\vec{\tau}_2)) = \mathbf{G}_1 \cap \mathbf{G}''$, where $\mathbf{G}'' = \mathbf{G}(\pi(s_{i-1}) = a_i) = \mathbf{G}_2$. Therefore, clearly $\mathbf{G}' \supseteq \mathbf{G}''$, which implies $\mathbf{G}_2 \supseteq \mathbf{G}_3$ (Eq. 3.6 in Definition 15, p. 52).

Case 2: $s_i \neq s_r \wedge \mathcal{N}(s_i) = \mathcal{N}(s_r) \wedge \forall (j < i) : \mathcal{N}(s_j) \neq \mathcal{N}(s_r)$. In this case, $\mathbf{G}(obs(\vec{\tau}_1)) = \mathbf{G}_1$ and $\mathbf{G}(obs(\vec{\tau}_2)) = \mathbf{G}_1 \cap \mathbf{G}' = \mathbf{G}_2$, where $\mathbf{G}'$ is given by Eq. 5.1. After refinement, $\mathcal{N}^m(s_i) \neq \mathcal{N}^m(s_{i-1}), \mathbf{G}(obs(\vec{\tau}_1)) = \mathbf{G}_1$, and $\mathbf{G}_3 = \mathbf{G}(obs(\vec{\tau}_2)) = \mathbf{G}_1 \cap \mathbf{G}''$, where $\mathbf{G}''$ is given by Eq. 5.1. Even though $\mathbf{G}'$ and $\mathbf{G}''$ are computed using the same equation, the policies evaluated before refinement potentially include π' if $\exists \pi' : \mathcal{T}(s_{i-1}, \pi'(s_{i-1}), s_r) > 0$, which is not included after refinement. Therefore, $\mathbf{G}' \supseteq \mathbf{G}''$, which implies $\mathbf{G}_2 \supseteq \mathbf{G}_3$ (Eq. 3.6 in Definition 15, p. 52).

Case 3: $s_i = s_r$. In this case, $\mathbf{G}(obs(\vec{\tau}_1)) = \mathbf{G}_1$ and $\mathbf{G}(obs(\vec{\tau}_2)) = \mathbf{G}_2$; if $\mathcal{N}(s_{i-1}) = \mathcal{N}(s_i), \mathbf{G}_2 = \mathbf{G}_1$, else $\mathbf{G}_2 = \mathbf{G}_1 \cap \mathbf{G}'$, where $\mathbf{G}'$ is given by Eq. 5.1. After refinement, $\mathcal{N}^m(s_i) \neq \mathcal{N}^m(s_{i-1}), \mathbf{G}(obs(\vec{\tau}_1)) = \mathbf{G}_1^m \subseteq \mathbf{G}_1$, since $\vec{\tau}_1$ includes Case 1 or 2. $\mathbf{G}(obs(\vec{\tau}_2)) = \mathbf{G}_2^m = \mathbf{G}_1^m \cap \mathbf{G}''$, where $\mathbf{G}'' = \mathbf{G}(\pi(s_{i-1}) = a_i)$. Therefore, $\mathbf{G}' \supseteq \mathbf{G}''$, which implies $\mathbf{G}_2 \supseteq \mathbf{G}_2^m$ (Eq. 3.6 in Definition 15, p. 52).

Case 4: $s_i \neq s_r \wedge \mathcal{N}(s_i) = \mathcal{N}(s_r)$. In this case, $\mathbf{G}(obs(\vec{\tau}_1)) = \mathbf{G}_1$ and $\mathbf{G}(obs(\vec{\tau}_2)) = \mathbf{G}_2$. If $\mathcal{N}(s_{i-1}) = \mathcal{N}(s_i), \mathbf{G}_2 = \mathbf{G}_1$, else $\mathbf{G}_2 = \mathbf{G}_1 \cap \mathbf{G}'$, where $\mathbf{G}'$ is given by Eq. 5.1. After refinement, $\mathbf{G}(obs(\vec{\tau}_1)) = \mathbf{G}_1^m \subseteq \mathbf{G}_1$ since $\vec{\tau}_1$ includes Case 1 or 2, and $\mathbf{G}(obs(\vec{\tau}_2)) = \mathbf{G}_2^m = \mathbf{G}_1^m \cap \mathbf{G}''$. If $s_{i-1} = s_r, \mathbf{G}'' = \mathbf{G}(\pi(s_{i-1}) = a_i)$, else if $s_{i-1} \neq s_r \wedge \mathcal{N}(s_{i-1}) = \mathcal{N}(s_i), \mathbf{G}'' = \mathbf{G}_1^m$, else $\mathbf{G}''$ is given by Eq. 5.1. Similar to Case 2, even though $\mathbf{G}'$ and $\mathbf{G}''$ are computed using the same equation when $\mathcal{N}(s_{i-1}) \neq \mathcal{N}(s_i)$, the policies evaluated before refinement potentially include π' if $\exists \pi' : \mathcal{T}(s'_{i-1}, \pi'(s'_{i-1}), s_r) > 0$, which is not included after refinement. Therefore, $\mathbf{G}' \supseteq \mathbf{G}''$, which implies $\mathbf{G}_2 \supseteq \mathbf{G}_2^m$ (Eq. 3.6 in Definition 15, p. 52).

The cases analyzed only consider states projecting the same observation as s_r because the goals satisfying a trajectory that does not contain any of those states do not change after refinement (Lemma 16 below). Case 1 analyzes the occurrence of observing s_r before observing any other state s with $\mathcal{N}(s) = \mathcal{N}(s_r)$, and Case 2 the instance where state $s \neq s_r$ but with $\mathcal{N}(s) = \mathcal{N}(s_r)$ is observed for the first time. Cases 3 and 4 do not make assumptions about sub trajectory $\vec{\tau}_1$ except that it has a prefix that follows the assumptions of Case 1 or Case 2.

Lemma 16. *A trajectory* $\vec{\tau} = \langle s_0, \ldots, s_n \rangle$ *such that* $\forall i : 0 \leq i \leq n \implies s_i \neq s_r \wedge \mathcal{N}(s_i) \neq \mathcal{N}(s_r)$*, where* s_r *is the state to refine, does not reduce its distinctiveness cost* ($DC(\vec{\tau})$) *after refinement.*

Proof. Given a problem $P = \langle M, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$, $obs(\vec{\tau}) = o = \langle \mathcal{N}(s_0), \ldots, \mathcal{N}(s_n) \rangle$ (Definition 13, p. 51). The set of goals satisfying the observed sequence $\mathbf{G}(o)$ is found using Definition 15 (p. 52). Hence, $\forall i : \mathcal{N}(s_{i-1}) \neq \mathcal{N}(s_i) \implies \mathbf{G}(obs(\langle s_0, \pi(s_0), \ldots, s_{i-1} \rangle)) \cap \mathbf{G}'$, where $\mathbf{G}'$ is computed using Eq. 5.1. After refinement, $P^m = \delta(P) = \langle M, \mathbf{G}, k, \mathcal{N}^m, \mathcal{C}_o \rangle$, and $obs(\vec{\tau}) = o^m = \langle \mathcal{N}^m(s_0), \ldots, \mathcal{N}^m(s_n) \rangle = o$ since $\forall i : 0 \leq i \leq n \implies (s_i \neq s_r \wedge \mathcal{N}(s_i) \neq \mathcal{N}(s_r)) \implies \mathcal{N}(s_i) = \mathcal{N}^m(s_i)$. Also, $\mathbf{G}(o^m) = \mathbf{G}(o)$ because none of the values used in Eq. 5.1 changed. Therefore, $DC(\vec{\tau})$ does not change after refinement (Eq. 3.9 in Definition 17, p. 55). ■

Eq. 3.9 links the distinctiveness cost (DC) of a trajectory $\vec{\tau}$ with the number of goals that satisfy $\vec{\tau}$. By reducing the number of possible goals of an observed trajectory, sensor refinement potentially reduces $DC(\vec{\tau})$ and by doing so, the expected distinctiveness (ED) of a policy containing $\vec{\tau}$ (Definition 17, p. 55). Lemma 17 below ensures that $ED(\vec{\tau})$ does not increase after applying sensor refinement to a stochastic GR model P.

Lemma 17. *Refining state* $s \in \mathbf{S}$ *does not increase the* expected distinctiveness $ED(\hat{\pi})$ *of any partial policy* $\hat{\pi}$ *that* satisfies *a goal* $g \in \mathbf{G}$.

Proof. By Definition 6 (p. 48), refining a state does not change the set of legal policies $\Pi_{\mathbf{G}}$, nor any component of the model besides the sensor function $\mathcal{N}$; therefore, it does not change any feasible trajectory, only its projected observation. We will show that after refinement, no trajectory increases its *distinctiveness cost* (DC) and since their probabilities do not change, no policy increases its expected distinctiveness (Definition 17, p. 55).

Let $P = \langle M, \mathbf{G}, k, \mathcal{N}, \mathcal{C}_o \rangle$ and $P^m = \langle M, \mathbf{G}, k, \mathcal{N}^m, \mathcal{C}_o \rangle$ represent two stochastic GR models such that m is a sensor refinement modification and $P^m = \delta(P)$. We can model the refinement of s as a partition of $\mathbf{O} = \{s'' \mid \mathcal{N}(s) = \mathcal{N}(s'') = o\}$, where $\{s\}$ and $\mathbf{S}' = \mathbf{O} \backslash \{s\}$ are the elements of the partition. Additionally, $\forall s' \in \mathbf{S}' : \mathcal{N}^m(s') = o$ and by Definition 1 (p. 45), $\forall s'' \neq s : \mathcal{N}^m(s) \neq \mathcal{N}^m(s'')$.

Consider a feasible trajectory $\vec{\tau} = \langle s_0, \dots s_n \rangle$, where s_0 is the initial state in M and $s_n = g \in \mathbf{G}$, then $\exists i : \vec{\tau}_1 = \langle s_0, \dots, s_i \rangle \wedge \mid \mathbf{G}(o(\vec{\tau}_1)) \mid \geq 2 \wedge \vec{\tau}_2 = \langle s_{i+1}, \dots, s_n \rangle \wedge \mid \mathbf{G}(o(\vec{\tau}_2)) \mid < 2$. Note that $DC(\vec{\tau}) = DC(\vec{\tau}_1)$, to increase it after refinement, there should be a trajectory $\vec{\tau}_m = \langle s_0, \dots, s'_j, \dots, s'_m \rangle$ such that $\forall j : o(\langle s_0, \dots, s'_j \rangle) \neq o(\langle s_0, \dots, s_i \rangle)$ before refinement and $\exists k : o(\langle s_0, \dots, s'_k \rangle) = o(\langle s_0, \dots, s_{i+1} \rangle)$ after it. Since $\vec{\tau}_m$ should have existed before refinement, the observable sequence of either $\vec{\tau}$, $\vec{\tau}_m$, or both must have changed to match their observable prefixes.

We prove now that changing the observable sequences does not increase $DC(\vec{\tau})$. By Definition 13 (p. 51), $o(\vec{\tau})$ only changes if s is in $\vec{\tau}$ and to possibly match the non-distinctive prefix of $\vec{\tau}_m$, s must be in $\vec{\tau}_1$ or $s = s_{i+1}$. Since $o(s) = \mathcal{N}^m(s)$ is unique, s must be in $\vec{\tau}_m$ as well.

Hence, the only possible case is that both, $\vec{\tau}$ and $\vec{\tau}_m$ change after refinement. However, the only observable change is $\mathcal{N}(s) = o$ to $\mathcal{N}^m(s) = s$, other observable parts in both trajectories remain unchanged. Therefore, $o(\langle s_0, \ldots, s_{i+1} \rangle)$ is equal to $o(\langle s_0, \ldots, s'_j \rangle)$ after refinement, only if $\exists j : o(\langle s_0, \ldots, s'_j \rangle) = o(\langle s_0, \ldots, s_i \rangle)$ before refinement, which contradicts the original assumption. Hence, $DC(\vec{\tau})$ does not increase, and consequently, no partial policy increases its expected distinctiveness after refining s. ■

Corollary 5 generalizes Lemma 17.

Corollary 5. *Refining all states will not increase the expected distinctiveness (ED) of any partial policy $\hat{\pi}$.*

Proof. Let m_i represent the refinement of state s_i, P_0 the original stochastic GR model without refinement, and $\mathcal{M} = \{m_0, \ldots, m_{|\mathbf{S}|}\}$ the set of all possible SR modifications. Following Definition 3 (p. 47), we apply one refinement at a time: $P^{\mathcal{M}} = \delta(m_{|\mathbf{S}|}, \ldots \delta(m_0, P_0))$ and each time, Lemma 17 guarantees the ED will not increase for any partial policy $\hat{\pi}$ of P_i. ■

From the analysis at the beginning of the subsection, refining a state can reduce the ED of some policies, hence, we achieve an optimal reduction by refining all states.

5.2.2 *wcd* Reduction with Sensor Refinement

A naive approach for using sensor refinement involves finding all sets of up to μ states to refine, evaluate each refined problem with the selected measure, *wcd* in our case, and choose the set that minimizes *wcd*.

We can use the results of the previous subsection to prune the modification space and improve scalability. However, since our algorithms compute a measure using augmented MDPs, we first establish the relationship between the set of goals satisfied by an observed sequence with the set of possible goals of an augmented state.

Let $\Pi_{aug} = \langle \mathbf{S}', s'_0, \mathbf{A}', \mathcal{T}', \mathcal{C}'_o, \mathbf{G}' \rangle$ be the augmented MDP of a POS-GRD problem. We use $w(s') = s$ to denote that s is the state of the world of $s' \in \mathbf{S}'$ and $\mathbf{G}(s')$ to represent the set of possible goals of s', that is, $\forall\ 0 \leq i \leq |\mathbf{G}| : g_i \in \mathbf{G}(s') \iff (s' = s \cdot \langle pos_{g_0}, \ldots, pos_{g_{|\mathbf{G}|}} \rangle \wedge pos_{g_i} = T) \vee (s' = s \cdot \langle pos_0^{g_0}, \ldots, pos_\rho^{g_{|\mathbf{G}|}} \rangle \wedge \exists j : pos_j^{g_i} = T)$.

Lemma 18. $\forall \vec{\tau} = \langle s_0, \ldots, s_n \rangle, 0 \leq i \leq n, \exists\ 0 \leq j \leq m : s'_j \in \mathbf{S}' \wedge w(s'_j) = s_i \wedge \mathbf{G}(obs(\vec{\tau} = \langle s_0, \ldots, s_i \rangle)) = \mathbf{G}(s'_j)$.

Proof. We will prove it by induction.

Base Case: By construction of Π_{aug} given in Subsections 4.2.1 and 4.4.1 , $\mathbf{G}(s'_0) = \mathbf{G}$, which is equal to $\mathbf{G}(obs(\langle s_o \rangle))$ (Eq. 3.4 in Definition 15, p. 52).

Induction Step: Assume that $\mathbf{G}(obs(\langle s_0, \ldots, s_{k-1}\rangle)) = \mathbf{G}_{k-1} = \mathbf{G}(s'_{m-1})$ such that $w(s'_m - 1) = s_{k-1}$. We need to prove that $\mathbf{G}(obs(\langle s_0, \ldots, s_k\rangle)) = \mathbf{G}_k = \mathbf{G}(s'_m)$ when $w(s'_m) = s_k$. There are two cases:

Case 1: $\mathcal{N}(s_{k-1}) = \mathcal{N}(s_k)$. By Eq. 3.5 in Definition 15, $\mathbf{G}_k = \mathbf{G}(obs(\langle s_0, \ldots, s_k\rangle)) = \mathbf{G}_{k-1} = \mathbf{G}(obs(\langle s_0, \ldots, s_{k-1}\rangle))$ and by Eq. 4.7 and Eq. 4.8, $\mathbf{G}(s'_{m-1}) = \mathbf{G}(s'_m)$, then $\mathbf{G}(s'_m) = \mathbf{G}_k$ when $w(s'_m) = s_k$.

Case 2: $\mathcal{N}(s_{k-1}) \neq \mathcal{N}(s_k)$. By Eq. 3.6 in Definition 15, $\mathbf{G}(obs(\langle s_0, \ldots, s_k\rangle)) = \mathbf{G}_k = \mathbf{G}_{k-1} \cap \mathbf{G}'$, where $\mathbf{G}'$ is given by Eq. 5.1 with $i = k$, and $\mathbf{G}(s'_m)$ is found using Eqs. 4.7, 4.9, 4.10-4.12. Eq. 4.7 is equivalent to $\mathbf{G}_{k-1}$ and $\mathbf{G}'$ is equivalent to Eqs. 4.10-4.12 when $w(s'_m) = s_k$. Therefore, $\mathbf{G}_k = \mathbf{G}(s'_m)$. ■

Lemma 18 verifies that augmented states carry the set of possible goals given the observed trajectory used to reach them. Now we are ready to present Algorithm 5 which minimizes *wcd* more efficiently (although it could support any other measure based on ambiguous policies).

Initially, the algorithm computes the measure value msr for the original problem, retrieving as well the set of augmented states $\mathbf{S}'$, and the partition $\mathbf{O}$ of the regular state space $\mathbf{S}$ such that $\mathbf{O} = \{\mathbf{O}_1, \ldots, \mathbf{O}_n\}$ and $\forall s, s' \in \mathbf{S} : \mathcal{N}(s) = \mathcal{N}(s') \iff \exists i : s, s' \in \mathbf{O}_i$ (Line 138). We leverage the fact that the best solution is a fully-refined model. Therefore, the algorithm computes the measure for a fully-refined problem, msr_{FR}, retrieving at the same time the set of augmented states for that case (Line 139). If msr equals the value of msr_{FR}, Algorithm 5 returns the measure's value and a set with an empty set denoting that no refinement will

Algorithm 5: ReduceMeasure-SR(P_0, μ)

$\langle msr, \mathbf{S}', \mathbf{O}\rangle \leftarrow computeMsr(P_0)$
$\langle msr_{FR}, \mathbf{S}^{FR}, \bullet\rangle \leftarrow computeMsr(P^{FR})$
$\bar{\mathcal{M}} \leftarrow \emptyset; \mathbf{M}^* \leftarrow \{\emptyset\}$
if $msr = msr_{FR}$ **then** return $\langle msr, \mathbf{M}^*\rangle$;
foreach $\mathbf{O}_i \in \mathbf{O}$ **do**
    if $\exists s \in \mathbf{O}_i : w(s') = s \wedge s' \in \mathbf{S}' \cup \mathbf{S}^{FR} \wedge s' \notin \mathbf{S}' \cap \mathbf{S}^{FR}$ **then**
        $P^{\mathbf{O}_i} \leftarrow \delta(\mathbf{O}_i, P_0)$
        $\langle msr_i, \bullet, \bullet\rangle \leftarrow computeMsr(P^{\mathbf{O}_i})$
        if $msr_i = msr$ **then** $\bar{\mathcal{M}} \leftarrow \bar{\mathcal{M}} \cup \mathbf{O}_i$;
    else $\mathbf{O} \leftarrow \mathbf{O} \backslash \mathbf{O}_i$;
$Q \leftarrow enqueue(\{\emptyset\})$
while $Q \neq \emptyset$ **do**
    $\mathcal{M} \leftarrow dequeue(Q)$
    foreach $\mathbf{O}_i \in \mathbf{O}$ **do**
        foreach $s \in \mathbf{O}_i$ **do**
            if $\mathcal{M} = \{\emptyset\} \vee \mathbf{O}_i \notin \bar{\mathcal{M}} \vee \forall \hat{s} \in \mathcal{M} : \hat{s} \notin \mathbf{O}_i$ **then**
                $\mathbf{M} \leftarrow \mathcal{M} \cup \{s\}$
                if $\mid \mathbf{M} \mid < \mu$ **then** $Q \leftarrow enqueue(\mathbf{M})$;
                if $\mid \mathbf{M} \mid > 1 \vee \mathbf{O}_i \notin \bar{\mathcal{M}}$ **then**
                    $P^{\mathbf{M}} \leftarrow \delta(\mathbf{M}, P_0)$
                    $\mathbf{S}^{\mathbf{M}} \leftarrow$ AugStates$(P^{\mathbf{M}})$
                    if $\exists s \in \mathbf{O}_i : w(s') = s \wedge s' \in \mathbf{S}' \cup \mathbf{S}^{\mathbf{M}} \wedge s' \notin \mathbf{S}' \cap \mathbf{S}^{\mathbf{M}}$ **then**
                        $\langle msr', \bullet, \bullet\rangle \leftarrow computeMsr(P^{\mathbf{M}})$
                        if $msr' < msr$ **then**
                            $msr \leftarrow msr'; \mathbf{M}^* \leftarrow \mathbf{M}$
                            if $msr = msr_{FR}$ **then** return $\langle msr, \mathbf{M}\rangle$;
            else break;
return $\langle msr, \mathbf{M}^*\rangle$

further reduce msr (Lines 140-141). Next, the algorithm selects states projecting the same observation, that is, states in one set $\mathbf{O}_i$ of the partition $\mathbf{O}$, and compares all their augmented versions in both the original and the fully-refined model (Lines 142-147). If the augmented states are different, it means their set of possible goals changed and so, the measure value could change. In that case, the algorithm refines all states projecting the same observation and if the measure does not reduce, all combinations of those states are marked to be pruned (Lines 143-146). On the other hand, if the set of goals for the augmented states of interest is the same as in the fully-refined model, we can prune the corresponding set of regular states (Line 147).

Algorithm 5 uses a queue Q of modifications represented by sets of states to refine. Q is initialized with a set containing the empty set, which represents the original stochastic GR model (Line 148). Each time a set of modifications from the queue is studied, we check if adding a new state does not generate any pair of states whose combination should be pruned (Line 153). After a new combinations of states is generated, it is enqueued for future additions if the maximum budget μ allows it (Lines 154-155). If the new set of modifications contains only one state, that state should not be refined if it belongs to a set of combinations marked for pruning, since it is guaranteed not to reduce the measure's value (Line 156). Otherwise, the new modification set is applied to the original model and the algorithm computes the selected measure for the modified problem if some augmented states change with the modification (Lines 157-160). Lemma 16 guarantees that we only need to inspect augmented versions of states projecting the same observation. The measure value found is

compared against the current minimum and the smallest one is stored together with the set of refined states that generated it (Lines 161-163). Finally, the algorithm returns the minimum measure and the set of modifications used to find it (Line 165).

5.2.3 *wcd* Reduction with Sensor Refinement when Using Policy Enumeration

Algorithm 5 requires storing the set of augmented states when computing *wcd*, which is not suitable with our method to find *wcd* when using policy enumeration. Similar to the case of action removal, we can avoid re-evaluating all policies of a refined model by storing the evaluation of each group of policies $\Pi^{\mho^i}$ found using Eq. 4.27 in a priority queue and applying Lemma 17 and Corollary 5.

Algorithm 6 reduces the *wcd* value when policy enumeration is used to compute it. The algorithm receives as arguments the initial stochastic GR problem P and the budget μ of allowed modifications. Initially, it computes *wcd* of the original problem retrieving as well a priority queue Q', and a partition $\mathbf{O}$ of the regular state space $\mathbf{S}$, where $\mathbf{O} = \{\mathbf{O}_1, \ldots, \mathbf{O}_n\}$ and $\forall s, s' \in \mathbf{S} : \mathcal{N}(s) = \mathcal{N}(s') \iff \exists i : s, s' \in \mathbf{O}_i$ (Line 166). Each entry in Q' contains a set of policies that share all their largest non-distinctive trajectories and their non-distinctive cots. We leverage the fact that the best solution is a fully-refined model and that the refinement does not increase the value of *wcd*. Therefore, the algorithm recomputes *wcd* for a fully-refined problem, wcd_{FR}, retrieving at the same time the set of policies that produce

Algorithm 6: ReduceWCD-SR(P_0, μ)

```
166 ⟨wcd, Q', O⟩ ← computeWCD(P_0)
167 ⟨wcd_FR, Π_FR⟩ ← recomputeWCDPO(Q', P^FR, ∅)
168 M̄ ← ∅; M* ← {∅}; i ← 0
169 if wcd = wcd_FR then return ⟨wcd, M*⟩;
170 foreach O_i ∈ O do
171     P^{O_i} ← δ(O_i, P_0)
172     ⟨wcd_i, Π_i⟩ ← recomputeWCDPO(Q', P^{O_i}, ∅)
173     if wcd_i = wcd then M̄ ← M̄ ∪ O_i;
174     i++
175 Q ← enqueue({∅}, ∅)
176 while Q ≠ ∅ do
177     ⟨M, Π_Q⟩ ← dequeue(Q)
178     foreach O_i ∈ O do
179         foreach s ∈ O_i do
180             if M = {∅} ∨ O_i ∉ M̄ ∨ ∀ŝ ∈ M : ŝ ∉ O_i then
181                 M ← M ∪ {s}
182                 if | M |> 1 ∨ O_i ∉ M̄ then
183                     P^M ← δ(M, P_0)
184                     ⟨wcd', Π'⟩ ← recomputeWCDPO(Q', P^M, Π_Q)
185                     if | M |< μ then Q ← enqueue(M, Π');
186                     if wcd' < wcd then
187                         wcd ← wcd'; M* ← M
188                         if wcd = wcd_FR then return ⟨wcd, M⟩;
189                 else if | M |= 1 ∧ μ > 1 then Q ← enqueue(M, ∅);
190             else break;
191 return ⟨wcd, M*⟩
```

wcd_{FR}(Line 167). If *wcd* equals the value of wcd_{FR}, Algorithm 6 returns the new *wcd* value and a set with an empty set denoting that no refinement will further reduce *wcd* (Line 169). Next, the algorithm refines all states projecting the same observation, recomputes the *wcd*, and if the value does not reduce, all combinations of those states are marked to be pruned (Lines 170-174).

Algorithm 6 uses a queue Q with sets of states to refine and all policies that contribute to the *wcd* value. Q is initialized with a set containing the empty set, which represents the original stochastic GR model and with an empty set of policies (Line 175). Each time an entry from the queue is analyzed, the algorithm checks if adding a new state does not generate any pair of states whose combination should be pruned (Line 180). After a new combination of states is generated, if the new set of modifications contains only one state, that state should not be refined if it belongs to a set of combinations marked for pruning, since it is guaranteed not to reduce the measure's value (Lines 181-182). Otherwise, the new modification set is applied to the original model and the algorithm recomputes the *wcd* for the modified problem (Lines 183-184). The *wcd* value found is compared against the current minimum and the smallest one is stored together with the set of refined states that generated it (Lines 186-187). If the new *wcd* equals the value found for the fully-refined model, the algorithm returns this value and the corresponding set of refined states (Line 188). A new entry is enqueued if the budget allows it (Lines 185, 189). Finally, the algorithm returns the minimum *wcd* and the set of modifications used to find it (Line 191).

Procedure RECOMPUTEWCDPO($Q, P^{\mathbf{M}}, \Pi$)

```
192 nwcd ← −1; Π̂ ← ∅; toRemove ← ∅; flag ← T
193 while Q ≠ ∅ do
194     do
195         ⟨cost, Π_c⟩ ← Q.dequeue
196     while Π ∩ Π_c = ∅ ∧ flag = T;
197     flag ← F
198     foreach π ∈ Π_c do
199         if π ∉ toRemove then
200             ⟨cost_i, Π_i⟩ ← evaluate(π, P^M)
201             toRemove ← toRemove ∪ Π_i
202             if cost_i > nwcd then nwcd ← cost_i; Π̂ ← Π_i ;
203             else if cost_i = nwcd then Π̂ ← Π̂ ∪ Π_i;
204     if Q ≠ ∅ then
205         ⟨cost′, Π′⟩ ← Q.peek
206         if cost′ < nwcd then return ⟨nwcd, Π̂⟩;
207 return ⟨nwcd, Π̂⟩
```

Procedure RECOMPUTEWCDPO receives as arguments a max priority queue Q with sets of policies and their non-distinctive expected costs, a refined stochastic GR problem $P^{\mathbf{M}}$, and a set of policies Π. If $\mathbf{M} = \{m_0 \dots m_m\}$ and $\mathbf{M} - 1 = \{m_0 \dots m_{m-1}\}$, Π contains the set of policies that contribute to $wcd(P^{\mathbf{M}-1})$. The idea is to evaluate policies that originally had the largest expected distinctiveness (ED), recompute their distinctiveness after modification, and stop when the new ED is larger than the next value in Q since it is guaranteed that it will not increase after refinement (Lemma 17).

First, the procedure initializes the new wcd value $nwcd$, the set of policies $\bar{\Pi}$ that will contribute to that value, and other support variables (Line 192). Next, it traverses Q until a policy in Π is found (Lines 194-197). Later, the ED of each group of policies per entry is

evaluated in the refined model, keeping track of the largest value found so far (Lines 198-203). If the next value in Q is lower than $nwcd$, the procedure returns $nwcd$ and $\bar{\Pi}$ (Lines 204-206). In the worst case, the whole queue will be traversed and all policies will be evaluated, returning at the end $nwcd$ and $\bar{\Pi}$ (Line 207).

Chapter 6

Empirical Evaluation

"True genius resides in the capacity for evaluation of uncertain, hazardous, and conflicting information."

– Winston Churchill

The objective of the empirical evaluation presented in this chapter is to evaluate the usefulness of our methods and the scalability of the proposed algorithms. We describe the domains and settings used and present and analyze the experimental results.

6.1 Data and Settings

Data: We evaluate our algorithms on five domains:

- ROOM is a grid world where the actions as well as the transition probabilities are determined individually for each state and each action may have up to 6 possible successors. Instances of this domain are named using the x- and y-dimensions of the room and the number of possible goals. The initial setting for partial observability defines four contiguous states mapping to the same observation. ROOM was used in the Non-Deterministic Track of the 2006 ICAPS International Planning Competition (IPC).[5]

- GRID-NAVIGATION is a grid world where the agent has a 90% probability to succeed when moving to an adjacent cell. The initial setting for partial observability defines a mapping of four contiguous states to the same observation. Instances' names indicate the horizontal and vertical dimensions of the grid and the number of possible goals.

- ATTACK-PLANNING is a cybersecurity domain where hosts on a network have stochastically assigned vulnerabilities and a subset of hosts has files that an attacker may want to access (Vorobeychik and Pritchard, 2020). Initially, the attacker has random user credentials. Possible actions:

  (a) *Exploit*: Gives read access to the target file or, in case of failure, compromises the host. The success probability is derived from the *Common Vulnerability Scoring System*, an industry standard from NIST (Mell et al., 2007).

  (b) *Update*: Gives remote network access. Success probability: 80%.

[5] http://idm-lab.org/wiki/icaps/ipc2006/probabilistic/

(c) *Access*: Allows access to a target file if attacker has read access. Success probability: 100%.

The initial setting for partial observability hides the outcomes of a random subset of actions applicable in a set of hosts. Instances' names indicate the number of hosts, the initial setting of partial observability, and the number of possible goals.

- BLOCKSWORLD is the traditional domain with a 25% probability of slippage each time a block is picked up or put down; the goal state defines the last position of every block. We use the COLORED-BLOCKSWORLD version, a modification of this domain where each block has a color and the goal is specified in terms of colors. Thus, more than one state can represent the same goal. The initial configuration of partial observability maps partial block arrangements to different orders or settings. Instances in this domain are named using the number of blocks, number of colors, the initial configuration for partial observability, and number of goals.

- BOXWORLD is a modified LOGISTICS domain where the only action that introduces noise is "drive-truck" and there is a 20% probability that the truck ends up in one of three wrong cities. Each instance name in this domain is defined by the number of boxes, trucks, airplanes, cities and goals. Partial observability assumes a confused observer that misidentifies trucks and/or cities.

  BLOCKSWORLD and BOXWORLD were used in the Probabilistic Track of 2004 ICAPS IPC.[6]

[6] http://www.cs.rutgers.edu/~mlittman/topics/ipc04-pt/

Settings: We ran a total of 1,886 experiments in 39 instances. We used a budget μ of up to 3 modifications and allowed up to 2 suboptimal actions ($k = 1$ and $k = 2$). Experiments were conducted on a 2.10 GHz machine with 16 GB of RAM and a timeout of 52 hours. The number of reachable states varies from 16 to 16,384, with 12 to 527,866 legal policies in suboptimal settings and 2 to 9,645 policies in optimal settings. Tables present times rounded up to the next second. The source code is available at `https://github.com/cwayllace/POS-GRD`.

6.2 Computing *wcd*

Table 6.1 presents a comparison of *wcd* values and runtime (measured in seconds) across all configurations. For optimal settings, we evaluated approaches that avoid or use policy enumeration (marked $\neg PE$ and PE, respectively). There are 7 cases (marked in bold numbers) where the approach that avoids policy enumeration provides an upper bound instead of the *wcd* value. While it is possible to have a similar case in OS-GRD, we did not encounter it in our experiments. Although running times are higher when using policy enumeration, the difference is small in most cases (less than $1sec$). As expected, the *wcd* value increases with the number of relaxed assumptions. In general, suboptimal settings present higher *wcd* values.

Our approach for suboptimal settings was only able to compute *wcd* for the smallest instance in the ROOM domain, we suspect it is due to the branching factor (up to 6 successors per action), which causes huge augmented state spaces.

		OS-GRD				POS-GRD				SS-GRD $k=1$		SS-GRD $k=2$		POSS-GRD $k=1$		POSS-GRD $k=2$	
Domain	Instances	*wcd* (¬PE)	*Runtime* (*s*)	*wcd* (PE)	*Runtime* (*s*)	*wcd* (¬PE)	*Runtime* (*s*)	*wcd* (PE)	*Runtime* (*s*)	*wcd* (PE)	*Runtime* (*s*)	*wcd* (PE)	*Runtime* (*s*)	*wcd* (PE)	*Runtime* (*s*)	*wcd* (PE)	*Runtime* (*s*)
ROOM	4-4-3	3.4	1	3.4	1	4.0	1	4.0	1	5.8	1	7.8	1	7.4	1	14.5	2
	8-8-2	15.0	1	15.0	1	**16.0**	1	15.9	1	-	t-o	-	t-o	-	t-o	-	t-o
	8-8-3	6.2	1	6.2	1	**9.6**	1	9.6	1	-	t-o	-	t-o	-	t-o	-	t-o
	12-12-3	11.7	1	11.7	1	15.8	1	15.8	1	-	t-o	-	t-o	-	t-o	-	t-o
	16-16-3	1.4	1	1.4	1	9.3	1	9.3	1	-	t-o	-	t-o	-	t-o	-	t-o
	20-20-3	38.1	1	38.1	1	**45.7**	1	45.2	1	-	t-o	-	t-o	-	t-o	-	t-o
	24-24-3	12.0	1	12.0	1	**18.1**	1	18.0	1	-	t-o	-	t-o	-	t-o	-	t-o
	32-32-2	39.8	1	-	t-o	77.0	1	-	t-o	-	t-o	-	t-o	-	t-o	-	t-o
	32-32-3	86.6	1	-	t-o	86.6	1	-	t-o	-	t-o	-	t-o	-	t-o	-	t-o
	32-32-3*a*	55.8	1	-	t-o	61.2	1	-	t-o	-	t-o	-	t-o	-	t-o	-	t-o
	44-44-3	73.8	1	-	t-o	91.7	2	-	t-o	-	t-o	-	t-o	-	t-o	-	t-o
GRID-NAVIGATION	5-5-2	4.4	1	4.4	1	5.7	1	5.7	1	6.7	1	8.9	1	8.9	1	11.1	1
	4-12-3	10.0	1	10.0	1	11.2	1	11.2	1	12.2	3	14.4	83	14.4	7	17.8	371
	4-12-4*a*	5.6	1	5.6	1	7.9	1	7.9	1	7.8	1	10.0	2	10.1	1	13.3	4
	4-12-4*b*	6.7	1	6.7	1	6.8	1	6.8	1	8.9	1	11.1	2	11.1	1	13.3	4
	4-12-4*c*	6.7	1	6.7	1	7.9	1	7.9	1	8.9	1	11.1	1	10.1	1	12.3	2
	4-12-6	13.3	1	13.3	1	13.4	1	13.4	1	15.6	10	17.8	1,350	16.7	16	18.9	2,487
	6-19-3	4.4	1	4.4	1	7.9	1	7.9	1	6.7	1	8.9	4	12.2	2	14.4	21
ATTACK-PLANNING	10-2-*O*1*a*	2.5	1	2.5	1	3.7	1	3.7	1	5.2	1	6.4	1	5.2	1	7.4	1
	10-2-*O*1*b*	1.0	3	1.0	3	2.3	3	2.3	3	3.5	4	5.5	7	3.5	3	5.5	7
	10-3-*O*1*a*	7.8	1	7.8	1	8.0	1	8.0	1	9.0	1	10.3	1	10.3	1	11.3	1
	10-2-*O*2*a*	2.5	1	2.5	1	2.9	1	2.9	1	5.2	1	6.4	1	5.2	1	7.4	1
	10-2-*O*2*b*	1.0	3	1.0	3	2.3	3	2.3	3	3.5	4	5.5	8	3.5	4	5.5	7
	20-2-*O*1*a*	8.7	1	8.7	1	10.2	1	10.2	1	11.2	1	13.7	2	11.5	1	13.9	2
BLOCKS-WORLD	3-2-2-*O*1	2.8	1	2.8	1	8.7	1	8.7	1	5.0	1	5.7	1	12.0	1	12.0	1
	3-2-2-*O*2	2.8	1	2.8	1	**4.6**	1	4.4	1	5.0	1	5.7	1	7.7	1	9.0	1
	5-2-3-*O*1	4.8	1	4.8	1	**9.5**	1	8.9	1	8.1	5	12.0	371	13.3	15	23.4	2,001
	5-2-3-*O*2	4.8	1	4.8	1	**9.5**	1	8.9	1	8.1	5	12.0	355	12.1	14	16.2	1,703
	5-3-3-*O*1	7.6	1	7.6	1	8.7	1	8.7	1	10.9	2	16.8	48	12.8	2	18.1	97
	5-3-3-*O*2	7.6	1	7.6	1	8.7	1	8.7	1	10.9	2	16.8	44	14.5	2	18.1	84
	6-2-3-*O*1	14.6	1	14.6	127	19.8	1	19.8	373	21.7	879,043	-	t-o	-	t-o	-	t-o
	6-2-3-*O*2	14.6	1	14.6	127	19.8	1	19.8	550	-	t-o	-	t-o	-	t-o	-	t-o
BOXWORLD	2-1-2-4-3	0.0	1	0.0	1	3.2	1	3.2	1	3.2	1	7.1	15	5.2	1	16.5	60
	2-2-0-4-3	4.4	1	4.4	1	5.7	1	5.7	1	6.4	643	-	t-o	24.2	9,454	-	t-o
	2-2-1-4-2	3.9	1	3.9	1	**5.1**	1	4.9	1	5.9	721	-	t-o	14.4	7,470	-	t-o
	2-2-1-4-3	3.9	1	3.9	1	5.4	1	5.4	1	5.9	823	-	t-o	14.4	7,115	-	t-o
	2-2-2-4-3	4.1	1	4.1	1	5.2	1	5.2	2	25.2	5,240	-	t-o	26.3	146,409	-	t-o
	3-1-1-4-2	0.0	1	0.0	1	2.1	1	2.1	1	3.2	1	12.1	21	5.3	2	16.7	52
	3-2-2-4-3	1.0	2	1.0	2	3.3	2	3.3	2	4.2	19	-	t-o	6.4	22	-	t-o

Table 6.1: *wcd* Computation for all settings.

6.3 Design: Minimizing *wcd*

In this section, we denote by *Opt* the optimized design version that prunes the search space as explained in Chapter 5 (p. 118) and by ¬*Opt* the naïve version. Numbers in gray are approximated running times computed by multiplying the time to find the original *wcd* with the number of expected modifications according to the budget. We use those values as a reference. Their values are an approximation because the time computing *wcd* for each modified problem can change substantially, for example, in cases where goals become unreachable due to lack of legal policies or when action removal causes infinite loops.

6.3.1 Action Removal

OS-GRD

Table 6.2 presents the results when selecting action removal to modify OS-GRD settings. The *wcd* values are found constructing and solving the augmented MDP as described in Subsection 4.1.4 (p. 67); that is, the approach that avoids policy enumeration [7]. The *wcd* value reduced in 19 instances with a budget of 1 ($\mu = 1$), in 23 with $\mu = 2$, and in 21 when we used $\mu = 3$. Additionally, 16 instances present a higher *wcd* reduction with $\mu = 2$ than with $\mu = 1$, and 9 instances with $\mu = 3$ (in relation to $\mu = 2$).

[7] We chose to tabulate the ¬PE approach because it has the highest number of finished instances.

OS-GRD	$\mu = 1$	$\mu = 2$	$\mu = 3$

Domain	Instances	*wcd* Reduction	*Runtime(s)* Opt.	*Runtime(s)* ¬Opt.	*wcd* Reduction	*Runtime(s)* Opt.	*Runtime(s)* ¬Opt.	*wcd* Reduction	*Runtime(s)* Opt.	*Runtime(s)* ¬Opt.
ROOM	4-4-3	3.4 → 3.4	1	1	3.4 → 3.4	1	4	3.4 → 3.4	1	18
	8-8-2	15.0 → 15.0	1	5	15.0 → 15.0	1	262	15.0 → 15.0	1	7,479
	8-8-3	6.2 → 6.2	1	5	6.2 → 6.2	1	78	6.2 → 6.2	1	1,059
	12-12-3	11.7 → 11.7	1	8	11.7 → 11.7	15	805	11.7 → 11.7	153	35,078
	16-16-3	1.4 → 1.4	5	32	1.4 → 1.4	5	4,461	1.4 → 1.4	5	335,125
	20-20-3	38.1 → 38.1	3	122	38.1 → 38.1	3	21,597	38.1 → 38.1	3	3,982,695
	24-24-3	12.0 → 12.0	2	60	12.0 → 12.0	37	8,057	12.0 → 12.0	957	695,234
	32-32-2	39.8 → 39.8	22	299	39.8 → 39.8	1827	183,416	39.8 → 39.8	t-o	92,411,770
	32-32-3	86.6 → 86.6	22	436	86.6 → 86.6	1967	260,114	86.6 → 86.6	t-o	94,529,799
	32-32-3*a*	55.8 → 55.8	19	392	55.8 → 55.8	2781	194,539	55.8 → 55.8	t-o	71,634,007
	44-44-3	73.8 → 73.8	480	1,597	73.8 → 73.8	498	1,653,626	73.8 → 73.8	501	1,090,720,836
GRID-NA-VIGATION	5-5-2	4.4 → 0.0	1	2	4.4 → 0.0	1	21	4.4 → 0.0	1	286
	4-12-3	10.0 → 10.0	1	3	10.0 → 8.9	1	54	10.0 → 8.9	6	1,628
	4-12-4*a*	5.6 → 5.6	1	3	5.6 → 4.4	1	93	5.6 → 4.4	11	1,808
	4-12-4*b*	6.7 → 5.6	1	4	6.7 → 4.4	1	76	6.7 → 4.4	11	3,063
	4-12-4*c*	6.7 → 6.7	1	1	6.7 → 6.7	1	6	6.7 → 6.7	1	35
	4-12-6	13.3 → 12.2	1	3	13.3 → 12.2	2	101	13.3 → 12.2	22	3,219
	6-19-3	4.4 → 4.4	1	5	4.4 → 3.3	1	155	4.4 → 3.3	9	2,721
ATTACK-PLANNING	10-2-*O*1*a*	2.5 → 2.5	1	2	2.5 → 0.0	1	12	2.5 → 0.0	1	51
	10-2-*O*1*b*	1.0 → 0.0	3	23	1.0 → 0.0	3	142	1.0 → 0.0	3	507
	10-3-*O*1*a*	7.8 → 7.8	1	2	7.8 → 7.8	1	9	7.8 → 7.8	1	42
	10-2-*O*2*a*	2.5 → 2.5	1	2	2.5 → 0.0	1	13	2.5 → 0.0	1	50
	10-2-*O*2*b*	1.0 → 0.0	3	24	1.0 → 0.0	3	148	1.0 → 0.0	3	487
	20-2-*O*1*a*	8.7 → 6.2	1	9	8.7 → 6.2	1	90	8.7 → 6.2	1	600
BLOCKS-WORLD	3-2-2-*O*1	2.8 → 2.8	1	1	2.8 → 2.8	1	7	2.8 → 2.8	1	38
	3-2-2-*O*2	2.8 → 2.8	1	1	2.8 → 2.8	1	12	2.8 → 2.8	1	36
	5-2-3-*O*1	4.8 → 3.2	2	28	4.8 → 3.1	68	2,557	4.8 → 3.1	3917	171,958
	5-2-3-*O*2	4.8 → 3.2	2	25	4.8 → 3.1	69	2,964	4.8 → 3.1	3930	151,244
	5-3-3-*O*1	7.6 → 6.6	1	17	7.6 → 6.6	21	631	7.6 → 6.6	471	15,208
	5-3-3-*O*2	7.6 → 6.6	1	11	7.6 → 6.6	21	564	7.6 → 6.6	481	20,910
	6-2-3-*O*1	14.6 → 14.6	12	391	14.6 → 14.6	4791	179,478	14.6 → 14.6	t-o	50,338,605
	6-2-3-*O*2	14.6 → 14.6	12	386	14.6 → 14.6	4830	169,345	14.6 → 14.6	t-o	52,580,582
BOXWORLD	2-1-2-4-3	0.0 → 0.0	1	1	0.0 → 0.0	1	1	0.0 → 0.0	1	1
	2-2-0-4-3	4.4 → 4.4	2	17	4.4 → 3.1	75	2,374	4.4 → 3.1	4579	97,042
	2-2-1-4-2	3.9 → 3.9	1	17	3.9 → 0.0	1	1,174	3.9 → 0.0	1	46,351
	2-2-1-4-3	3.9 → 3.9	1	26	3.9 → 0.0	1	1,987	3.9 → 0.0	1	68,178
	2-2-2-4-3	4.1 → 4.1	2	34	4.1 → 0.0	34	1,987	4.1 → 0.0	39	92,975
	3-1-1-4-2	0.0 → 0.0	1	1	0.0 → 0.0	1	1	0.0 → 0.0	1	1
	3-2-2-4-3	1.0 → 1.0	2	43	1.0 → 1.0	4	697	1.0 → 1.0	16	9,876

Table 6.2: OS-GRD: Action removal avoiding policy enumeration.

Additionally, we solved OS-GRD with AR using policy enumeration (PE). Our experiments did not present differences between methods, except for the running times. As expected, computing the *wcd* for the original problem took more time (a percentage increase of 1,978% on average). However, thanks to the pruning and heuristics used, the gap reduced after design (a percentage increase of 1,265% on average for a budget $\mu = 1$, 24% for $\mu = 2$, and 50% for $\mu = 3$, not including instances that timed-out). Figure 6.1 compares both methods for all instances (represented in the horizontal axis in the same order as in Table 6.2). The vertical axes show running times in seconds on a logarithmic scale. Purple dots mark values found while avoiding policy enumeration (¬PE), and yellow dots the corresponding values for the policy enumeration approach (PE). Figure 6.1(a) shows that in almost all cases, ¬PE outperforms PE when computing the *wcd* for the original stochastic GR problem. However, when we consider design, ¬PE outperforms PE in 15 cases with a budget of 3 (Figure 6.1(d)). In general, PE will not work well in cases with a high number of policies or states. In our experiments, PE is inadequate (timed-out) for instances with those characteristics. In one case, instances 8 to 11 belong to the ROOM domain and the algorithm generates large augmented state spaces. In the other case, PE failed to solve the problem due to a large number of legal policies (instances 31 and 32 — BLOCKS-WORLD — have 9,645 policies and 7,057 reachable states each).

POS-GRD	$\mu = 1$	$\mu = 2$	$\mu = 3$

Domain	*Instances*	*wcd* Reduction	*Runtime(s)* *Opt.*	*Runtime(s)* ¬*Opt.*	*wcd* Reduction	*Runtime(s)* *Opt.*	*Runtime(s)* ¬*Opt.*	*wcd* Reduction	*Runtime(s)* *Opt.*	*Runtime(s)* ¬*Opt.*
ROOM	4-4-3	4.0 → 4.0	1	1	4.0 → 4.0	1	7	4.0 → 4.0	1	29
	8-8-2	16.0 → 16.0	1	12	16.0 → 16.0	1	446	16.0 → 16.0	1	13,567
	8-8-3	9.6 → 9.6	1	8	9.6 → 9.6	1	137	9.6 → 9.6	1	2,366
	12-12-3	15.8 → 14.4	1	18	15.8 → 14.4	15	1,461	15.8 → 14.4	143	54,225
	16-16-3	9.3 → 9.3	6	49	9.3 → 9.3	6	6,091	9.3 → 9.3	6	602,148
	20-20-3	45.7 → 45.7	4	128	45.7 → 45.7	4	33,710	45.7 → 45.7	4	4,577,620
	24-24-3	18.1 → 18.1	2	82	18.1 → 18.1	49	9,907	18.1 → 18.1	1,321	943,056
	32-32-2	77.0 → 48.0	28	515	77.0 → 47.8	2,338	254,306	77.0 → 77.0	t-o	114,835,408
	32-32-3	86.6 → 86.6	32	642	86.6 → 86.6	2,897	347,854	86.6 → 86.6	t-o	138,149,531
	32-32-3*a*	61.2 → 61.0	26	497	61.2 → 61.0	3,473	286,499	61.2 → 61.2	t-o	95,734,961
	44-44-3	91.7 → 91.7	479	2,479	91.7 → 91.7	478	2,622,521	91.7 → 91.7	457	1,787,872,170
GRID-NA-VIGATION	5-5-2	5.7 → 2.3	1	4	5.7 → 1.2	1	91	5.7 → 1.2	2	1,282
	4-12-3	11.2 → 11.2	1	10	11.2 → 10.1	1	197	11.2 → 10.1	9	4,197
	4-12-4*a*	7.9 → 6.8	1	7	7.9 → 5.7	2	225	7.9 → 4.6	15	5,783
	4-12-4*b*	6.8 → 5.7	1	7	6.8 → 4.6	2	206	6.8 → 4.6	14	5,275
	4-12-4*c*	7.9 → 7.9	1	2	7.9 → 7.9	1	14	7.9 → 7.9	1	88
	4-12-6	13.4 → 13.4	1	7	13.4 → 13.4	2	271	13.4 → 12.3	31	9,308
	6-19-3	7.9 → 6.8	1	7	7.9 → 5.7	2	251	7.9 → 5.7	15	6,494
ATTACK-PLANNING	10-2-*O*1*a*	3.7 → 3.7	1	2	3.7 → 3.7	1	19	3.7 → 3.7	1	68
	10-2-*O*1*b*	2.3 → 0.0	3	25	2.3 → 0.0	3	142	2.3 → 0.0	860,233	562
	10-3-*O*1*a*	8.0 → 8.0	1	2	8.0 → 8.0	1	11	8.0 → 8.0	1	52
	10-2-*O*2*a*	2.9 → 2.9	1	2	2.9 → 2.7	1	14	2.9 → 2.7	1	62
	10-2-*O*2*b*	2.3 → 0.0	3	25	2.3 → 0.0	3	131	2.3 → 0.0	3	497
	20-2-*O*1*a*	10.2 → 10.2	1	9	10.2 → 10.2	1	90	10.2 → 10.2	1	596
BLOCKS-WORLD	3-2-2-*O*1	8.7 → 8.7	1	2	8.7 → 6.9	1	12	8.7 → 6.9	1	53
	3-2-2-*O*2	4.6 → 3.8	1	2	4.6 → 3.8	1	20	4.6 → 3.8	1	72
	5-2-3-*O*1	9.5 → 6.9	2	35	9.5 → 4.9	73	3,308	9.5 → 3.9	3,879	227,545
	5-2-3-*O*2	9.5 → 6.0	2	32	9.5 → 3.9	72	3,349	9.5 → 3.9	3,906	201,189
	5-3-3-*O*1	8.7 → 7.4	1	18	8.7 → 7.4	21	681	8.7 → 5.6	488	26,671
	5-3-3-*O*2	8.7 → 7.4	1	15	8.7 → 7.4	22	785	8.7 → 5.6	496	23,571
	6-2-3-*O*1	19.8 → 19.8	17	500	19.8 → 18.0	6,886	195,145	19.8 → 19.8	t-o	54,047,916
	6-2-3-*O*2	19.8 → 19.8	17	488	19.8 → 19.8	7,109	172,938	19.8 → 19.8	t-o	73,153,732
BOXWORLD	2-1-2-4-3	3.2 → 3.2	1	7	3.2 → 3.2	1	119	3.2 → 3.2	1	1,457
	2-2-0-4-3	5.7 → 5.4	2	31	5.7 → 3.1	87	2,753	5.7 → 3.1	5,124	177,417
	2-2-1-4-2	5.1 → 5.1	1	35	5.1 → 1.0	6	1,938	5.1 → 1.0	155	67,366
	2-2-1-4-3	5.4 → 5.1	1	30	5.4 → 0.1	7	2,240	5.4 → 0.1	199	82,612
	2-2-2-4-3	5.2 → 4.3	3	43	5.2 → 4.2	78	2,572	5.2 → 0.9	2,313	100,138
	3-1-1-4-2	2.1 → 1.1	1	8	2.1 → 1.1	1	171	2.1 → 1.1	1	1,703
	3-2-2-4-3	3.3 → 3.3	2	35	3.3 → 3.3	4	670	3.3 → 3.3	15	7,596

Table 6.3: POS-GRD: Action removal avoiding policy enumeration.

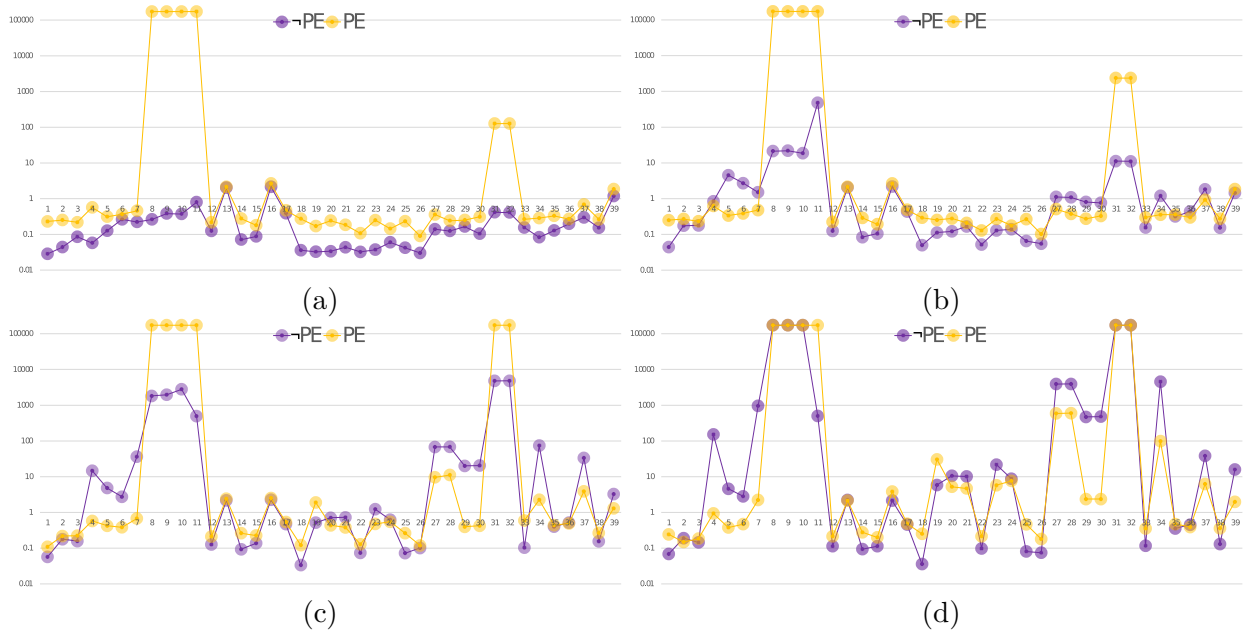


Figure 6.1: Comparison between approaches avoiding and using policy enumeration to solve OS-GRD problems. Vertical axes show the running time in seconds on a logarithmic scale for each instance in the horizontal axes. (a) *wcd* computation. (b) Design using AR and $\mu = 1$. (c) Design using AR and $\mu = 2$. (d) Design using AR and $\mu = 3$.

POS-GRD

Table 6.3 presents the information for POS-GRD settings when using action removal as a design modification. The values of *wcd* shown in the table were computed using the approach described in Section 4.4 (p. 113) that avoids policy enumeration. Similar to OS-GRD, no new instances reduced their *wcd* values when using a budget of 3 compared with 2. Nevertheless, more instances reduced the *wcd* value in this setting. We observe a *wcd* reduction in 28 instances when $\mu = 1$, in 29 instances when using $\mu = 2$, and in 25 with a budget $\mu = 3$. Further, 19 instances present lower *wcd* values with $\mu = 2$ (compared to $\mu = 1$) and 11 instances with $\mu = 3$ (in relation to $\mu = 2$).

The policy enumeration (PE) approach for AR in POS-GRD, has a lower performance than the OS-GRD case. On average, the percentage increase in running time is 5456% for the *wcd* computation without considering instances that timed-out; the gap reduces to an average percentage increase of 85% when using $\mu = 1$, 233% with $\mu = 2$, and to 489% with $\mu = 3$. Figure 6.2 visualizes the running time in seconds on a logarithmic scale (vertical axes) for every instance (horizontal axes). Purple markers present the information for ¬PE and the yellow ones for PE. Stars along the horizontal axes signal the instances where ¬PE finds an upper bound instead of the exact *wcd* value. Figure 6.2(a) shows the running-time difference when computing *wcd* and Figures 6.2(b) to (d) compares total running times for solving POS-GRD problems using AR and different budgets. Similar to OS-GRD, ¬PE outperforms PE in almost all instances when computing only *wcd* and improves in some cases when solving the complete problem. After refinement, PE outperforms ¬PE in 17 instances with a budget of 3.

SS-GRD

Tables 6.4 and 6.5 present SS-GRD problems using AR with 1 and 2 suboptimal actions and budgets ranging from 1 to 3. As expected, the original *wcd* values increase with the number of suboptimal policies. After refinement, when considering $k = 1$, *wcd* reduced in 20 instances when using budgets of $\mu = 1$ and $\mu = 2$, and in 23 instances with $\mu = 3$. However, the reduction was higher with a larger budget. More precisely, 18 instances present a higher reduction with $\mu = 2$ and 11 instances with $\mu = 3$. For SS-GRD problems with $k = 2$, *wcd*

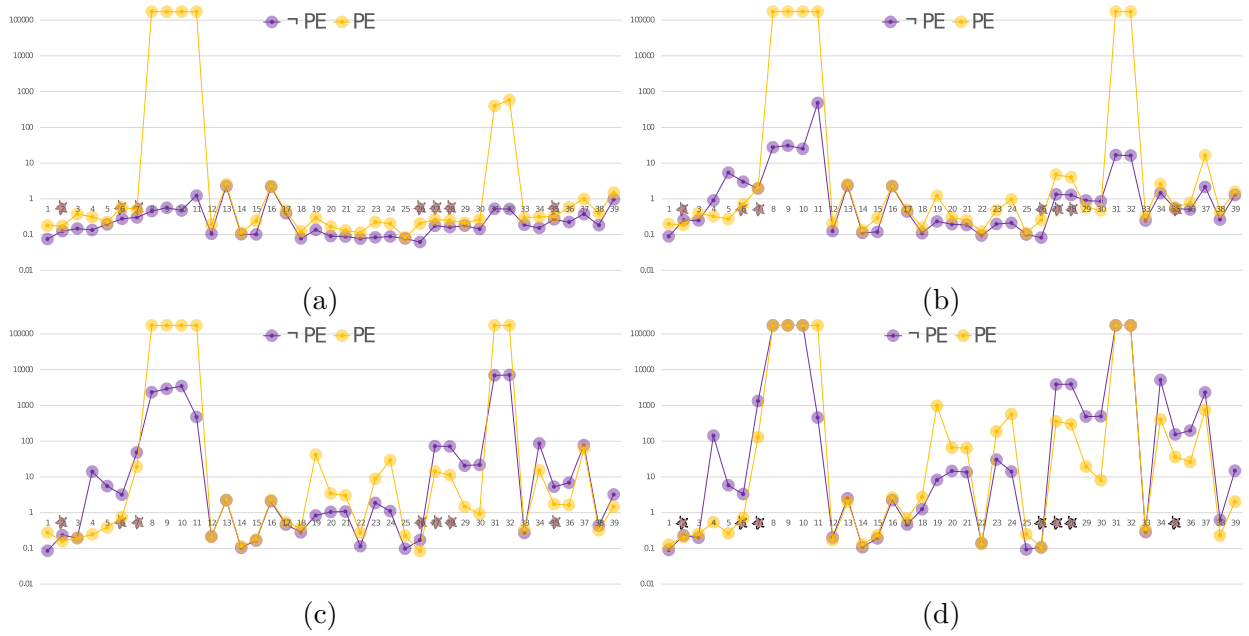


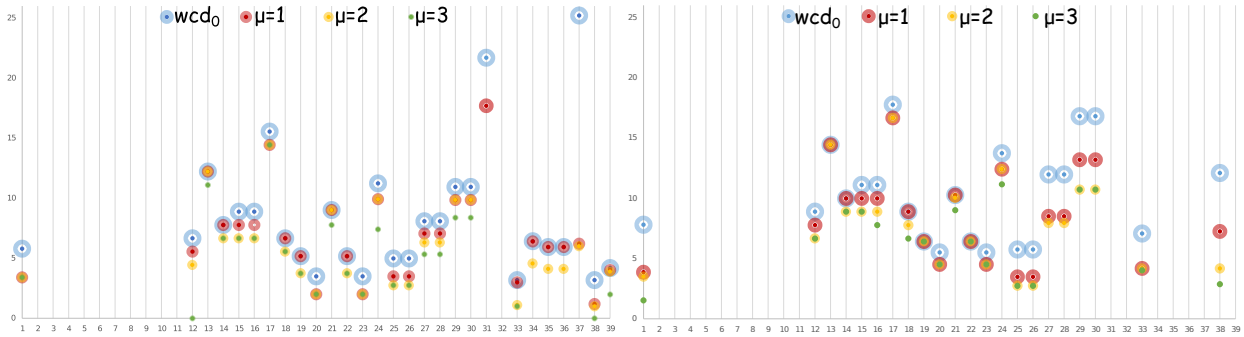
Figure 6.2: Comparison between approaches avoiding and using policy enumeration (¬PE and PE) to solve POS-GRD. Vertical axes show the running time in seconds on a logarithmic scale for each instance in the horizontal axes. Stars on the horizontal axes signal instances where both methods differ. (a) *wcd* computation. (b) Design using AR and $\mu = 1$. (c) Design using AR and $\mu = 1$. (d) Design using AR and $\mu = 1$.

SS-GRD	$\mu = 1$	$\mu = 2$	$\mu = 3$

Domain	*Instances*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*
ROOM	4-4-3	5.8 → 3.4	1	16	5.8 → 3.4	1	89	5.8 → 3.4	1	2,003
	8-8-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	8-8-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	12-12-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	16-16-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	20-20-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	24-24-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3*a*	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	44-44-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
GRID-NA-VIGATION	5-5-2	6.7 → 5.6	1	21	6.7 → 4.4	1	468	6.7 → 0.0	2	5,284
	4-12-3	12.2 → 12.2	3	313	12.2 → 12.2	254	18,776	12.2 → 11.1	13,911	751,209
	4-12-4*a*	7.8 → 7.8	1	59	7.8 → 6.7	9	3,561	7.8 → 6.7	428	87,409
	4-12-4*b*	8.9 → 7.8	1	57	8.9 → 6.7	6	3,977	8.9 → 6.7	276	96,873
	4-12-4*c*	8.9 → 7.8	1	18	8.9 → 6.7	1	454	8.9 → 6.7	8	3,653
	4-12-6	15.6 → 14.4	10	1,201	15.6 → 14.4	288	75,549	15.6 → 14.4	15,042	3,073,031
	6-19-3	6.7 → 6.7	1	102	6.7 → 5.6	25	9,060	6.7 → 5.6	2,372	431,749
ATTACK-PLANNING	10-2-*O*1*a*	5.2 → 5.2	1	41	5.2 → 3.7	1	2,500	5.2 → 3.7	1	21,312
	10-2-*O*1*b*	3.5 → 2.0	4	248	3.5 → 2.0	4	10,649	3.5 → 2.0	6	441,168
	10-3-*O*1*a*	9.0 → 9.0	1	14	9.0 → 9.0	1	137	9.0 → 7.8	1	1,461
	10-2-*O*2*a*	5.2 → 5.2	1	37	5.2 → 3.7	1	1,752	5.2 → 3.7	1	39,188
	10-2-*O*2*b*	3.5 → 2.0	4	269	3.5 → 2.0	4	11,283	3.5 → 2.0	5	347,762
	20-2-*O*1*a*	11.2 → 9.9	2	116	11.2 → 9.9	1	4,721	11.2 → 7.4	2	187,849
BLOCKS-WORLD	3-2-2-*O*1	5.0 → 3.5	1	3	5.0 → 2.8	1	26	5.0 → 2.8	1	218
	3-2-2-*O*2	5.0 → 3.5	1	3	5.0 → 2.8	1	24	5.0 → 2.8	1	170
	5-2-3-*O*1	8.1 → 7.1	6	2,066	8.1 → 6.3	80	506,399	8.1 → 5.3	8,007	80,509,437
	5-2-3-*O*2	8.1 → 7.1	6	2,074	8.1 → 6.3	78	505,743	8.1 → 5.3	7,733	80,781,193
	5-3-3-*O*1	10.9 → 9.9	2	406	10.9 → 9.9	6	63,785	10.9 → 8.4	326	5,926,924
	5-3-3-*O*2	10.9 → 9.9	2	352	10.9 → 9.9	5	54,857	10.9 → 8.4	299	6,007,500
	6-2-3-*O*1	21.7 → 17.7	902,301	37,798,820	- → -	t-o	t-o	- → -	t-o	t-o
	6-2-3-*O*2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
BOXWORLD	2-1-2-4-3	3.2 → 3.0	1	198	3.2 → 1.1	1	25,166	3.2 → 1.0	9	199
	2-2-0-4-3	6.4 → 6.4	661	674,334	6.4 → 4.6	5,563	383,481,843	6.4 → -	t-o	686,895
	2-2-1-4-2	5.9 → 5.9	779	871,423	5.9 → 4.1	8,005	468,817,240	5.9 → -	t-o	935,588
	2-2-1-4-3	5.9 → 5.9	866	990,569	5.9 → 4.1	9,146	631,888,423	- → -	t-o	t-o
	2-2-2-4-3	25.2 → 6.2	6,240	7,699,581	25.2 → 6.0	133,162	6,208,668,279	- → -	t-o	t-o
	3-1-1-4-2	3.2 → 1.2	2	228	3.2 → 1.0	2	25,012	3.2 → 0.0	8	225
	3-2-2-4-3	4.2 → 4.0	20	10,246	4.2 → 3.9	19	2,499,673	4.2 → 2.0	196	10,439

Table 6.4: SS-GRD: Action removal – 1 suboptimal action ($k = 1$)

reduced in 16 instances with a budget of $\mu = 1$ and 19 instances with $\mu = 2$ and $\mu = 3$. The reduction amount increased in 14 instances from $\mu = 1$ to $\mu = 2$, and in 7 instances from $\mu = 2$ to $\mu = 3$. Figure 6.3 visualizes the *wcd* reduction with different budgets. Markers map each instance to its corresponding *wcd* value. Blue marks represent the *wcd* value for the original stochastic GR problem, red, yellow, and green markers denote the final *wcd* values for budgets of $\mu = 1, \mu = 2$, and $\mu = 3$ respectively. Comparisons should happen among markers in the same vertical line; the lower the mark, the smaller the *wcd* value.



(a) (b)

Figure 6.3: SS-GRD: *wcd* reduction with AR using different budgets. Vertical axes show *wcd* values for each instance in the horizontal axes. Allowed budget is color-coded. (a) $k = 1$. (b) $k = 2$.

POSS-GRD

Tables 6.6 and 6.7 present experimental results for POSS-GRD problems using AR with 1 and 2 suboptimal actions, respectively, and budgets ranging from 1 to 3. Like SS-GRD, the original *wcd* value increases with k in most cases; only four instances present the same *wcd* values. POSS-GRD problems with $k = 1$ present a *wcd* reduction in 22 instances when $\mu = 1$,

SS-GRD	$\mu = 1$		$\mu = 2$		$\mu = 3$			

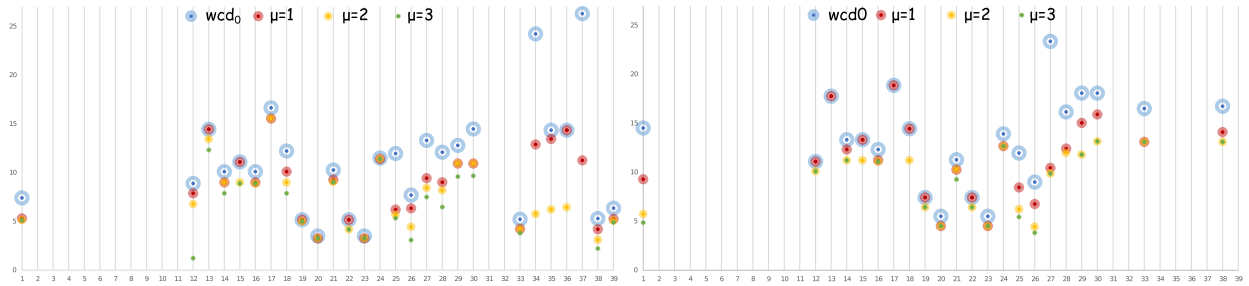
Domain	*Instances*	*wcd* Reduction	*Runtime(s)* *Opt.*	*Runtime(s)* ¬*Opt.*	*wcd* Reduction	*Runtime(s)* *Opt.*	*Runtime(s)* ¬*Opt.*	*wcd* Reduction	*Runtime(s)* *Opt.*	*Runtime(s)* ¬*Opt.*
ROOM	4-4-3	7.8 → 3.9	1	29	7.8 → 3.5	2	26	7.8 → 1.5	14	30
	8-8-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	8-8-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	12-12-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	16-16-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	20-20-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	24-24-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3*a*	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	44-44-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
GRID-NA-VIGATION	5-5-2	8.9 → 7.8	1	35	8.9 → 6.7	5	1,279	8.9 → 6.7	102	19,281
	4-12-3	14.4 → 14.4	82	10,891	14.4 → 14.4	13,496	732,655	14.4 → -	t-o	33,756,422
	4-12-4*a*	10.0 → 10.0	2	166	10.0 → 8.9	71	10,515	10.0 → 8.9	4,464	435,455
	4-12-4*b*	11.1 → 10.0	2	224	11.1 → 8.9	47	13,077	11.1 → 8.9	3,100	587,405
	4-12-4*c*	11.1 → 10.0	1	76	11.1 → 8.9	49	2,920	11.1 → 7.8	1,859	91,925
	4-12-6	17.8 → 16.7	1,343	183,968	17.8 → 16.7	66,880	12,731,161	17.8 → -	t-o	582,739,225
	6-19-3	8.9 → 8.9	4	667	8.9 → 7.8	499	70,807	8.9 → 6.7	60,752	4,883,973
ATTACK-PLANNING	10-2-*O*1*a*	6.4 → 6.4	1	137	6.4 → 6.4	1	13,460	6.4 → 6.4	7	473,174
	10-2-*O*1*b*	5.5 → 4.5	6	1,968	5.5 → 4.5	7	343,438	5.5 → 4.5	20	37,227,785
	10-3-*O*1*a*	10.3 → 10.3	1	16	10.3 → 10.0	1	282	10.3 → 9.0	1	2,009
	10-2-*O*2*a*	6.4 → 6.4	1	133	6.4 → 6.4	1	12,959	6.4 → 6.4	9	889,723
	10-2-*O*2*b*	5.5 → 4.5	7	2,027	5.5 → 4.5	7	335,314	5.5 → 4.5	27	61,678,328
	20-2-*O*1*a*	13.7 → 12.4	2	558	13.7 → 12.4	2	83,331	13.7 → 11.2	27	10,315,134
BLOCKS-WORLD	3-2-2-*O*1	5.7 → 3.5	1	3	5.7 → 2.8	1	26	5.7 → 2.8	1	172
	3-2-2-*O*2	5.7 → 3.5	1	3	5.7 → 2.8	1	27	5.7 → 2.8	1	174
	5-2-3-*O*1	12.0 → 8.5	447	342,983	12.0 → 7.9	2,798	134,866,774	12.0 → -	t-o	34,811,318,624
	5-2-3-*O*2	12.0 → 8.5	437	331,938	12.0 → 7.9	2,581	122,278,128	12.0 → -	t-o	34,759,734,775
	5-3-3-*O*1	16.8 → 13.2	48	25,806	16.8 → 10.7	113	6,867,090	16.8 → 10.7	4,228	1,589,864,429
	5-3-3-*O*2	16.8 → 13.2	44	23,875	16.8 → 10.7	112	6,736,866	16.8 → 10.7	3,780	1,365,688,406
	6-2-3-*O*1	- → -	t-o	t-o	- → -	t-o	t-o	-→ -	t-o	t-o
	6-2-3-*O*2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
BOXWORLD	2-1-2-4-3	7.1 → 4.2	15	11,440	7.1 → 4.2	30	5,049,141	7.1 → 4.0	2,709	12,371
	2-2-0-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	2-2-1-4-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	2-2-1-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	2-2-2-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	3-1-1-4-2	12.1 → 7.3	21	14,697	12.1 → 4.2	34	5,243,833	12.1 → 2.9	2,205	14,837
	3-2-2-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o

Table 6.5: SS-GRD: Action removal – 2 suboptimal actions ($k = 2$)

in 26 when $\mu = 2$, and in 22 when $\mu = 3$. The reduction amount increased in 18 cases from $\mu = 1$ to $\mu = 2$, and in 16 cases from $\mu = 2$ to $\mu = 3$.

When considering $k = 2$, (Table 6.7), we observe a *wcd* reduction in 15 instances when $\mu = 1$, in 20 when $\mu = 2$, and in 17 when $\mu = 3$. A lower number of instances with a budget of $\mu = 3$ reduced their *wcd* value because some of them did not finish on time. The reduction amount increased in 16 cases from $\mu = 1$ to $\mu = 2$, and in 5 cases from $\mu = 2$ to $\mu = 3$.

Figure 6.4 visualizes the reductions obtained in all cases. Markers map each instance to its corresponding *wcd* value. Blue markers denote the *wcd* for the original stochastic GR problem, red, yellow, and green markers denote the final *wcd* values for budgets of $\mu = 1, \mu = 2$, and $\mu = 3$ respectively. Comparisons should happen among markers in the same vertical line; the lower the mark, the smaller the *wcd* value.



(a) (b)

Figure 6.4: POSS-GRD: *wcd* reduction with AR using different budgets. Vertical axes show *wcd* values for each instance in the horizontal axes. Allowed budget is color-coded. (a) $k = 1$. (b) $k = 2$.

Figure 6.5 summarizes the tendency of running time for instances in the GRID-NAVIGATION domain across all settings and configurations when $\mu = 1$ and AR is used to modify the

POSS-GRD	$\mu = 1$			$\mu = 2$			$\mu = 3$		

Domain	*Instances*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*
ROOM	4-4-3	7.4 → 5.3	1	13	7.4 → 5.1	1	125	7.4 → 5.1	2	1,355
	8-8-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	8-8-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	12-12-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	16-16-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	20-20-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	24-24-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3*a*	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	44-44-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
GRID-NA-VIGATION	5-5-2	8.9 → 7.9	1	11	8.9 → 6.8	6	360	8.9 → 1.2	109	8,840
	4-12-3	14.4 → 14.4	9	743	14.4 → 13.4	310	47,511	14.4 → 12.3	16,049	1,896,417
	4-12-4*a*	10.1 → 9.0	1	65	10.1 → 9.0	6	3,927	10.1 → 7.9	277	167,446
	4-12-4*b*	11.1 → 11.1	3	62	11.1 → 9.0	254	4,016	11.1 → 8.9	12,409	184,684
	4-12-4*c*	10.1 → 9.0	1	10	10.1 → 8.9	1	212	10.1 → 8.9	11	3,786
	4-12-6	16.7 → 15.6	58	1,725	16.7 → 15.6	4,953	115,835	16.7 → -	t-o	4,795,860
	6-19-3	12.2 → 10.1	3	163	12.2 → 9.0	158	12,103	12.2 → 7.9	9,697	690,033
ATTACK-PLANNING	10-2-*O*1*a*	5.2 → 5.2	1	20	5.2 → 5.0	1	675	5.2 → 5.0	3	30,403
	10-2-*O*1*b*	3.5 → 3.3	4	259	3.5 → 3.3	4	10,679	3.5 → 3.3	5	280,700
	10-3-*O*1*a*	10.3 → 9.3	1	8	10.3 → 9.0	1	184	10.3 → 9.0	2	1,403
	10-2-*O*2*a*	5.2 → 5.2	1	26	5.2 → 4.2	1	666	5.2 → 4.2	3	19,263
	10-2-*O*2*b*	3.5 → 3.3	3	245	3.5 → 3.3	4	12,144	3.5 → 3.3	5	300,837
	20-2-*O*1*a*	11.5 → 11.4	1	71	11.5 → 11.4	2	4,023	11.5 → 11.4	11	171,500
BLOCKS-WORLD	3-2-2-*O*1	12.0 → 6.2	1	2	12.0 → 5.7	1	17	12.0 → 5.3	1	151
	3-2-2-*O*2	7.7 → 6.3	1	2	7.7 → 4.4	1	18	7.7 → 3.1	1	119
	5-2-3-*O*1	13.3 → 9.4	16	6,705	13.3 → 8.4	117	1,611,361	13.3 → 7.5	15,120	276,687,652
	5-2-3-*O*2	12.1 → 9.0	15	6,401	12.1 → 8.2	87	1,539,115	12.1 → 6.5	9,237	261,363,479
	5-3-3-*O*1	12.8 → 10.9	2	502	12.8 → 10.9	8	83,385	12.8 → 9.6	341	8,215,463
	5-3-3-*O*2	14.5 → 10.9	2	466	14.5 → 10.9	5	75,284	14.5 → 9.7	218	7,718,705
	6-2-3-*O*1	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	6-2-3-*O*2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
BOXWORLD	2-1-2-4-3	5.2 → 4.2	1	208	5.2 → 4.2	2	28,767	5.2 → 3.8	10	2,771,581
	2-2-0-4-3	24.2 → 12.9	9,507	10,137,789	24.2 → 5.8	15,680	5,185,174,627	24.2 → -	t-o	2,175,871,874,052
	2-2-1-4-2	14.4 → 13.5	7,151	8,526,916	14.4 → 6.2	19,730	5,416,952,567	14.4 → -	t-o	2,318,875,923,732
	2-2-1-4-3	14.4 → 14.4	7,145	8,632,698	14.4 → 6.4	19,955	4,747,455,910	- → -	t-o	t-o
	2-2-2-4-3	26.3 → 11.3	154,967	233,133,410	26.3 → -	t-o	143,290,330,007	- → -	t-o	t-o
	3-1-1-4-2	5.3 → 4.2	2	250	5.3 → 3.1	3	28,369	5.3 → 2.2	57	2,142,614
	3-2-2-4-3	6.4 → 5.3	28	14,562	6.4 → 5.1	23	2,702,301	6.4 → 4.9	812	552,766,536

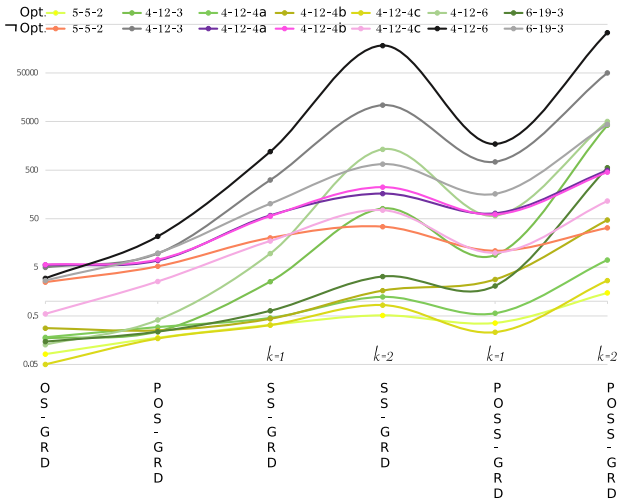
Table 6.6: POSS-GRD: Action removal – 1 suboptimal action ($k = 1$)

POSS-GRD	$\mu = 1$			$\mu = 2$			$\mu = 3$		

Domain	*Instances*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*
ROOM	4-4-3	14.5 → 9.3	2	86	14.5 → 5.7	4	2,043	14.5 → 4.9	24	30,792
	8-8-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	8-8-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	12-12-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	16-16-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	20-20-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	24-24-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3*a*	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	44-44-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
GRID-NA-VIGATION	5-5-2	11.1 → 11.1	2	33	11.1 → 10.1	37	1,147	11.1 → 10.1	681	37,815
	4-12-3	17.8 → 17.8	4,201	50,306	17.8 → -	t-o	3,399,891	17.8 → -	t-o	140,089,770
	4-12-4*a*	13.3 → 12.3	8	497	13.3 → 11.2	452	35,049	13.3 → 11.2	23,780	1,584,580
	4-12-4*b*	13.3 → 13.3	48	457	13.3 → 11.2	4,510	33,184	13.3 → -	t-o	1,383,771
	4-12-4*c*	12.3 → 11.2	3	116	12.3 → 11.1	99	5,278	12.3 → 11.1	3,621	175,531
	4-12-6	18.9 → 18.9	5,051	340,149	18.9 → -	t-o	25,066,102	18.9 → -	t-o	1,075,926,501
	6-19-3	14.4 → 14.4	562	4,372	14.4 → 11.2	78,982	449,277	14.4 → -	t-o	30,704,874
ATTACK-PLANNING	10-2-*O*1*a*	7.4 → 7.4	1	132	7.4 → 6.4	2	9,673	7.4 → 6.4	48	686,654
	10-2-*O*1*b*	5.5 → 4.5	7	2,132	5.5 → 4.5	9	355,631	5.5 → 4.5	281	37,539,919
	10-3-*O*1*a*	11.3 → 10.3	1	10	11.3 → 10.3	1	193	11.3 → 9.3	3	2,076
	10-2-*O*2*a*	7.4 → 7.4	1	121	7.4 → 6.4	2	9,414	7.4 → 6.4	41	605,462
	10-2-*O*2*b*	5.5 → 4.5	7	2,265	5.5 → 4.5	9	333,112	5.5 → 4.5	313	40,308,656
	20-2-*O*1*a*	13.9 → 12.7	3	515	13.9 → 12.7	18	83,388	13.9 → 12.7	2,254	9,815,335
BLOCKS-WORLD	3-2-2-*O*1	12.0 → 8.5	1	2	12.0 → 6.2	1	20	12.0 → 5.4	1	148
	3-2-2-*O*2	9.0 → 6.8	1	2	9.0 → 4.4	1	18	9.0 → 3.8	1	150
	5-2-3-*O*1	23.4 → 10.5	1,971	1,653,560	23.4 → 9.9	3,764	683,305,216	23.4 → 9.9	170,318	213,503,283,541
	5-2-3-*O*2	16.2 → 12.4	1,744	1,422,045	16.2 → 12.0	6,338	624,551,690	16.2 → -	t-o	164,887,063,677
	5-3-3-*O*1	18.1 → 15.1	104	57,128	18.1 → 11.8	241	15,622,121	18.1 → 11.8	18,887	3,229,062,349
	5-3-3-*O*2	18.1 → 15.9	90	49,053	18.1 → 13.2	206	14,029,759	18.1 → 13.2	15,862	2,621,568,211
	6-2-3-*O*1	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	6-2-3-*O*2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
BOXWORLD	2-1-2-4-3	16.5 → 13.1	59	48,060	16.5 → 13.1	104	18,689,391	16.5 → 13.1	3,562	5,482,474,519
	2-2-0-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	2-2-1-4-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	2-2-1-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	2-2-2-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	3-1-1-4-2	16.7 → 14.1	52	36,154	16.7 → 13.1	84	12,788,228	16.7 → 13.1	6,217	3,049,322,144
	3-2-2-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o

Table 6.7: POSS-GRD: Action removal – 2 suboptimal actions ($k = 2$)

original model. Lines only allows us to visually connect the values for the same instance. Greenish markers plot the values obtained for the optimized versions. In general, settings with a higher number of suboptimal actions ($k = 2$) require more time to find a solution since they handle the highest number of legal policies.



(a)

Figure 6.5: Running time of GRID-NAVIGATION instances across different settings when using AR and $\mu = 1$. Vertical axis: running time in seconds (logarithmic scale).

6.3.2 Sensor Refinement

POS-GRD

Table 6.8 presents the results when SR is used to modify POS-GRD settings and the *wcd* computation avoids policy enumeration [8]. The *wcd* reduced in 27 instances with $\mu = 1$, in 34 with $\mu = 2$, and in 27 with $\mu = 3$. In the last case, seven instances that had reduced their

[8]We chose to tabulate the ¬PE approach because it has the highest number of finished instances.

POS-GRD	$\mu = 1$	$\mu = 2$	$\mu = 3$

Domain	Instances	*wcd* Reduction	*Runtime(s)* Opt.	*Runtime(s)* ¬Opt.	*wcd* Reduction	*Runtime(s)* Opt.	*Runtime(s)* ¬Opt.	*wcd* Reduction	*Runtime(s)* Opt.	*Runtime(s)* ¬Opt.
ROOM	4-4-3	4.0 → 3.7	1	2	4.0 → 3.7 *	1	15	4.0 → 3.7 *	860,395	55
	8-8-2	16.0 → 16.0	1	6	16.0 → 16.0	2	201	16.0 → 16.0	27	5,696
	8-8-3	9.6 → 9.3	1	8	9.6 → 9.1	4	212	9.6 → 9.1	62	6,079
	12-12-3	15.8 → 15.8	1	23	15.8 → 15.7	6	1,677	15.8 → 15.7	189	82,237
	16-16-3	9.3 → 7.6	1	61	9.3 → 7.6	67	7,462	9.3 → 7.1	4,777	592,919
	20-20-3	45.7 → 41.1	2	121	45.7 → 40.3	333	25,369	45.7 → 39.8	43,958	3,234,404
	24-24-3	18.1 → 13.8	1	177	18.1 → 12.5	259	46,273	18.1 → 12.3	49,771	9,237,414
	32-32-2	77.0 → 53.8	15	430	77.0 → 47.2	10,377	226,099	77.0 → -	t-o	83,666,056
	32-32-3	86.6 → 86.6	8	590	86.6 → 86.6	12,577	287,432	86.6 → -	t-o	100,652,112
	32-32-3*a*	61.2 → 55.9	7	505	61.2 → 55.8	8,747	285,873	61.2 → -	t-o	88,544,663
	44-44-3	91.7 → 80.5	90	2,820	91.7 → -	t-o	2,332,353	91.7 → -	t-o	1,599,973,370
GRID-NA-VIGATION	5-5-2	5.7 → 4.6	1	3	5.7 → 4.6 *	1	33	5.7 → 4.6 *	1	210
	4-12-3	11.2 → 11.2	1	5	11.2 → 10.1	1	99	11.2 → 10.1	2	1,430
	4-12-4*a*	7.9 → 5.7	1	5	7.9 → 5.7	1	123	7.9 → 5.7 *	2	1,515
	4-12-4*b*	6.8 → 6.8 *	1	7	6.8 → 6.8 *	1	99	6.8 → 6.8 *	1	1,688
	4-12-4*c*	7.9 → 6.8 *	1	4	7.9 → 6.8 *	1	101	7.9 → 6.8 *	1	1,732
	4-12-6	13.4 → 13.4	1	5	13.4 → 13.4 *	1	119	13.4 → 13.4 *	1	2,004
	6-19-3	7.9 → 6.8	1	10	7.9 → 5.7	1	849	7.9 → 5.7	15	26,163
ATTACK-PLANNING	10-2-*O*1*a*	3.7 → 3.7	1	15	3.7 → 3.7	1	1,200	3.7 → 2.9 *	1	77,861
	10-2-*O*1*b*	2.3 → 1.3 *	3	10,654	2.3 → 1.3 *	3	29,071,372	2.3 → 1.3 *	3	39,757,899,216
	10-3-*O*1*a*	8.0 → 8.0 *	1	5	8.0 → 8.0 *	1	118	8.0 → 8.0 *	1	979
	10-2-*O*2*a*	2.9 → 2.9 *	1	18	2.9 → 2.9 *	1	1,182	2.9 → 2.9 *	1	60,299
	10-2-*O*2*b*	2.3 → 2.3	5	9,922	2.3 → 1.3 *	5	26,695,646	2.3 → 1.3 *	5	41,421,173,774
	20-2-*O*1*a*	10.2 → 10.2	1	263	10.2 → 9.0 *	1	90,685	10.2 → 9.0 *	1	21,325,009
BLOCKS-WORLD	3-2-2-*O*1	8.7 → 3.8 *	1	3	8.7 → 3.8 *	1	16	8.7 → 3.8 *	1	79
	3-2-2-*O*2	4.6 → 3.8 *	1	1	4.6 → 3.8 *	1	17	4.6 → 3.8 *	1	120
	5-2-3-*O*1	9.5 → 7.9	1	129	9.5 → 6.2	39	53,389	9.5 → 5.6 *	845	15,130,443
	5-2-3-*O*2	9.5 → 7.4	1	136	9.5 → 6.2	36	62,725	9.5 → 5.6 *	324	18,343,215
	5-3-3-*O*1	8.7 → 8.7	1	126	8.7 → 8.7 *	1	51,717	8.7 → 8.7 *	1	16,908,080
	5-3-3-*O*2	8.7 → 8.7 *	1	121	8.7 → 8.7 *	1	51,048	8.7 → 8.7 *	1	15,182,485
	6-2-3-*O*1	19.8 → 19.8	10	3,490	19.8 → 18.0	59,039	10,049,661	19.8 → -	t-o	22,508,237,285
	6-2-3-*O*2	19.8 → 19.8	3	3,969	19.8 → 19.8	94,401	11,070,930	19.8 → -	t-o	30,464,848,511
BOXWORLD	2-1-2-4-3	3.2 → 2.9	1	206	3.2 → 2.8 *	1	49,286	3.2 → 2.8 *	1	12,023,373
	2-2-0-4-3	5.7 → 5.5	1	105	5.7 → 5.4	22	35,671	5.7 → 5.4	3,156	5,863,013
	2-2-1-4-2	5.1 → 5.1	1	380	5.1 → 4.9	80	391,538	5.1 → 4.9	31,253	156,482,239
	2-2-1-4-3	5.4 → 5.1	1	308	5.4 → 5.0	78	388,251	5.4 → 4.9 *	4,309	188,123,453
	2-2-2-4-3	5.2 → 4.3	1	2,123	5.2 → 4.2	534	3,422,501	5.2 → -	t-o	3,938,454,780
	3-1-1-4-2	2.1 → 1.9 *	1	302	2.1 → 1.9 *	1	336,294	2.1 → 1.9 *	1	213,743,870
	3-2-2-4-3	3.3 → 3.1	2	18,507	3.3 → 3.1 *	24	182,214,212	3.3 → 3.1 *	23	950,514,378,021

Table 6.8: POS-GRD: Sensor refinement avoiding policy enumeration. Values with (*) represent maximal refinement

wcd value with a lower budget ran out of time. Additionally, 27 instances present lower *wcd* values with $\mu = 2$ (compared to $\mu = 1$) and 14 instances with $\mu = 3$ (in relation to $\mu = 2$), indicating that SR is more effective than AR in the analyzed instances. Since the optimized algorithm finds the *wcd* value for a fully-refined model as an intermediate step, we marked (with a *) the cases where the final result is optimal, i.e. where the reduced *wcd* is equal to the *wcd* for a fully-refined model. In total, 9 instances present optimal results when using $\mu = 1$, 17 when $\mu = 2$, and 22 when $\mu = 3$. While it is expected to have better results with higher budget, it is also good to show that we can achieve optimal results with lower budgets in several cases (more than 23% cases for$\mu = 1$ and more than 43% cases for $\mu = 2$).

Similar to the case of POS-GRD with AR, we compared the ¬PE and PE approaches. As expected, PE has a lower performance: on average, the percentage increase in running time is 5456% for the *wcd* computation of the original stochastic GR problem. However, the gap reduced to an average percentage increase of 283% for $\mu = 1$, 1929% for $\mu = 2$, and 1062% when using $\mu = 3$[9]. Figure 6.6 visualizes the running time in seconds on a logarithmic scale (vertical axes) for every instance (horizontal axes). Purple markers present the information for ¬PE and the yellow ones for PE. Figure 6.6(a), the same as Figure 6.2(a), shows the difference of the *wcd* calculation for the original stochastic GR problem and Figures 6.6(b) to (d) compare total running times when solving POS-GRD problems using SR and budgets of 1,2,and 3. With SR, the performance of PE is closer to ¬PE than with AR. Since the set of legal policies does not change and the creation of augmented MDPs in both methods has

[9]We did not consider instances that timed-out for these calculations.

similar complexity, the policy information does not provide any advantage. PE outperforms ¬PE in only 6 instances with a budget of 3.

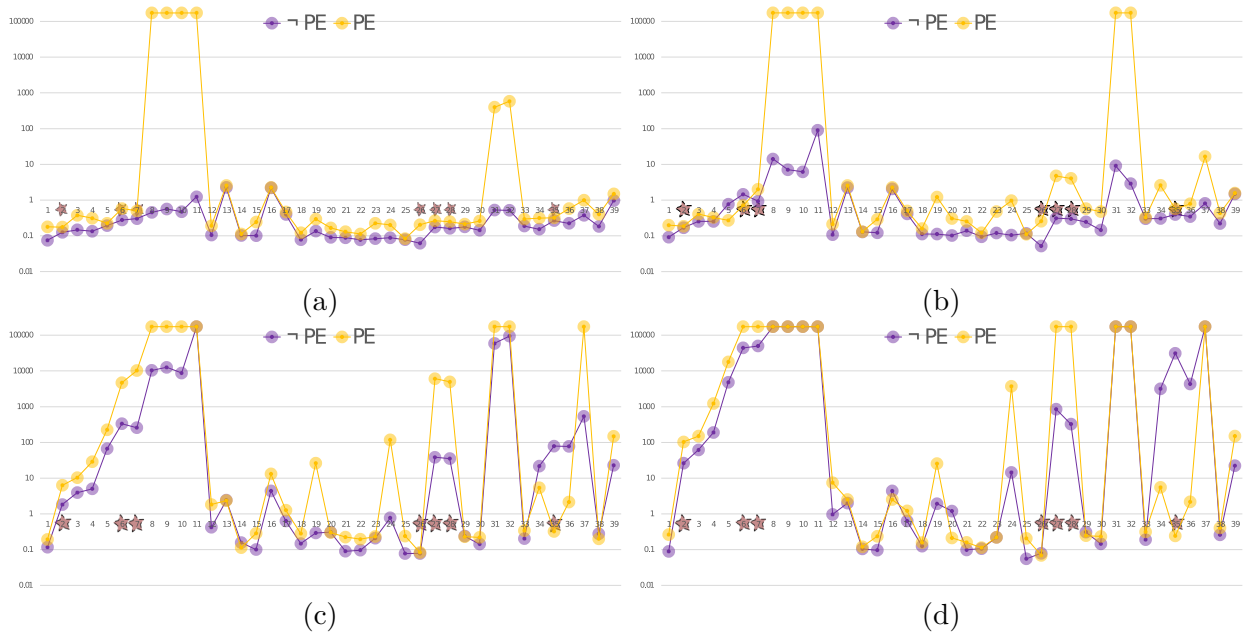


Figure 6.6: Comparison between approaches avoiding and using policy enumeration (¬PE and PE) to solve POS-GRD. Vertical axes show the running time in seconds on a logarithmic scale for each instance in the horizontal axes. (a) *wcd* computation. (b) Design using SR and $\mu = 1$. (c) Design using SR and $\mu = 2$. (d) Design using SR and $\mu = 3$.

POSS-GRD

Tables 6.9 and 6.10 present experimental results for POSS-GRD problems using SR with 1 and 2 suboptimal actions, respectively, and budgets ranging from 1 to 3. Like the POS-GRD case, we marked instances whose *wcd* value is optimal, i.e., is equal to the *wcd* of a fully-refined stochastic GR problem. The *wcd* value for the initial stochastic GR problem is higher for almost all cases with two suboptimal actions; there is only one case where the values are equal. POSS-GRD problems with $k = 1$ present a *wcd* reduction in 18 instances when $\mu = 1$

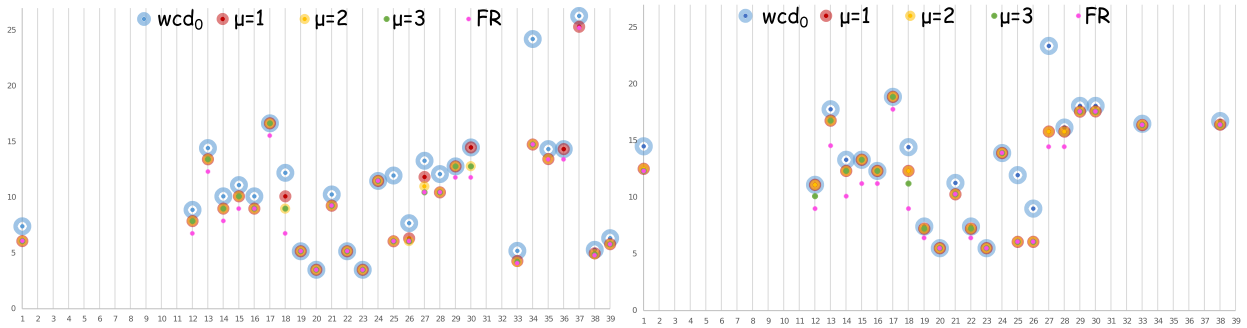
(12 of which reached optimal values), in 18 cases when $\mu = 2$ (13 with optimal values), and in 17 cases when $\mu = 3$ (14 of them with optimal values). The *wcd* reduction was higher in 6 cases when comparing $\mu = 1$ to $\mu = 2$, and only 2 instances presented higher reduction when using $\mu = 3$ instead of $\mu = 2$.

When working with $k = 2$, 11 instances reduced *wcd* when $\mu = 1$ (6 of which were optimal), 15 reduced when $\mu = 2$ (8 with optimal values), and 14 when $\mu = 3$ (10 of them with optimal values). When comparing the amount of reduction, 8 instances had higher *wcd* reduction when using $\mu = 2$ instead of $\mu = 1$ and 5 when using $\mu = 3$ rather than $\mu = 2$.

Figure 6.7 visualizes the reduction obtained in all cases. Markers map each instance to its corresponding *wcd* value. Blue markers denote the *wcd* value for the original stochastic GR problem, red, yellow, and green markers indicate the final *wcd* values for budgets of $\mu = 1$, $\mu = 2$, and $\mu = 3$, respectively, and pink markers represent *wcd* values for a fully-refined model. Comparisons should happen among markers in the same vertical line; the lower the mark, the smaller the *wcd* value, i.e., no marker should be above a blue one or below a pink one.

POSS-GRD		$\mu = 1$			$\mu = 2$			$\mu = 3$		
Domain	*Instances*	*wcd Reduction*	*Runtime(s) Opt.*	*Runtime(s) ¬Opt.*	*wcd Reduction*	*Runtime(s) Opt.*	*Runtime(s) ¬Opt.*	*wcd Reduction*	*Runtime(s) Opt.*	*Runtime(s) ¬Opt.*
ROOM	4-4-3	7.4 → 6.1 *	1	5	7.4 → 6.1 *	1	36	7.4 → 6.1 *	1	142
	8-8-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	8-8-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	12-12-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	16-16-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	20-20-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	24-24-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3*a*	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	44-44-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
GRID-NA-VIGATION	5-5-2	8.9 → 7.9	1	12	8.9 → 7.9	1	117	8.9 → 7.9	3	456
	4-12-3	14.4 → 13.4	17	325	14.4 → 13.4	165	8,194	14.4 → 13.4	2,895	105,146
	4-12-4*a*	10.1 → 9.0	1	31	10.1 → 9.0	7	798	10.1 → 9.0	106	8,644
	4-12-4*b*	11.1 → 10.1	1	34	11.1 → 10.1	7	999	11.1 → 10.1	76	14,210
	4-12-4*c*	10.1 → 9.0 *	1	15	10.1 → 9.0 *	1	417	10.1 → 9.0 *	1	6,693
	4-12-6	16.7 → 16.7	26	704	16.7 → 16.7	44	24,528	16.7 → 16.7	105	273,921
	6-19-3	12.2 → 10.1	2	128	12.2 → 9.0	24	7,060	12.2 → 9.0	1,198	225,715
ATTACK-PLANNING	10-2-*O*1*a*	5.2 → 5.2 *	1	56	5.2 → 5.2 *	1	4,753	5.2 → 5.2 *	1	229,121
	10-2-*O*1*b*	3.5 → 3.5 *	4	15,070	3.5 → 3.5 *	4	36,191,448	3.5 → 3.5 *	3	55,427,840,059
	10-3-*O*1*a*	10.3 → 9.3 *	1	15	10.3 → 9.3 *	1	324	10.3 → 9.3 *	1	3,018
	10-2-*O*2*a*	5.2 → 5.2 *	1	62	5.2 → 5.2 *	1	4,227	5.2 → 5.2 *	1	216,587
	10-2-*O*2*b*	3.5 → 3.5 *	4	17,068	3.5 → 3.5 *	4	38,918,911	3.5 → 3.5 *	4	70,572,628,839
	20-2-*O*1*a*	11.5 → 11.5 *	1	602	11.5 → 11.5 *	1	196,500	11.5 → 11.5 *	1	36,960,922
BLOCKS-WORLD	3-2-2-*O*1	12.0 → 6.1 *	1	2	12.0 → 6.1 *	1	19	12.0 → 6.1 *	1	142
	3-2-2-*O*2	7.7 → 6.3	1	2	7.7 → 6.1 *	1	19	7.7 → 6.1 *	1	120
	5-2-3-*O*1	13.3 → 11.9	21	13,015	13.3 → 11.0	2,899	5,265,292	13.3 → 10.5 *	34,077	1,496,124,609
	5-2-3-*O*2	12.1 → 10.5 *	21	12,464	12.1 → 10.5 *	21	5,495,593	12.1 → 10.5 *	19	1,399,576,283
	5-3-3-*O*1	12.8 → 12.8	3	1,507	12.8 → 12.8	350	652,974	12.8 → 12.8	105,410	281,869,607
	5-3-3-*O*2	14.5 → 14.5	3	1,348	14.5 → 12.8	406	564,650	14.5 → 12.8	107,605	181,115,401
	6-2-3-*O*1	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	6-2-3-*O*2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
BOXWORLD	2-1-2-4-3	5.2 → 4.3	2	720	5.2 → 4.2	76	246,091	5.2 → 4.2	14,962	64,591,047
	2-2-0-4-3	24.2 → 14.8 *	8,891	5,085,103	24.2 → 14.8 *	9,681	1,597,116,121	24.2 → 14.8 *	9,494	300,314,848,161
	2-2-1-4-2	14.4 → 13.5	8,042	11,721,343	14.4 → -	t-o	9,329,287,991	14.4 → -	t-o	4,475,811,246,602
	2-2-1-4-3	14.4 → 14.4	8,429	11,814,028	14.4 → -	t-o	9,354,016,555	14.4 → -	t-o	t-o
	2-2-2-4-3	26.3 → 25.3	184,398	729,879,139	26.3 → -	t-o	1,082,782,102,440	26.3 → -	t-o	t-o
	3-1-1-4-2	5.3 → 5.0	2	1,902	5.3 → 4.9	398	1,623,935	5.3 → 4.9	184,555	1,033,180,138
	3-2-2-4-3	6.4 → 5.8 *	44	340,717	6.4 → 5.8 *	44	2,805,382,290	6.4 → 5.8 *	44	15,603,487,670,685

Table 6.9: POSS-GRD: Sensor refinement – 1 suboptimal action ($k = 1$). Values with (*) denote maximal refinement



(a) (b)

Figure 6.7: POSS-GRD: *wcd* reduction with SR using different budgets. Vertical axes show *wcd* values for each instance in the horizontal axes. Allowed budget is color-coded. (a) $k = 1$. (b) $k = 2$.

POSS-GRD		$\mu = 1$			$\mu = 2$			$\mu = 3$		
Domain	*Instances*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*	*wcd* *Reduction*	*Runtime(s)* *Opt.*	*Runtime(s)* *¬Opt.*
ROOM	4-4-3	14.5 → 13.1	3	30	14.5 → 12.5	4	256	14.5 → 12.3	11	1,237
	8-8-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	8-8-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	12-12-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	16-16-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	20-20-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	24-24-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	32-32-3*a*	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	44-44-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
GRID-NA-VIGATION	5-5-2	11.1 → 11.1	1	17	11.1 → 11.1	2	236	11.1 → 10.1	2	1,268
	4-12-3	17.8 → 16.8	727	18,526	17.8 → 16.8	4,121	472,276	17.8 → 16.8	63,117	6,238,317
	4-12-4*a*	13.3 → 12.3	8	183	13.3 → 12.3	78	4,529	13.3 → 12.3	1,560	67,119
	4-12-4*b*	13.3 → 13.3	5	163	13.3 → 13.3	8	3,922	13.3 → 13.3	43	61,120
	4-12-4*c*	12.3 → 12.3	3	69	12.3 → 12.3	5	1,709	12.3 → 12.3	34	23,350
	4-12-6	18.9 → 18.9	4,525	116,497	18.9 → 18.9	4,506	2,808,360	18.9 → 18.9	5,529	45,412,506
	6-19-3	14.4 → 13.4	37	2,328	14.4 → 12.3	702	130,594	14.4 → 11.2	36,303	5,061,747
ATTACK-PLANNING	10-2-*O*1*a*	7.4 → 7.4	1	126	7.4 → 7.2	4	10,253	7.4 → 7.2	88	443,319
	10-2-*O*1*b*	5.5 → 5.5 *	7	30,322	5.5 → 5.5 *	7	76,268,868	5.5 → 5.5 *	6	114,388,104,917
	10-3-*O*1*a*	11.3 → 10.3 *	1	24	11.3 → 10.3 *	1	318	11.3 → 10.3 *	1	3,228
	10-2-*O*2*a*	7.4 → 7.4	1	121	7.4 → 7.2	4	10,869	7.4 → 7.2	140	439,300
	10-2-*O*2*b*	5.5 → 5.5 *	7	30,251	5.5 → 5.5 *	7	77,369,713	5.5 → 5.5 *	7	122,199,719,778
	20-2-*O*1*a*	13.9 → 13.9 *	2	1,246	13.9 → 13.9 *	2	428,752	13.9 → 13.9 *	2	94,381,115
BLOCKS-WORLD	3-2-2-*O*1	12.0 → 6.1 *	1	2	12.0 → 6.1 *	1	20	12.0 → 6.1 *	1	128
	3-2-2-*O*2	9.0 → 7.4	1	2	9.0 → 6.1 *	1	20	9.0 → 6.1 *	1	123
	5-2-3-*O*1	23.4 → 15.8	2,180	1,825,417	23.4 → 15.8	43,422	805,213,570	23.4 → -	t-o	200,942,004,152
	5-2-3-*O*2	16.2 → 15.8	1,988	1,566,772	16.2 → 15.8	82,050	559,047,627	16.2 → -	t-o	208,996,482,989
	5-3-3-*O*1	18.1 → 18.1	86	70,461	18.1 → 17.6 *	465	30,722,303	18.1 → 17.6 *	108	11,548,398,460
	5-3-3-*O*2	18.1 → 18.1	77	63,041	18.1 → 17.6	2,384	27,781,019	18.1 → 17.6 *	3,269	10,306,296,983
	6-2-3-*O*1	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	6-2-3-*O*2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
BOXWORLD	2-1-2-4-3	16.5 → 16.4 *	72	45,867	16.5 → 16.4 *	73	9,669,156	16.5 → 16.4 *	85	5,517,095,653
	2-2-0-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	2-2-1-4-2	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	2-2-1-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	2-2-2-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o
	3-1-1-4-2	16.7 → 16.5	71	89,171	16.7 → 16.4	18,677	7,084,873,491	16.7 → 16.4 *	80,492	45,745,136,493
	3-2-2-4-3	- → -	t-o	t-o	- → -	t-o	t-o	- → -	t-o	t-o

Table 6.10: POSS-GRD: Sensor refinement – 2 suboptimal actions ($k = 2$). Values with (*) denote maximal refinement

Chapter 7

Related Work

"If we knew what we were doing,

it would not be called research, would it?"

– Albert Einstein

(S-)GRD problems exist at the intersection of GR and design optimization. This chapter briefly surveys related work classified in three main areas: We review work that stems from GRD, alternative approaches dealing with ambiguous behavior, and several AI models that use design to solve different problems other than GR.

7.1 GRD Extensions

7.1.1 Alternative Evaluation Measures: Expected-Case Distinctiveness (*ecd*)

The *wcd* measure focuses only in the worst ambiguous case. While it is useful, different observers or situations could use other criteria for optimizing a stochastic GR problem. Wayllace et al., 2017 proposed an alternative measure that evaluates the expected case of ambiguity instead of the worst case.

An implicit assumption made by the *worst-case distinctiveness* (*wcd*) metric is that there is no prior information on the actual agent's goal. While this assumption is reasonable in many problems, it may be the case that some information is available. For example, in human-computer interaction applications, user profiles may assign different weights to each goal, where weights correspond to an agent's prior probabilities of choosing its goal. Risk-aware agents may prefer choosing slightly suboptimal trajectories, for example, to avoid optimal policies closer to risky states. Therefore not all legal policies are equally probable. Weighting goals can also symbolize the importance of goals from the observer's perspective. For instance, in a security application, some goals (i.e., targets of terrorists) are more valuable than others and should thus deserve more protection.

Further, it may often be the case where no modification sequence can reduce *wcd* (i.e., it is impossible to decrease the cost of the most expensive non-distinctive policy because there are no alternative ways to reach the goals). However, it can be possible to reduce the cost of other, less costly non-distinctive policies. Thus, intuitively, one should prefer the solution that decreases the cost of any non-distinctive policy. In such a scenario, the *wcd* value remains the same and fails to distinguish between these solutions. This situation is further exacerbated when the longest non-distinctive path is to goals with low weights!

Finally, even if a redesign using *wcd* could remove all ambiguous policies, it often requires a long sequence of modifications. Naïvely searching for the optimal sequence requires $\binom{|\mathcal{M}|}{1} + \ldots + \binom{|\mathcal{M}|}{\mu}$ computations of *wcd*, where $\mathcal{M}$ denotes the set of all modifications and μ the allowed budget. Since $\mid \mathcal{M} \mid$ is usually large and $\mid \mathcal{M} \mid > \mu$, it is commonly not possible to search for sequences of more than few modifications.

Therefore, in response to these three observations, Wayllace et al., 2017 proposed the *expected-case distinctiveness* (*ecd*), for S-GRDs that weighs the length of a trajectory to a goal (either by the probability of an agent choosing that goal or by the observer's priority) and takes the sum of all the weighted trajectory lengths. By doing so, accounts for all non-distinctive policies and potentially improves the GR ability with few modifications. Appendix A provides the formal definition and methods to compute *ecd* in optimal settings.

7.1.2 Solving GRD Using Different Approaches

Son et al., 2016 use Answer Set Programming (ASP) instead of planning to solve the problem, providing faster algorithms. However, they only consider the initial assumptions, i.e., deterministic outcomes, full observability, and optimal actors. Harman and Simoens, 2019 use action graphs to solve GRD problems in human-inhabited environments. One difficulty is that removing actions in these settings can be expensive or not feasible; thus, the authors propose a new type of modification that consists of replacing actions. For instance, moving items to another cupboard in a kitchen can help if each goal requires opening different cabinets. The authors also provided a measure that considers the average (instead of the worst) distinctiveness. This new measure is a particular case of *ecd*.

7.1.3 Plan Recognition Design

Plan recognition design (PRD) (Mirsky et al., 2017) is an extension of GRD where the objective shifts to recognize plans instead of goals and uses plan libraries instead of inverse planning. The paper also proposes the use of hierarchical plans to solve GRD (GRD-PL). The authors show that the *wcd* used in GRD-PL is a lower bound of the analogous measure for PRD. The design component removes rules from the plans; in hierarchical planning, a rule represents how a complex action can decompose to a sequence of other atomic actions.

The work in (S-)GRD and PRD use design to reduce ambiguous behavior, which is a significant problem for goal and plan recognition. The next section reviews work with ambiguity as one of its main components.

7.2 Ambiguity

One of the most critical problems of goal recognition is ambiguity; it confuses AI recognizers and people (Bonchek-Dokow and Kaminka, 2014). Unlike GRD, other techniques do not change the environment but the agent's behavior. For instance, Dragan et al., 2013 proposes methods to generate *legible* motion autonomously. Legible behavior helps an observer infer the agent's goal quickly, and it could even imply deviating from an optimal behavior purposely. Under the current assumptions, GRD does not increase the original cost to reach a goal but forces the agent to prune non-legible plans.

Agents can likewise modify their behavior with the opposite purpose, that is, to obfuscate their goal and deceive the observer (Masters and Sardina, 2017b). When we deal with adversarial goal recognition, it is logical to presume a deceptive agent. However, the observer could also use deception to its advantage. Basak et al., 2019 propose to design a cybersecurity domain to detect the type (goals, capabilities, and strategies) of an intruder or attacker. The paper uses game theory to deploy honeypots strategically, and by doing so, deceive and force the attacker to reveal its kind. They analyze three scenarios where attackers have different

goals or skills to take advantage of system vulnerabilities. This problem could use techniques very similar to GRD as an alternative approach.

Researchers also studied the ambiguity caused by partial observability (of an observer) as a tool to signal friendly observers and misguide adversarial ones *simultaneously*. Kulkarni, Srivastava, et al., 2020 use legible behavior to convey an agent's goal to a cooperative observer while hiding it from adversaries. Presumably, this requires a balance between legibility and obfuscation. The authors leverage that observers may have different degrees of observability and propose methods to optimize the necessary trade-offs.

7.3 Design

This section considers design as fixed changes to the original environment that accelerate attaining a specific objective without significantly affecting acting agents' performance.

Zhang and Parkes, 2008, Zhang, Y. Chen, et al., 2009, and Zhang, 2012 proposed a general model of computational environment design which contains most elements of (S-)GRD except for the set of goals. Environment design as defined in their work is the same as ours, where a designer performs a limited amount of modifications in an environment to induce a desirable behavior in acting agents. Zhang and Parkes, 2008 focus on modifying the reward on a limited number of states to teach the agent a policy that maximizes an observer's utility. That is, the designer aims to find incentives that cause an agent to follow a specific behavior. The authors consider both cases where the agent's reward function is known and also where it is not. The

first case produces an offline design problem. In the second case, they use online observations of the modified behavior to learn the reward function through preference elicitation and generate future modifications. Zhang, Y. Chen, et al., 2009 formally proposes a general model of environment design for one agent and one observer or designer. Similar to previous work, they analyze both static and dynamic formulation. The agent's model parameters are partially unknown initially and provide an improved algorithm with logarithmic convergence for the dynamic case. (S-)GRD falls under the umbrella of computational environment design, specifically in the static case.

Reward shaping in reinforcement learning (RL) could be regarded as environmental design since it looks for ways to modify a reward function to improve an agent's learning rate. Specifically, it alters the reward function by supplying rewards to guide an RL agent's learning process. While the intention was to accelerate the learning process, researchers found that changing the reward function might cause the agent to learn locally optimal policies, originating unwanted or unexpected behavior. Ng et al., 1999 studies the conditions under which reward shaping of an MDP does not deviate an agent from an optimal policy. The first problematic case analyzed is when the values added to the reward functions create infinite-reward cycles, in which case the agent decides to stay in the loop forever. A solution is to use potential-based shaping functions since the difference of state potentials is 0 for infinite loops. As it turns out, this is a necessary and sufficient condition to guarantee optimal behavior. Potential functions should use as much domain knowledge as possible.

Environment design is also related to mechanism design problems (Jackson, 2014). In mechanism design, modifications are equivalent to sets of game rules chosen to achieve desired outcomes (for the greater good) when agents are self-interested and possibly hide information. The collection of rules or selected mechanism affects agents' interaction. (S-)GRD may be considered a simple case of mechanism design since at least one legal policy is still valid after modifications, which implies that the agent selects its best option and a modified model may improve an observer's conditions. Considerations like hidden parameters and agent interaction lead to extensions for the S-GRD framework.

Keren, Pineda, et al., 2017 proposed the design of stochastic environments to maximize utility. In this case, both the agent and system share the same utility function and operate in a stochastic environment. In GRD, however, the agent's utility function is tied to the objective to reach its goal, while the system's utility models aim to minimize the goal recognition time.

Researchers also applied environmental design in deterministic domains to facilitate human-robot interaction. Recent approaches to improving human-robot collaboration require robots to exhibit interpretable behavior (Chakraborti et al., 2019), that is, behavior that aligns with people's expectations. Alternatively, the robot could explain its actions. However, generating this type of behavior or an explanation could be costly or impossible in certain situations. Kulkarni, Sreedharan, et al., 2020 propose to use design optimization as an offline tool to reduce the complexity of generating explicable behavior. Their approach is applicable in structured environments where robots perform repetitive tasks.

Additionally, the observer is always the same human, and the cost of modifications significantly overcomes the overhead of explicable performance. The authors further consider that after a certain number of repetitive behavior, a person will get used to it, and the behavior's degree of interpretability will increase. Therefore, the objective is to minimize both the cost incurred by the changes and explainable behavior in a fixed amount of time.

Finally, research in social laws for artificial agents (Shoham and Tennenholtz, 1992) studies the offline design of rules that modify an existing environment to improve the coordination of agents in artificial social systems. Multi-agent systems require a type of coordination so that each agent can achieve their goal without interfering with each other. Centralized control is useful when there are relatively few agents, and they all trust a central entity. Different distributed approaches require communication and coordination between agents, which can become a bottleneck in crowded environments. Social laws are mechanisms that can minimize or eliminate the need for communication between agents by restricting their behaviors. The objective of the problem is to search in the space of social laws and find a social law that, given the social multi-agent system, will induce a system where every agent is always able to move between two given focal states independently of other agents' actions. Social laws reduce the acting agents' policies, resembling action removal in (S-)GRD problems. After the design, agents should have enough freedom to achieve their goals without interfering with others'.

Nir et al., 2020 use planning to synthesize robust social laws, that is, social laws that allow all agents to reach their goals without interfering others' actions (Karpas et al., 2017). Like

the design phase of (S-)GRD problems, a sequence of small modifications is applied to the original multi-agent setting looking for a robust social law or exhausting all possible changes. The authors use action removal as modifications, and the set of removed actions describe a social law.

Chapter 8

Conclusions and Future Work

"There is no real ending.

It's just the place where you stop the story."

– Frank Herbert

Recognizing or inferring an agent's actual goal from observing its behavior is an essential skill in multi-agent environments where intelligent agents, either humans or AI, need to interact. The complexity of the task depends on the degree of ambiguous behavior observed. Goal recognition design (GRD) uses design optimization to reduce goal recognition's complexity by minimizing the ambiguity of a problem. Solving a GRD problem implies selecting a measure to evaluate the original model and finding the minimal set of modifications to the model that will optimize the adopted measure. Traditionally, GRD assesses the model quality using the

worst-case distinctiveness (*wcd*), a measure representing the largest path an acting agent can take without revealing its real goal. The seminal work on GRD (Keren, A. Gal, et al., 2014) made three assumptions: (1) Acting agents are optimal, (2) Observers and acting agents have full observability, and (3) Agent action outcomes are deterministic.

This dissertation analyzes the consequences of relaxing the initial GRD assumptions and extends the GRD framework to work in stochastic environments where the agent action outcomes are stochastic. We proposed the stochastic goal recognition design (S-GRD) framework supporting settings with different assumptions regarding the acting agent's optimality and the degree of observer's perception.

The S-GRD framework is better suited to model problems of the physical world where most interactions are inherently stochastic. For example, intelligent agents (robots) are prone to have stochastic action outcomes even if their behavior is optimal. Further, sensor limitation affects the observer. For instance, in agent navigation cases, it is more likely that an observer perceives states (the action outcomes) instead of actions. Additionally, some contiguous states may be difficult to distinguish due to sensor resolution. Finally, in most real situations, optimal behavior is not possible. However, agents, in general, try to behave rationally. Therefore, considering some degree of suboptimality can arguably produce better design solutions.

The list of factors analyzed in the previous paragraph influenced the assumptions of all supported models. Concretely, S-GRD considers four settings: (1) The Optimal S-GRD

(OS-GRD), where the acting agent is optimal, and both the observer and agent have full observability. (2) The Partially-Observable S-GRD (POS-GRD) model, where the observer cannot perceive actions and states are only partially observable, whereas the agent has full observability and is optimal. (3) The Suboptimal S-GRD (SS-GRD) problem, with suboptimal but rational agents and a fully observable model. (4) The Partially-Observable Suboptimal S-GRD (POSS-GRD), a model that combines POS-GRD and SS-GRD assumptions.

We observed that the main difference from deterministic settings is that the set of candidate goals at any given state depends on the observed trajectory used to reach it, which conditions the methods to compute *wcd* in a given scenario. The principal algorithmic tasks comprise the computation of *wcd* under all different assumptions and its reduction using two environment modifications: action removal and sensor refinement. Sensor refinement is a novel modification type applicable to partially-observable settings.

A naïve approach to compute *wcd* requires evaluating the non-distinctive (ambiguous) prefix of every legal policy and finding the maximum. We offered algorithms that avoid policy enumeration in optimal settings and reduce the number of evaluations when policy enumeration is unavoidable. Optimizing the value of *wcd* is modeled as a search problem in the space of modifications. Our algorithms prune the search space using heuristics based on the modifications' properties and taking advantage of the data structures used to compute the original *wcd*.

The empirical evaluation shows the benefits of algorithm optimization and the usefulness of S-GRD. Optimized versions outperform the naïve approach in 100% of the cases, with differences of up to six orders of magnitude. While generating legal policies with a high number of suboptimal actions is not possible, the approach of policy enumeration seems to effectively prune the search space for the design stage. If we assume a given set of " probable" policies for a determined agent, we could use that information to improve the design phase in a similar way of our PE approach with more scalable results. Overall, our S-GRD framework seems useful decreasing the complexity of stochastic GR problems by reducing the *wcd* in many cases. For the instances used in our experiments, *wcd* reduced at least in 28% of the cases and at most in 87%.

There are different directions of future work. For instance, using different measures to evaluate the GR problem, similar to the expected-case distinctiveness (*ecd*) (Subsection 7.1.1, p. 172). An alternative/complementary measure could use entropy to assess the information produced when an agent executes an action that discards some candidate goals. The redesigned model should allow higher information-gain policies, and it could serve to break ties among policies with the same non-distinctive expected cost. Defining a measure that does not consider the history but only the current state and its relation with the start state and possible goals could benefit settings with suboptimal agents, e.g., analyzing policy suffixes instead of prefixes (Masters, 2019).

Assuming that various kinds of agents could act in the environment, a designer can prioritize types and consider modifications that benefit some groups and harm others. If agents have

different reward functions, skills (available actions), or resources, some changes could have different impacts. Modifying other components of the model, such as the reward function, could lead to interesting results. For example, reducing the cost of distinctive trajectories under a budget or changing the start state's number or position.

Considering other assumptions such as dynamic environments or incomplete agents (or environment) models also pose new challenges. Since the problem's complexity increases with the number of relaxed assumptions, it would be interesting to investigate approximate solutions in real-world scenarios. One case of particular interest is assuming that one of the interested parties is human. In that case, accounting for different cost functions is critical as they reflect their preferences and biases. Estimating mutual influence is also relevant in this context, even if there is no direct interaction. Active goal recognition, where the observer actively modifies the environment (and updates the human model), is more appropriate when considering a human acting agent.

References

Amir, Ofra and Ya'akov Gal (2013). "Plan Recognition and Visualization in Exploratory Learning Environments". In: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 3.3, pp. 1–23.

Ang, Samuel, Hau Chan, Albert Xin Jiang, and William Yeoh (2017). "Game-Theoretic Goal Recognition Models with Applications to Security Domains". In: *Proceedings of the International Conference on Decision and Game Theory for Security (GameSec)*, pp. 256–272.

Archer, L Bruce (1968). "The Structure of the Design Process". PhD thesis. Royal College of Art.

Baker, Chris L, Rebecca Saxe, and Joshua B Tenenbaum (2009). "Action Understanding as Inverse Planning". In: *Cognition* 113.3, pp. 329–349.

Baker, Chris L and Joshua B Tenenbaum (2014). "Modeling Human Plan Recognition Using Bayesian Theory of Mind". In: *Plan, Activity, and Intent Recognition: Theory and Practice*. Elsevier Amsterdam, pp. 177–204.

Baker, Chris L, Joshua B Tenenbaum, and Rebecca R Saxe (2007). "Goal Inference as Inverse Planning". In: *Proceedings of the Annual Meeting of the Cognitive Science Society (CogSci)*, pp. 779–784.

Basak, Anjon, Charles Kamhoua, Sridhar Venkatesan, Marcus Gutierrez, Ahmed H Anwar, and Christopher Kiekintveld (2019). "Identifying Stealthy Attackers in a Game Theoretic Framework Using Deception". In: *Proceedings of the International Conference on Decision and Game Theory for Security (GameSec)*, pp. 21–32.

Bellman, Richard (1957). *Dynamic Programming*. Princeton University Press.

Blaylock, Nate and James Allen (2003). "Corpus-Based, Statistical Goal Recognition". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1303–1308.

Bonchek-Dokow, Elisheva and Gal A Kaminka (2014). "Towards Computational Models of Intention Detection and Intention Prediction". In: *Cognitive Systems Research* 28, pp. 44–79.

Bonet, Blai (2007). "On the Speed of Convergence of Value Iteration on Stochastic Shortest-Path Problems". In: *Mathematics of Operations Research* 32.2, pp. 365–373.

Chakraborti, Tathagata, Anagha Kulkarni, Sarath Sreedharan, David E Smith, and Subbarao Kambhampati (2019). "Explicability? Legibility? Predictability? Transparency? Privacy?

Security? the Emerging Landscape of Interpretable Agent Behavior". In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 86–96.

Charniak, Eugene and Robert Prescott Goldman (1991). *Probabilistic Abduction for Plan Recognition*. Brown University, Department of Computer Science.

Chen, Guanlin, Hui Yao, and Zebing Wang (2010). "An Intelligent WLAN Intrusion Prevention System Based on Signature Detection and Plan Recognition". In: *Proceedings of the International Conference on Future Networks (ICFN)*, pp. 168–172.

Cohen, Philip R, C Raymond Perrault, and James F Allen (1981). "Beyond Question Answering". In: *Strategies for Natural Language Processing*. Lawrence Erlbaum Associates, pp. 245–274.

Dai, Peng and Judy Goldsmith (2009). "Finding Best k Policies". In: *Proceedings of the International Conference on Algorithmic Decision Theory (ADT)*, pp. 144–155.

Dai, Peng and Judy Goldsmith (2010). "Ranking Policies in Discrete Markov Decision Processes". In: *Annals of Mathematics and Artificial Intelligence* 59.1, pp. 107–123.

Dai, Peng, Daniel S Weld, Judy Goldsmith, et al. (2011). "Topological Value Iteration Algorithms". In: *Journal of Artificial Intelligence Research (JAIR)* 42, pp. 181–209.

Dragan, Anca D, Kenton CT Lee, and Siddhartha S Srinivasa (2013). "Legibility and Predictability of Robot Motion". In: *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 301–308.

Fikes, Richard and Nils J. Nilsson (1971). "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence(AIJ)* 2.3/4, pp. 189–208.

Geffner, Hector and Blai Bonet (2013). *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Geib, Christopher W (2009). "Delaying Commitment in Plan Recognition Using Combinatory Categorial Grammars". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1702–1707.

Geib, Christopher W and Robert P Goldman (2001). "Plan Recognition in Intrusion Detection Systems". In: *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX II)*, pp. 46–55.

Geib, Christopher W and Pavan Kantharaju (2018). "Learning Combinatory Categorial Grammars for Plan Recognition". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3007–3014.

Geib, Christopher W and Mark Steedman (2007). "On Natural Language Processing and Plan Recognition". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1612–1617.

Gero, John (2012). *Design Optimization*. Elsevier.

Ghallab, Malik et al. (1998). *PDDL| The Planning Domain Definition Language*. Tech. rep. CVC TR-98-003 / DCS TR-1165. Yale Center for Computational Vision and Control.

Ha, Eun Young, Jonathan P Rowe, Bradford W Mott, and James C Lester (2011). "Goal Recognition with Markov Logic Networks for Player-Adaptive Games". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pp. 2113–2119.

Harman, Helen and Pieter Simoens (2019). "Action Graphs for Performing Goal Recognition Design on Human-Inhabited Environments". In: *Sensors* 19.12, p. 2741.

Hobbs, Jerry R, Mark Stickel, Paul Martin, and Douglas Edwards (1988). "Interpretation as Abduction". In: *Proceedings of the Annual Meeting on Association for Computational Linguistics (ACL)*, pp. 95–103.

Jackson, Matthew O (2014). *Mechanism Theory*. Stanford University - Department of Economics; Santa Fe Institute.

Jarvis, Peter, Teresa Lunt, and Karen Myers (2005). "Identifying Terrorist Activity with AI Plan Recognition Technology". In: *AI Magazine* 26.3, pp. 73–81.

Johnson, W. Lewis (2010). "Serious Use of a Serious Game for Language Learning". In: *International Journal of Artificial Intelligence in Education* 20.2, pp. 175–195.

Kabanza, Froduald, Philipe Bellefeuille, Francis Bisson, Abder Rezak Benaskeur, and Hengameh Irandoust (2010). "Opponent Behaviour Recognition for Real-Time Strategy Games". In: *Proceedings of the Workshop on Plan, Activity and Intent Recognition (PAIR)*, pp. 29–36.

Karpas, Erez, Alexander Shleyfman, and Moshe Tennenholtz (2017). "Automated Verification of Social Law Robustness in STRIPS". In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 163–171.

Kautz, Henry A and James F Allen (1986). "Generalized Plan Recognition." In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, p. 5.

Kelley, Richard, Liesl Wigand, Brian Hamilton, Katie Browne, Monica Nicolescu, and Mircea Nicolescu (2012). "Deep Networks for Predicting Human Intent with Respect to Objects". In: *Proceedings of the International Conference on Human-Robot Interaction (HRI)*, pp. 171–172.

Keren, Sarah, Avigdor Gal, and Erez Karpas (2014). "Goal Recognition Design". In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 154–162.

Keren, Sarah, Avigdor Gal, and Erez Karpas (2015). "Goal Recognition Design for Non-Optimal Agents". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3298–3304.

Keren, Sarah, Avigdor Gal, and Erez Karpas (2016a). "Goal Recognition Design With Non-Observable Actions". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3152–3158.

Keren, Sarah, Avigdor Gal, and Erez Karpas (2016b). "Privacy Preserving Plans in Partially Observable Environments". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3170–3176.

Keren, Sarah, Avigdor Gal, and Erez Karpas (2018). "Strong Stubborn Sets for Efficient Goal Recognition Design". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 141–149.

Keren, Sarah, Avigdor Gal, and Erez Karpas (2019). "Goal recognition Design in Deterministic Environments". In: *Journal of Artificial Intelligence Research (JAIR)* 65, pp. 209–269.

Keren, Sarah, Luis Pineda, Avigdor Gal, Erez Karpas, and Shlomo Zilberstein (2017). "Equi-Reward Utility Maximizing Design in Stochastic Environments". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4353–4360.

Keren, Sarah, Haifeng Xu, Kofi Kwapong, and David Parkes (2020). "Information Shaping for Enhanced Goal Recognition of Partially-Informed Agents". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 9908–9915.

Kolobov, Andrey, Daniel S Weld, et al. (2009). "ReTrASE: Integrating Paradigms for Approximate Probabilistic Planning". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1746–1753.

Kolobov, Andrey, Daniel Weld, et al. (2010). "SixthSense: Fast and Reliable Recognition of Dead Ends in MDPs". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1108–1114.

Kulkarni, Anagha, Sarath Sreedharan, Sarah Keren, Tathagata Chakraborti, David Smith, and Subbarao Kambhampati (2020). "Designing Environments Conducive to Interpretable Robot Behavior". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10982–10989.

Kulkarni, Anagha, Siddharth Srivastava, and Subbarao Kambhampati (2020). "Signaling Friends and Head-Faking Enemies Simultaneously: Balancing Goal Obfuscation and Goal Legibility". In: *Proceedings of the Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 1889–1891.

Kushmerick, Nicholas, Steve Hanks, and Daniel S Weld (1995). "An Algorithm for Probabilistic Planning". In: *Artificial Intelligence(AIJ)* 76.1-2, pp. 239–286.

Lago Pereira, Silvio do, Leliane Nunes De Barros, and Fábio Gagliardi Cozman (2008). "Strong Probabilistic Planning". In: *Proceedings of the Mexican International Conference on Artificial Intelligence (MICAI)*, pp. 636–652.

Lee, Seung, Bradford Mott, and James Lester (2012). "Real-Time Narrative-Centered Tutorial Planning for Story-Based Learning". In: *Proceedings of the International Conference on Intelligent Tutoring Systems (ITS)*, pp. 476–481.

Levine, Steven James and Brian Charles Williams (2018). "Watching and acting together: concurrent plan recognition and adaptation for human-robot teams". In: *Journal of Artificial Intelligence Research (JAIR)* 63, pp. 281–359.

Littman, Michael L, Judy Goldsmith, and Martin Mundhenk (1998). "The computational complexity of probabilistic planning". In: *Journal of Artificial Intelligence Research (JAIR)* 9, pp. 1–36.

Majercik, Stephen M and Michael L Littman (1998). "MAXPLAN: A New Approach to Probabilistic Planning." In: *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 86–93.

Maraist, John (2017). "String Shuffling Over a Gap Between Parsing and Plan Recognition". In: *Proceedings of the Workshop on Plan, Activity and Intent Recognition (PAIR)*.

Massardi, Jean, Mathieu Gravel, and Eric Beaudry (2019). "Error-Tolerant Anytime Approach to Plan Recognition Using a Particle Filter". In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 284–291.

Masters, Peta (2019). "Goal Recognition and Deception in Path-Planning". PhD thesis. RMIT University.

Masters, Peta and Sebastian Sardina (2017a). "Cost-Based Goal Recognition for Path-Planning". In: *Proceedings of the Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 750–758.

Masters, Peta and Sebastian Sardina (2017b). "Deceptive Path-Planning." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4368–4375.

Mausam and Andrey Kolobov (2012). *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

McQuiggan, Scott, Jonathan Rowe, Sunyoung Lee, and James Lester (2008). "Story-Based Learning: The Impact of Narrative on Learning Experiences and Outcomes". In: *Proceedings of the International Conference on Intelligent Tutoring Systems (ITS)*, pp. 530–539.

Mell, Peter, Karen Scarfone, and Sasha Romanosky (June 2007). *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. 1st ed. NIST and Carnegie Mellon University. URL: `http://www.first.org/cvss/cvss-guide.html`.

Min, Wookhee, Eunyoung Ha, Jonathan Rowe, Bradford Mott, and James Lester (2014). "Deep Learning-Based Goal Recognition in Open-Ended Digital Games". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pp. 37–43.

Min, Wookhee, Bradford W Mott, Jonathan P Rowe, Barry Liu, and James C Lester (2016). "Player Goal Recognition in Open-World Digital Games with Long Short-Term Memory Networks." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2590–2596.

Mirsky, Reuth, Roni Stern, Ya'akov Gal, and Meir Kalech (2017). "Plan Recognition Design". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 4971–4972.

Ng, Andrew Y, Daishi Harada, and Stuart Russell (1999). "Policy invariance under reward transformations: Theory and application to reward shaping". In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 278–287.

Nir, Ronen, Alexander Shleyfman, and Erez Karpas (2020). "Automated Synthesis of Social Laws in STRIPS". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 9941–9948.

Oh, Jean, Felipe Meneguzzi, Katia Sycara, and Timothy Norman (2010). "ANTIPA: An Agent Architecture for Intelligent Information Assistance". In: *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 1055–1056.

Oh, Jean, Felipe Meneguzzi, Katia Sycara, and Timothy Norman (2011a). "An Agent Architecture for Prognostic Reasoning Assistance". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2513–2518.

Oh, Jean, Felipe Meneguzzi, Katia Sycara, and Timothy Norman (2011b). "Probabilistic Plan Recognition for Intelligent Information Agents: Towards Proactive Software Assistant Agents". In: *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART)*, pp. 281–287.

OxfordUniversityPress (2019). *Designer: Definition of Designer by Lexico*. URL: https://www.lexico.com/en/definition/designer.

Papalambros, Panos Y and Douglass J Wilde (2000). *Principles of Optimal Design: Modeling and Computation*. Cambridge University Press.

Pattison, David and Derek Long (2010). "Domain Independent Goal Recognition". In: *Proceedings of the Starting AI Researchers Symposium (STAIRS)*, p. 238.

Pattison, David and Derek Long (2011). "Accurately Determining Intermediate and Terminal Plan States Using Bayesian Goal Recognition". In: *Proceedings of the Workshop on Goal, Activity and Plan Recognition (GAPRec)*, p. 32.

Pereira, Ramon Fraga, Nir Oren, and Felipe Meneguzzi (2017). "Landmark-Based Heuristics for Goal Recognition". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3622–3628.

Qin, Xinzhou and Wenke Lee (2004). "Attack plan recognition and prediction using causal networks". In: *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, pp. 370–379.

Rahnev, Dobromir and Rachel N Denison (2018). "Suboptimality in Perceptual Decision Making". In: *Behavioral and Brain Sciences (BBS)* 41, e223.

Ramírez, Miquel and Hector Geffner (2009). "Plan Recognition as Planning". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1778–1783.

Ramírez, Miquel and Hector Geffner (2010). "Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1121–1126.

Ramírez, Miquel and Hector Geffner (2011). "Goal Recognition over POMDPs: Inferring the Intention of a POMDP Agent". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2009–2014.

Riabov, Anton Viktorovich, Shirin Sohrabi Araghi, and Octavian Udrea (Feb. 2020). *Plan recognition with unreliable observations*. US Patent App. 16/670,098.

Rosenfeld, Ariel and Sarit Kraus (2018). "Predicting Human Decision-Making: From Prediction to Action". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 12.1, pp. 1–150.

Rosenman, Michael A and John S Gero (1985). "A system for Integrated Optimal Design". In: *Design Optimization*, pp. 259–294.

Schmidt, Charles F., NS Sridharan, and John L. Goodson (1978). "The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence". In: *Artificial Intelligence(AIJ)* 11.1, pp. 45–83.

Shoham, Yoav and Moshe Tennenholtz (1992). "On the synthesis of useful social laws for artificial agent societies (preliminary report)". In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 276–281.

Shvo, Maayan (2019). "Towards Empathetic Planning and Plan Recognition". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 525–526.

Singla, Parag and Raymond J Mooney (2011). "Abductive Markov Logic for Plan Recognition". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1069–1075.

Sohrabi, Shirin, Anton V Riabov, and Octavian Udrea (2016). "Plan Recognition as Planning Revisited." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3258–3264.

Son, Tran Cao, Orkunt Sabuncu, Christian Schulz-Hanke, Torsten Schaub, and William Yeoh (2016). "Solving Goal Recognition Design using ASP". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Sukthankar, Gita, Christopher Geib, Hung Hai Bui, David Pynadath, and Robert P Goldman (2014). *Plan, activity, and intent recognition: Theory and practice*. Newnes.

Tarjan, Robert (1972). "Depth-First Search and Linear Graph Algorithms". In: *SIAM Journal on Computing (SICOMP)* 1.2, pp. 146–160.

Tauber, Sean and Mark Steyvers (2011). "Using Inverse Planning and Theory of Mind for Social Goal Inference". In: *Proceedings of the Annual Meeting of the Cognitive Science Society (CogSci)*, pp. 2480–2485.

Tavakkoli, Alireza, Richard Kelley, Christopher King, Mircea Nicolescu, Monica Nicolescu, and George Bebis (2007). "A Vision-Based Architecture for Intent Recognition". In: *Proceedings of the International Symposium on Advances in Visual Computing (ISVC)*, pp. 173–182.

Uzan, Oriel, Reuth Dekel, Or Seri, and Ya'akov Gal (2015). "Plan Recognition for Exploratory Learning Environments Using Interleaved Temporal Search". In: *AI Magazine* 36.2, pp. 10–21.

Valmari, Antti (1989). "Stubborn sets for reduced state space generation". In: *Proceedings of the International Conference on Application and Theory of Petri Nets (ICATPN)*, pp. 491–515.

Vered, Mor, Ramon Fraga Pereira, Maurício Cecílio Magnaguagno, Gal A Kaminka, and Felipe Meneguzzi (2018). "Towards Online Goal Recognition Combining Goal Mirroring and Landmarks." In: *Proceedings of the Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 2112–2114.

Vilain, Marc (1990). "Getting Serious About Parsing Plans: A Grammatical Analysis of Plan Recognition". In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 190–197.

Vorobeychik, Yevgeniy and Michael Pritchard (2020). "Plan interdiction games". In: *Adaptive Autonomous Secure Cyber Systems*. Springer, pp. 159–182.

Wayllace, Christabel, Ping Hou, and William Yeoh (2017). "New Metrics and Algorithms for Stochastic Goal Recognition Design Problems". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4455–4462.

Yoon, Sung Wook, Alan Fern, and Robert Givan (2007). "FF-Replan: A Baseline for Probabilistic Planning." In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 352–359.

Yoon, Sung Wook, Alan Fern, Robert Givan, and Subbarao Kambhampati (2008). "Probabilistic Planning via Determinization in Hindsight." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1010–1016.

Younes, Håkan LS and Michael L Littman (2004). *PPDDL1. 0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects*. Tech. rep. CMU-CS-04-167. Carnegie Mellon University.

Zhang, Haoqi (2012). "Computational Environment Design". PhD thesis. Harvard University.

Zhang, Haoqi, Yiling Chen, and David C Parkes (2009). "A General Approach to Environment Design with One Agent". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2002–2008.

Zhang, Haoqi and David C Parkes (2008). "Value-Based Policy Teaching with Active Indirect Elicitation." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 208–214.

Appendix A

Alternative Evaluation Measure for S-GRD

S-GRD problems use design optimization to improve goal recognition. Like any other optimization problem, S-GRD requires an evaluation criterion to select the best design. In Chapter 4, we (re)defined the worst-case distinctiveness (*wcd*) measure, first introduced by Keren, A. Gal, et al., 2014, to assess stochastic GR problems. While *wcd* is an ingenious metric, focusing on an ambiguous policy with the highest expected cost may disregard other variables important for an observer. In this appendix, we propose methods to compute the expected-case distinctiveness (*ecd*), a measure that could replace or complement *wcd*.

A.1 Expected-Case Distinctiveness (*ecd*)

A.1.1 Model

Definition 21. *Given that the* expected distinctiveness $ED(\hat{\pi})$ *of a partial policy* $\hat{\pi}$ *is the expected distinctiveness cost of its trajectories,* $\sum_{\vec{\tau}} P_{\hat{\pi}}(\vec{\tau})DC(\vec{\tau})$ *(Definition 17, p. 55).*

The expected case distinctiveness *of a stochastic GR problem* P *is:*

$$\begin{aligned} \mathrm{ecd}(P) &= \sum_{\hat{\pi}\in\Pi_{\mathbf{G}}} P_r(\hat{\pi})ED(\hat{\pi}) \\ where \quad P_r(\hat{\pi}) &= \frac{1}{Z}\sum_{g\in\mathbf{G}(\hat{\pi})} P_r(g)P_r(ED_{\hat{\pi}}) \\ Z &= \sum_{\hat{\pi}\in\Pi_{\mathbf{G}}}\sum_{g\in\mathbf{G}(\hat{\pi})} P_r(g)P_r(ED_{\hat{\pi}}) \end{aligned} \tag{A.1}$$

$P_r(g)$ *is the weight of goal* g *that could model the prior of the agent or a preference given by an observer.* $P_r(ED_{\hat{\pi}})$ *is the preference an actor may have over available policies based on their expected distinctiveness cost.* Z *is the normalization constant such that* $\sum_{\hat{\pi}\in\Pi_{\mathbf{G}}} P_r(\hat{\pi}) = 1$

Intuitively, Eq. A.1 associates a probability $P_r(\hat{\pi})$ to each non-distinctive policy based on the number of satisfied goals as well as the probabilities of those goals being the true goal. Note that in case of a uniform prior distribution, $P_r(\hat{\pi})$ favors trajectories common to a higher number of goals.

Theorem 4. $ecd(P) \leq wcd(P)$

Proof. *wcd* is the *maximum* expected distinctiveness among all legal policies, while *ecd* is the *average* expected distinctiveness over those policies. Therefore, $ecd(P) \leq wcd(P)$ ■

Corollary 6. *If there is exactly one policy in the augmented MDP, then* $\text{ecd}(P) = \text{wcd}(P)$.

A.1.2 Computing *ecd*

From the analysis in Chapter 4, finding non-distinctive policies in a stochastic GR problem requires to account for all candidate goals (as opposed to pairs of goals in deterministic GR problems). Additionally, the set of possible goals at a given state s is not Markovian as it depends on the trajectory that reaches s. Therefore, we compute the *ecd* of an optimal setting of the S-GRD framework using the appropriate augmented MDP (Subsections 4.1.3, p. 64 and 4.2.1, p. 83). By Lemma 6 (Chapter 4, p. 87), the expected distinctiveness cost of a partial policy value is equal to the expected cost from a policy in the augmented MDP.

In general, we can determine the *ecd* value of a stochastic GR problem P using:

$$ecd(P) = \sum_{\hat{\pi} \in \Pi_{aug}} P_r(\hat{\pi}) V_{\hat{\pi}}(s'_0) \tag{A.2}$$

$$V_{\hat{\pi}}(s') = \sum_{s'' \in \mathbf{S}} \mathcal{T}(s', \hat{\pi}(s'), s'') \big[\mathcal{C}'(s', \hat{\pi}(s'), s'') + V_{\hat{\pi}}(s'') \big] \tag{A.3}$$

where Π_{aug} is the set of non-distinctive policies (Definition 16, p. 53), $P_r(\hat{\pi})$ is the probability of policy $\hat{\pi}$, $s'_0 = s_0 \cdot \langle T \dots T \rangle$ is the augmented initial state, $s' = s \cdot \langle pos_1 \dots pos_n \rangle$ is an

augmented state, and $V_{\hat{\pi}}(s'_0)$ is the expected cost for s'_0 with augmented policy $\hat{\pi}$ computed recursively using Eq. A.3.

Lemma 6 (p. 87) guarantees the existence of an augmented policy in Π_{aug} with the same expected cost as a non-distinctive policy of a partially-observable stochastic GR problem. We use Lemma 19 below to make the same guarantee for OS-GRD models.

Lemma 19. *Let π' be any policy for the augmented MDP Π_{aug} defined in Chapter 4 (Subsection 4.3.2, p. 109) and define the non-distinctive partial policy $\hat{\pi}$ for P by:*

$$\hat{\pi}(s) = \begin{cases} a & \pi'(s') = a \ \forall s' \in \mathbf{S}' : w(s') = s \\ & \wedge \ \forall s'_s \in \mathbf{S}' : \mathcal{T}'(s', \pi'(s'), s'_s) > 0 \\ \perp & \textit{otherwise} \end{cases} \tag{A.4}$$

Then $V_{\pi'}(s_0) = ED(\hat{\pi})$, that is, the expected value of policy π' in Π_{aug} at s_0 is equal to the expected distinctiveness of $\hat{\pi}$ in P.

Proof. Let $\mathbf{S}_{\hat{\pi}}$ be the set of states reached by any non-distinctive trajectory of $\hat{\pi}$ and $\mathbf{S}_{\pi'}$ the set of reachable states using augmented policy π'. By Definition 17, the *expected distinctiveness* of a non-distinctive policy $\hat{\pi}$ is its expected cost. From Eq. A.4, $\hat{\pi}$ starts at $s_0 = w(s'_0)$ and ends at states $s_n = w(s'_n) \mid \exists s'_s \in \mathbf{S}' : \mathcal{T}'(s'_n, \pi'(s'_n), s'_s) = 0$, which are the predecessors of

distinctive states. Therefore, $ED(\hat{\pi}) = V_{\hat{\pi}}(s_0)$, where:

$$V_{\hat{\pi}}(s) = \begin{cases} 0 & \text{if } \hat{\pi}(s) = \perp \\ \sum_{s_s \in \mathbf{S}_{\hat{\pi}}} \mathcal{T}(s, a, s_s)\big[\mathcal{C}_o(s, a, s_s) + V_{\hat{\pi}}(s_s)\big] & \textit{Otherwise} \end{cases} \tag{A.5}$$

By Eq. 3.9 (p. 55), the expected distinctiveness will not include costs of non distinctive trajectories. Hence, Eq. A.5 considers only successors reachable by non-distinctive trajectories.

On the other hand, $V_{\pi'}(s'_0)$ can be evaluated using:

$$V_{\pi'}(s') = \begin{cases} 0 & \text{if } s' \in \mathbf{G}' \\ \sum_{s'_s \in \mathbf{S}'} \mathcal{T}(s', a, s'_s)\big[\mathcal{C}'(s', a, s'_s) + V_{\pi'}(s'_s)\big] & \textit{Otherwise} \end{cases} \tag{A.6}$$

By construction of the augmented MDP, $\forall s', s'_s \in \mathbf{S}', \exists s, s_s \in \mathbf{S}_{\hat{\pi}} : s = w(s') \wedge s_s = w(s'_s) \wedge \mathcal{T}(s', a, s'_s) = \mathcal{T}(s, a, s_s) \wedge \mathcal{C}'(s', a, s'_s) = \mathcal{C}_o(s, a, s_s)$. If $s' \in \mathbf{S}'$ is not reachable using π', then $s = w(s') \notin \mathbf{S}_{\hat{\pi}}$, that is, s is not part of any non-distinctive trajectory, nor is the action reaching s. Therefore, Eqs. A.5 and A.6 provide the same values for all states $s \in \mathbf{S}_{\hat{\pi}}$ and $s' \in \mathbf{S}_{\pi'}$. Hence, $V_{\hat{\pi}}(s_0) = ED(\hat{\pi}) = V_{\pi'}(s'_0)$. ■

A.1.3 Computing *ecd* for Optimal Settings

We use a TVI-like algorithm, on an augmented MDP constructed using Algorithm 1 or 2, with a Bellman-like update defined as:

$$ecd(P) = V(s'_0) \tag{A.7}$$

$$V(s') = \sum_{\pi \in \Pi_{\mathbf{G}}} \sum_{s'' \in \mathbf{S}'} P_r(\pi(s')) \mathcal{T}'(s', \pi(s'), s'') \left[\mathcal{C}'(s', \pi(s'), s'') + V(s'') \right] \tag{A.8}$$

$$P_r(\pi(s')) = \frac{1}{Z} \sum_{g_i \in \mathbf{G}(\pi(s'))} P_r(g) \tag{A.9}$$

$$Z = \sum_{\pi \in \Pi_{\mathbf{G}}} \sum_{g \in \mathbf{G}(\pi(s'))} P_r(g) \tag{A.10}$$

where $P_r(g) > 0$ is the probability of the agent choosing goal g that satisfies action $\pi(s')$, and Z is the normalization constant such that $\sum_{\pi(s') | \pi \in \Pi_{\mathbf{G}}} P_r(\pi(s')) = 1$.