ROBOT DESIGN FROM THE PERSPECTIVE OF PLANNING AND ESTIMATION

A Dissertation

by

YULIN ZHANG

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Dylan A. Shell
Committee Members,	Dezhen Song
	Suman Chakravorty
	Thomas R. Ioerger
Head of Department,	Scott Schaefer

August 2021

Major Subject: Computer Science

ABSTRACT

Roboticists design robots by relying heavily on empirical knowledge and prior experience. That design process is seldom done in an automated fashion. This dissertation aims to automate robot design for robots for estimation and planning tasks. The work shows how to make decisions about robot hardware choices while jointly solving for a plan or estimator. To do so, this dissertation reasons about the information required by the task and the information that can be collected by a robot with a particular design. The robot should be able to provide sufficient information to accomplish a task. But such information can be both valuable and sensitive, and can be potentially leaked from the robot. Motivated by applications where privacy is important, this dissertation studies automated robot design for planning and estimation tasks, subject to additional constraints on the information that can be disclosed by or learned from the robot. It investigates the design of robot sensors, the information disclosure policy that determines how information is disclosed from the robot, and prior knowledge in privacy-preserving tracking and planning tasks. It also develops exact algorithms to construct estimators, which have minimum state complexity and achieve specified functionalities in the estimation tasks.

We first characterize the strategy space of one-dimensional privacy-preserving tracking problems, where a robot must track a target with its accuracy above a tracking bound and below a privacy bound. We present impossibility results for some tracking and privacy bounds, and show that the feasibility of the task is sensitive to the robot's initial belief. By characterizing sensor power as the number of its pre-images, we prove that the amount of solvable privacy-preserving problems is bounded, converging asymptotically with increasing sensing power.

Secondly, we examine privacy-preserving planning problems, where the robot should not only solve the planning problem but also reason about the disclosed information of the plan execution to a third party we call an observer—these are active variants of the passive problem we first consider. In this problem, we introduce the notion of information disclosure policy to create a knowledge gap between the observations perceived by the robot and the information received by the observer during plan execution. In addition, we also examine different assumptions about the observer's prior knowledge of the robot's plan, and present a family of algorithms to search for a plan and an information disclosure policy. Among the problems that are solved, the most challenging one is to search for a plan and information disclosure policy, while assuming the same plan is disclosed as prior knowledge to the observer. In this problem, before the plan is found, there is no explicit representation of the observer's prior knowledge, which makes the plan search problem challenging to formulate.

Next, we consider robot sensors abstractly. Due to the sensor noise, there is a gap between what the robot sees and the real event happening in the world. We model such a sensing gap with a formal structure that generalizes information disclosure policy, and this structure serves as an abstraction of the sensor. We further model sensor fabrication constraints (such as sensing fidelity) as properties of the abstraction, and propose search algorithms to enumerate all sensors that suffice for solving a planning problem.

Finally, we study the minimization problems of the combinatorial filters, which are the discrete estimators used by the observer in the privacy-preserving planning problems. We generalize the existing notion of combinatorial filters and examine the hardness of both deterministic and non-deterministic filter minimization problems. We show that multiple concepts previously believed to be true about combinatorial filters (and actually conjectured, claimed, or assumed to be) are in fact false. We then present the first known complete filter minimization algorithm, and further propose an algorithm that is efficient in practice.

To summarize, this dissertation includes impossibility results for sensor design in privacy-preserving tracking tasks, and algorithms for finding plans, sensor-plan pairs, and disclosure-plan pairs to enable automated design in privacy-preserving planning tasks. It also contributes counter-examples, constructions, hardness results, and algorithms to minimize the state complexity of combinatorial filters.

DEDICATION

To my parents, and friends.

ACKNOWLEDGMENTS

Ph.D. is a journey to the unknown. What we learn and who we travel with is more important than its destination. This thesis would never be done without those who are together with me in my Ph.D. career.

First, I must thank my parents for always encouraging me to pursue my passions. Although being thousands of miles away, they have been consistently and unconditionally supporting me.

I am very thankful for having Dr. Dylan Shell as my advisor. He is a true gentleman, always treating people (including his students and collaborators) with deep respect. He teaches me critical thinking and academic writing, shares with me his thoughts and opinions, and helps me develop academic aesthetic tastes. We share every happy moment when coming up with good ideas. We treat every work we have done as a piece of art, by cleaning every math formula, tweaking every word, and polishing every figure. His enthusiasm for learning is contagious.

I am also lucky to have the opportunity to work with Dr. Jason O'Kane from University of South Carolina. My dissertation is mostly inspired by Jason's earlier work, and he gives me a lot of support in both research and job application.

I would like to thank my committee members for their instructions and advice. I have taken Artificial Intelligence with Dr. Thomas Ioerger, Random Dynamic System with Dr. Suman Chakravorty, Computer Vision with Dr. Dezhen Song. They play an important role to help me build a solid understanding of Artificial Intelligence and Robotics, and they give insightful comments and suggestions on my dissertation.

I would also like to thank the current and previous fellow members of Distributed AI Robotics Lab for their support. I really enjoy the fun times at the reading groups, lunches, movie nights with Jung-Hwan Kim, Changjoo Nam, Young-Ho Kim, Sasin Janpuangtong, Rui Liu, Kai Qin, Taahir Ahmed, Reza Oftadeh, Eric Cochrane, Diptanil Chaudhuri, Ya-Chuan Hsu, Grace McFassel, and Reza Teshnizi. I also want to thank my collaborator Hazhar Rahmani, who is a very smart and pleasant person to work with.

I am also grateful for the delightful environment from the Department of Computer Science and Engineering. The lounge has always been my favorite place, where I meet a lot of interesting spirits including Binbin Li, Haifeng Jin, Xuesu Xiao, Chieh Chou, Hsin-min Cheng, Jiayi Huang. I would also like to take this opportunity to thank one of my best friends Niladri Das from Aerospace Engineering. He introduces me to Indian culture and food, and we together spend a lot of wonderful times exploring Bryan–College Station.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor Dylan A. Shell [advisor], Professor Dezhen Song, and Professor Thomas R. Ioerger of the Department of Computer Science and Engineering and Professor Suman Chakravorty of the Department of Aerospace Engineering.

Funding Sources

Graduate study was supported by the National Science Foundation (NSF) through grants IIS-1453652 and IIS-1527436, and the Chateaubriand Fellowship Program.

NOMENCLATURE

GPS	Global Positioning System
AI	Artificial Intelligence
ICRA	International Conference on on Robotics and Automation
RSS	Robotics: Science and Systems
PPC	Privacy Preserving Condition
TTC	Target Tracking Condition
1-dim.	One-dimensional
PP	Privacy Preserving
TT	Target Tracking
p-graph	procrustean graph
I-state	Information state
SDE	State Determined Expansion
SEEK$_{\boldsymbol{x}}$	Finding a plan
SEEK$_{\boldsymbol{\lambda}}$	Finding a label map
SEEK$_{\boldsymbol{x},\boldsymbol{\lambda}}$	Finding a plan and a label map
JPSD	Joint-Plan-Sensor-Design
VSO-FDM	Vertex Single-outputting Filter Minimization
MCCZC	Minimum Clique Cover with Zipper Constraints
FDM	Filter Minimization
GMCZC	Generalized Minimum Cover with Zipper Constraints
MIP	Mixed Integer Program
INP	Integer Nonlinear Program

ILP	Integer Linear Program
SAT	Boolean Satisfaction
LazySAT	Boolean Satisfaction with Just-in-time Constraints
CNF	Conjunctive Normal Form
NFA	Nondeterministic Finite Automaton
DFA	Deterministic Finite Automaton
PFM	P-filter Minimization
PFDM	P-filter Determinization and Minimization
PFM-DEC	Decision Problem of P-filter Minimization
PFDM-DEC	Decision Problem of P-filter Determinization and Minimization
Det	Deterministic
NDet	Nondeterministic
DF	Deterministic Filter
NF	Nondeterministic Filter
DOF	Degree Of Freedom
c-NMA	Coherent Nondeterministic Moore Automaton

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

As fundamental capabilities for autonomous robots, estimation and planning are realized as software modules that take sensor readings as input and output commands to the robot's actuators, so as to solve a particular task. This dissertation extends classical estimation and planning problems toward more general classes of tasks and toward the automated design of robots, including in making hardware selections: As summarized visually in Figure 1.1, at the task level, we consider estimation and planning tasks in an untrusted environment, where the robot executing the task is being observed. The objective is to constrain the information divulged from the robot as well as to allow the robot to accomplish its tasks. Approaching the level of hardware selection, this dissertation examines the estimation and planning tasks from a designer's perspective, and automatically reasons about sensor design jointly with the robot's plans and estimation strategies. By reasoning about the robot design jointly with planning and estimation, it yields better solvability for the tasks, and emphasizes the utility of limited sensors and the value of ignorance.

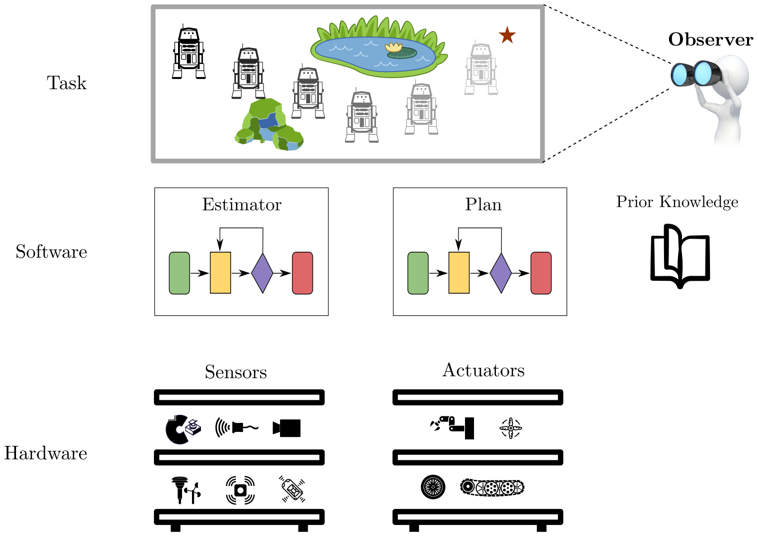


Figure 1.1: The core problem of study: informed design of robot hardware and software in order to achieve tasks. A particular (novel) focus is in considering this subject to constraints that may arise from the robot being observed.

1.1 Estimation and planning with stipulations on the information divulged

When interacting with the world, a robot often needs to collect information about its environment and itself with respect to the environment. Such information comes from the robot's prior knowledge (about the environment or its own dynamics) or sensor readings. Due to the inaccuracy in the available prior knowledge, sensor noise, and limited field of view, the information known by the robot is plagued by uncertainty. Indeed, uncertainty is one of the main sources of difficulties for robotic estimation and planning tasks. It corrupts estimates via noise making the estimation task challenging; it causes plans to fail by introducing non-deterministic or stochastic outcomes in the system and preventing the robot from choosing the right actions to reach its goal. Uncertainty is commonly treated as something that should be avoided, minimized, or even eliminated (if possible) in the robot's estimation and planning tasks. Bayesian techniques are used to build an unbiased estimate with the minimum variance [4]; robot's movements or sensing strategies are considered to improve the sensing performance [5]; planning algorithms are developed for the robot to act in the presence of uncertainty [6].

But as robots become widespread, it is likely that there will be a shift in thinking—robots that know too much are also problematic in their own way. A robot operating in your home, benignly monitoring your activities and your daily routine, for example to schedule vacuuming at unobtrusive times, possesses information that is valuable. There are certainly those who could derive profit from it. For example, in 2017, iRobot announced that they intended to sell maps of people's homes, as generated by their robot vacuum cleaners. The result was a public outcry [7]. It is increasingly clear that, as robots become part of our everyday lives, the information they could collect can be both sensitive and valuable.

When living in an untrusted environment, the robot is observed while executing its tasks, and the information possessed by the robot can get leaked via a variety of ways. For instance, the robot's information can be directly disclosed from its status display, which is designed to communicate necessary information to the user but can be misused if it is leaked to an adversary. The information may also be recorded in the robot's log file, which is designed to help a technician for maintenance purposes. The same log file may also be potentially accessed and used by an adversary to achieve its own purpose. We call the medium (e.g., status display or log file) through which information is disclosed, an information disclosure channel.

From the robot's perspective, it can control the divulged information via the information disclosure channel, its sensing strategy, or actions. The robot can decide what information is disclosed or how information is disclosed via the information disclosure channel. For example, the robot may choose what information to present in its status display, or it can inject ambiguity when recording information in its log file. On the other hand, the robot may choose to behave in a less informative way, so as to prevent sensitive information to be collected or divulged. For example, the robot can choose where to point its sensor and determine what information to be collected; the robot may also choose not to move through a sensitive location and avoid private information from being detected or divulged. Existing work focuses separately on finding appropriate sensing strategies [1], plans [8], or synthesizing strategies for the information disclosure channel [9, 10]. This dissertation examines those elements jointly in a privacy-preserving tracking or planning scenario, where the objective is to help the robot accomplish its target tracking or planning task and to constrain the divulged information from the robot.

1.2 Robot design for estimation and planning tasks

A second direction to approach the estimation and planning tasks is to solve them from the perspective of a designer. To solve tasks, we roboticists tend to approach robot sensors from the perspective of consumers, purchasing whatever seems necessary from a catalog, and then write code to make robots useful. This perspective puts practical constraints up front: it is influenced by technologies that are currently available, it limits options to what can be fabricated cheaply and sold profitably, and it is tied to a narrowed solution space that can affect the solvability and optimality of the tasks. Robot design aims to reason about the robot's hardware including its sensors and actuators that suffice for an estimation or planning task.

Currently, robot design is seldom done in an automated fashion. Roboticists have to reason about the robot's configuration (often only heuristically), then develop algorithms based on the given configuration, and repeat these steps until a task can be solved. To automate the design process, this dissertation examines ways in which sensors affect whether a tracking or a planning problem can be solved, and automatically searches for sensors that suffice for an estimation or planning task.

1.3 Design a robot for estimation and planning tasks with information stipulations

We have provided two directions to generalize existing estimation and planning problems: introducing stipulations on the divulged information on the robot's tasks, and considering searching for robot sensor designs. Though the two directions may seem at first blush to be independent, they are not. Introducing stipulations on the divulged information, especially privacy stipulations, teaches us about the less powerful sensors, which are underappreciated when designing robots for estimation and planning tasks. To achieve the robot's estimation and planning purposes, the sensors that are more powerful in terms of the information they can collect are always preferred, since they are more capable of satisfying the information requirement of the tasks [11]. However, privacy stipulations on the divulged information will limit the usage of those more powerful sensors and encourages the less powerful ones. For example, if the location of a robot carrying a Global Positioning System (GPS) sensor is sensitive and the GPS sensor reading is accessible to the adversary, then using a high-precision GPS sensor will immediately reveal the robot's position to the adversary. In this case, a low-precision GPS sensor that is sufficient for the robot's tasks is preferred.

Privacy stipulations on the divulged information constrain the information that the robot collects, which limits the usage of those more powerful sensors and encourages the less powerful ones. For example, the Global Positioning System (GPS) sensors with higher precision are always preferred for the robot to localize itself and take actions to reach its goal. But if sensing information is also accessible to the adversary, then it will reveal the robot's precise position to its adversary. By preventing the adversary from localizing the robot too precisely with stipulations, we encourage the usage of low-precision sensors in robot design.

On the other hand, introducing robot design improves the solvability of the tasks with information stipulations. Estimation and planning tasks may be infeasible due to poor sensors, which do not provide enough information and fail to work with any estimation or planning strategy to accomplish the robot's tasks. Adding constraints on the divulged information makes the solvability of such problems even worse. If the sensor is too powerful, it collects too much information as long as it is open and may immediately violate the privacy stipulations. As a consequence, sensor design is critical to improve the solvability of the estimation and planning tasks while satisfying the information stipulations, and it should be solved jointly with the robot's estimation or planning strategies.

1.4 Research objectives

The objective of this dissertation is to understand the information that can be communicated and learned from the robot. It contains the information that is communicated online during the robot's plan execution, and the information that is disclosed offline. We want to reason about how to constrain the communicated information in two ways: to prevent sensitive information from being leaked, and to explicitly communicate useful information if that is part of the robot's tasks. We want to understand the implications of those constraints for the robot's tasks, and develop algorithms to solve those tasks. Further, we want to understand the relationship between the robot's tasks and robot design choices. We want to provide algorithms that can tame the complexity of searching over the space of robot designs to find feasible estimation or planning strategies as well as satisfying the stipulations on the divulged information.

1.5 Research contributions

This dissertation makes the following contributions toward the research objectives:

First, we characterize a class of one-dimensional privacy-preserving tracking problems in terms of the robot's initial knowledge, sensor design, and stipulations. Inheriting the problem of target tracking whilst simultaneously preserving the target's privacy, as was proposed by O'Kane [1], who dubbed it the robot panda tracking scenario, elegantly illustrating the value of ignorance, we examine the problem from the design perspective. We give a complete characterization of the solvability of the one-dimensional panda tracking problem with different tracking and privacy bounds. Between the problems that can be solved and the non-solvable ones, we identify the cases where the solvability of the problem is sensitive to the robot's initial belief. By characterizing the sensor power as the number of its output classes, we prove asymptotic results in privacy-preserving tracking as we increase the sensor power. We also relate abstract sensors for scenarios to different dimensionalities and different number of pre-images.

Second, we develop exact algorithms to jointly search for information disclosure policies together with plans in privacy-preserving planning problems. This dissertation contributes the first formulation, to our knowledge, of planning where solutions can be constrained so as to require that some information be communicated and other information obscured subject to an adversarial model of an observer. Nor do we know of other work where both a plan and some notion of an interface (the information disclosure policy, in our terminology) can both be solved for jointly. We identify a hierarchy of the observer's prior knowledge about the robot's plan, and solve the privacy-preserving planning problems for each possible prior knowledge.

Third, we develop exact algorithms to search for sensor designs that suffice for privacy-preserving planning problems. This dissertation also contributes data structures, rooted in mathematical abstractions, to represent the space of sensors as covers, which enable whole sets of sensors to be summarized via a single special representative. Based on this representation, we proposed an algorithm to enumerate all feasible sensor designs, each of which solves the privacy-preserving planning problem jointly with some plan. Furthermore, we give a means by which other structures (either task domain knowledge, sensor technology, or fabrication constraints) can be incorporated to reduce the sets to be enumerated.

1.6 Additional contributions

In the work of both sensor design and privacy-preserving planning problems, we model the world, plans, and estimators as p-graphs [12]. P-graphs originally come from the formulation of discrete estimation problems, and they are then extended to represent plans in the planning problems. But the number of states in such p-graphs grows with the scale of the problem. One of the relevant questions arising form the prior work is how to reduce the number of states in the graphs, which has direct implications for the minimization of resource consumption.

This dissertation contributes hardness results for a family of combinatorial filter minimization problems, and proposes exact and practical algorithms for deterministic filter minimization. Also it shows that multiple concepts previously believed to be true about combinatorial filters (and actually conjectured, claimed, or assumed to be) are in fact false. For instance, minimization does not induce an equivalence relation. We give the first known exact algorithm for the filter minimization problem. The algorithm also generalizes naturally to cover filter minimization, a larger class of instances. We further propose a practical filter minimization algorithm by treating the constraints just-in-time, which yields improvements in efficiency and has the potential to minimize larger filters.

1.7 Outline

The rest of this dissertation is organized as follows: We present related work in Chapter 2. In Chapter 3, we give a complete characterization of one-dimensional privacy-preserving problems and present an impossibility result. We examine privacy-preserving planning problems in Chapter 4, and develop algorithms to search for plans and information disclosure policy. In Chapter 5, we propose an abstraction for sensors, incorporate sensor fabrication constraints in the abstraction, and present an algorithm to enumerate all sensors that suffice for a planning problem. Chapter 6 presents hardness results and algorithms to minimize combinatorial filters.

2. RELATED WORK

Motivated by privacy applications, this dissertation reasons about the information communicated from the robot, and solves for the robot's tasks from a robot design perspective. In this chapter, we will mainly present the work related to privacy, actions that communicate information, and robot design.

2.1 Protecting private information

As we interact with devices such as computers, phones, and robots, our information is being collected and processed to provide better service. But some information may be sensitive and it could put our privacy in danger if leaked. To limit the divulgence of sensitive information in data processing, a great deal of work, above and beyond the straightforward application of cryptographic techniques, has been investigated to encrypt the data and prevent it from being accessed by unauthorized parties [13]. For applications where computation needs to be done on the private data and no secret can be distributed a priori, homomorphic encryption techniques are developed to allow computation on the encrypted data [14]. For applications where some statistics about the private information need to be released, differential privacy techniques are used to make sure no private information can be learned from those statistics [15].

Individuals often need to interact with others in the same network, and extensive work has been done to prevent privacy information from being leaked through the network. In a peer-to-peer network, routing protocols have been proposed to increase anonymity [16]. Spatial and temporal privacy in wireless sensor networks have also been the subject of study [17, 18]. In a social network, privacy-preserving techniques are developed to control user privacy in online services while preserving some functionality from the online services [19].

Traditionally within the AI and robotics communities, autonomous agents are modeled as transducers or dynamic systems that receive percepts (i.e., observations) from their environment, which they use to choose actions to take in order to influence the world. A great deal of work examines how to reason about their behaviors, beliefs, goals, and planning processes [20, 21] on the basis of some observed portions of their interaction history. In fact, a recent special issue in the journal *Artificial Intelligence* focuses on the topic of "Autonomous Agents Modeling Other Agents" (the issue is still to appear, see [22]). Epistemic logic is used to reason about and constrain the knowledge of the agents in a team [23, 24]. In tracking tasks, agents only receive information from the world and are modeled geometrically [1], via discrete filters [25], or with Bayesian filters [10, 26, 27] to build an estimate about the world. The estimation result is disclosed, or partially disclosed, to the observer. Stipulations (constraints) or costs (optimization penalties) are introduced to confine the resulting estimates or outputs of the filters, thereby preventing the observer from inferring sensitive information or knowing too much. There is also work that guarantees privacy via appropriate sensor selection [28].

Additionally, policies about how events during plan execution are disclosed from the agents to the observer are also exploited, so as to satisfy privacy and utility specifications [29, 30]. In multiagent planning, agents are searching for their own plans in a distributed manner. Coordinating the search process may require that information be shared among these agents. In these circumstances, some agents' actions and state information might be deemed private, thus some work has examined privacy-constrained coordinated plan search [31, 32].

2.2 Communicating information from the robot's behavior

Information can be communicated from the robot's behavior. An important topic in human-robot interaction is expressive action (e.g., see [33]). In recent years there has been a great deal of interest in mathematical models that enable generation of communicative plans. Important formulations include those of [34, 35], proposing plausible models for human observers (from the perspectives of presumed cost efficiency, surprisal, or generalizations thereof). In this prior work, conveying information becomes part of an optimization objective, whereas we treat it as a constraint instead. Both [34] and [35] are probabilistic in nature, here we consider a worst-case model that is arguably more suitable for privacy considerations: We ask what an observer can plausibly infer via the history of its received observations.

When agents are actively executing actions to achieve their goals, the observer may be particularly interested in predicting this goal before the end of plan execution. Deceptive actions [36, 37] and goal obfuscated plans [8, 38, 39] aim to hide the agent's true goal for as long as possible, while predictable plans make their goals predictable to their human collaborators [40].

2.3 Robot design

Approaches for automated design of robots have been the subject of three recent workshops at RSS and ICRA over the last 3 years [41]. Current research examines aspects of hardware fabrication (e.g., 3D-printing [42] and prototyping [43, 44]), interconnection optimization [45], rapid end-to-end development and deployment [46, 47], automated synthesis (jointly for mechanisms and controllers) from specifications of desired capabilities [48], optimization subject to functionality–resource interdependencies [49, 50], interactive design decision support [51, 52], and automated material design [53].

A rich history of robotics research has examined the information required to accomplish a particular task, including specifically what sensors ought to provide [54]. Since sensors can be costly and unreliable, important early papers explored how one might forgo them entirely [55, 56]; other work examined how one might reason about sensors to establish that they do provide enough information [57, 11]. We are interested in all possible sensors, including hypothetical ones, that provide adequate information to solve the planning problem. Imperfection in sensors is modeled as conflation in the perceived events. This conflation is usually considered to be transitive in existing work [11, 58, 39], when reasoning about the information through the sensors. This dissertation describes sensors via a sensor map representation which can model non-transitive conflation, in the spirit of [59, 12]. It contributes methods to search for all sensors such that there exists a plan for each one to accomplish a given task.

2.4 Other related techniques

In this dissertation, we are influenced by the philosophy of LaValle [6], following his use of the term information state (I-state) to refer to a representation of information derived from a history of observations.* We use the information state to refer both the robot's belief and the observer's estimate. The information state for discrete state space is a subset of consistent states. In this dissertation, we use procrustean graphs (or p-graphs) [12] to represent the transition of the states in the world, the robot's plan, and the observer's estimator. Then the information state for both the robot and the observer can be constructed based on these p-graphs.

In the p-graph representation, the robot's action–observation history becomes a sequence that can be traced in the graph. The planning problems with information stipulations can be specified as the existence of some sequences that have a particular goal attainment behavior, which matches the functionality of model checking tools [60]. In this dissertation, we will cast the planning problems into linear temporal logic (LTL) formulas and use model checking tools to solve them.

When p-graphs or graphs serve as representations for estimators, which are called combinatorial filters, one of the questions raised by prior work is filter minimization, which aims to minimize the number of states in the filters. Despite the apparent similarity of combinatorial filter minimization to the problem of state minimization of deterministic automata, with Myhill–Nerode's famous and efficient reduction [61], minimization of combinatorial filters is NP-complete [62]. All existing work studies filter minimization based on merger operations. These algorithms reduce filter minimization to a graph coloring instance [62, 2, 63] or integer linear programming [64]. Saberifar et. al. [63] examined special cases, approximation and parameterized complexity of filter minimization. Rahmani and O'Kane [65] showed that the well-known notion of *bisimulation* relation in general yields only sub-optimal solutions. They proposed three different integer linear programming formulations to search for the smallest equivalence relation [64].

*In his invited talk at WAFR'18, LaValle pointed out that his use of the term was based on earlier use by O. Morgenstern and J. von Neumann.

3. ROBOT DESIGN IN ONE-DIMENSIONAL PRIVACY-PRESERVING TRACKING[*]

This chapter focuses on the problem of tracking a target whilst preserving the target's privacy. Though fairly narrow, this is a crisply formulated instance of the broader dilemma of balancing the information a robot possesses, for the robot must maintain some estimate of the target's pose, but information that is too precise is an unwanted intrusion and potential hazard if leaked. By fully characterizing the robot's tracking strategies for one-dimensional problems, we are able to examine the solvability of the problems from robot design perspective, including the robot's sensing capability, prior knowledge, and information about the target. We contribute impossibility results about the solvability of the one-dimensional problems, and generalize them to high-dimensional ones.

To illustrate the problem and idea, we follow the robotic panda tracking scenario, which is introduced by O'Kane to express the idea of uncertainty being valuable, aloofness having utility. The following, quoted verbatim from [1, p. 235], describes the scenario:

> "A giant panda moves unpredictably through a wilderness preserve. A mobile robot tracks the panda's movements, periodically sensing partial information about the panda's whereabouts and transmitting its findings to a central base station. At the same time, poachers attempt to exploit the presence of the tracking robot—either by eavesdropping on its communications or by directly compromising the robot itself—to locate the panda. We assume, in the worst case, that the poachers have access to any information collected by the tracking robot, but they cannot control its motions. The problem is to design the tracking robot so that the base station can record coarse-grained information about the panda's movements, without allowing the poachers to obtain the fine-grained position information they need to harm the panda."

Note that it is not sufficient for the robot to simply forget or to degrade sensor data via post-processing because the adversary may have compromised these operations, possibly writing the

[*] Reprinted with permission from "Complete characterization of a class of privacy-preserving tracking problems" by Yulin Zhang and Dylan Shell, 2019. *International Journal of Robotics Research*. 38(7):299–315. Copyright 2019 by *Sage*.

information to separate storage.

One can view the informational constraints as bounds:

1. A maximal- or upper-bound specifies how coarse the tracking information can be. The robot is not helpful in assuring the panda's well-being when this bound is exceeded.

2. A second constraint, a lower-bound, stipulates that if the information is more fine-grained than some threshold, a poacher may succeed in having his wicked way.

The problem is clearly infeasible when the lower-bound exceeds the upper-bound, but what of other circumstances? Is it always possible to ensure that one will satisfy both bounds indefinitely? In the original paper, [1] proposed a tracking strategy for a robot equipped with a two-bit quadrant sensor, showing its successful operation in several circumstances. As no claim of completeness was made, one might well ask: will his strategy work for all feasible bounds? And how are the strategies affected by improvements in the sensing capabilities of the robot? These are the class of questions that are of interest to us.

The last of the preceding questions suggests another, more fundamental, one worth asking—what sensors are appropriate for a given problem? The problem of establishing the minimal information required to perform a particular task and the problem of analyzing the trade-off between information and performance, despite both being fundamental challenges, remain poorly understood territory — the efforts of [57], [55], and [56] notwithstanding. As a consequence, we do not attempt to answer this very general question. However, recognising that little has been said regarding scenarios like the one we study, where too much information can be detrimental, we are able to shed some light on the complexity of the sensors involved by examining the relationship played by our sensor's preimages. While a several authors have examined sensor power from a preimage perspective before [54, 66], there are two aspects which are novel in problem we study: (i) Our abstract sensor is parameterized, allowing one to change the number of output classes, and we have a situation where taking the limit of increasing power is instructive (see Theorem 3.2, Theorem 3.3, and Corollary 3.1) (ii) We relate abstract sensors for scenarios with different dimensionalities and different numbers of preimages (see Theorem 3.4 and comment thereafter).

3.1 One-dimensional panda tracking problem

The original problem was posed in two dimensions: the robot and panda, inhabiting a plane that is assumed to be free of obstacles, both move in discrete time-steps, interleaving their motions. A powerful adversary, who is interested in computing the possible locations of the panda, is assumed to have access to the full history of information. Any information is presumed to be used optimally by the adversary in reconstruction of possible locations of the panda — by which we mean that the region that results is both sound (that is, consistently accounted for by the information) but is also tight (no larger than necessary). The problem is formulated without needing to appeal to probabilities by considering only worst-case reasoning and by using motion- and sensor-models characterized by regions and applying geometric operations.

Information stipulation: The tracking and privacy requirements were specified as two disks. The robot is constrained to ensure that the region describing possible locations of the panda always fits inside the *tracking disk*, which has the larger diameter of the two. The *privacy disk* imposes the requirement that it always be possible to place the smaller disk wholly inside the set of possible locations.

Sensor model: As reflected in the title of his paper, O'Kane considered an unconventional sensor that consists of four IR sensors*, and outputs only two bits of information per measurement. With origin centered on the robot and an axis parallel to the robot's heading, the sensor outputs the quadrant containing the panda.

Target motion model: The panda moves unpredictably with bounded velocity. After a time-step has elapsed, the set of newly feasible locations for the panda is obtained by convolving a disk with the previous time-step's region. The disk must be sized appropriately for the time-step interval and the target's velocity.

*For the reader especially interested in O'Kane's physical robot implementation: he describes tracked item (i.e., the panda) carrying four beacons arranged to give 360° coverage, and four infrared sensors, clustered in the center of the robot, angled so that each senses a quadrant.

Now, by way of simplification, consider pandas and robots that inhabit obstacle-free one-dimensional worlds, each only moving left or right along a line.

Information stipulation: Using the obvious one-dimensional analogue, now the robot tracker has to bound its belief about the panda's potential locations to an interval of minimum size r_p (p for privacy) and maximum size r_t (t for tracking).

Sensor model: Most simply, the quadrant sensor corresponds to a one-bit sensor indicating whether the panda is on the robot's left- or right-hand side. When the robot is at u_1, the sensor indicates whether the panda is within $(-\infty, u_1]$ or (u_1, ∞).

In what follows, we will also explore how modifying the robot's sensing capabilities alters its possible behavior. Thus, we give the robot c set-points $u_1 < u_2 < \cdots < u_c$, each within the robot's control, so that it can determine which of the $c + 1$ non-overlapping intervals contains the panda. With c set-points one can model any sensor that fully divides the state space and produces at most $c + 1$ observations. Note that this is not a model of any physical sensor of which we are aware. The reader, finding this too contrived, may find later (e.g., for n-dimensional tracking, see Lemma 3.13) that the choice has merit. The case with $c = 1$ is the straightforward analogue of the quadrant sensor. Increasing c yields a robot with greater sensing power, since the robot has a wider choice of how observations should inform itself of the panda's location.

Target motion model: The convolution becomes a trivial operation on intervals.

Figure 3.1 provides a visual example with $c = 2$. The panda is sensed by the robot as falling within one of the following intervals: $(-\infty, u_1]$, $(u_1, u_2]$, (u_2, ∞), where $u_1, u_2 \in \mathbb{R}$ and $u_1 < u_2$. These three intervals are represented by observation values: 0, 1 and 2. For simplicity, no constraints are imposed on the robot's motion and we assume that at each time-step, the robot can pick positions of $u_1 < \cdots < u_c$ as it likes.

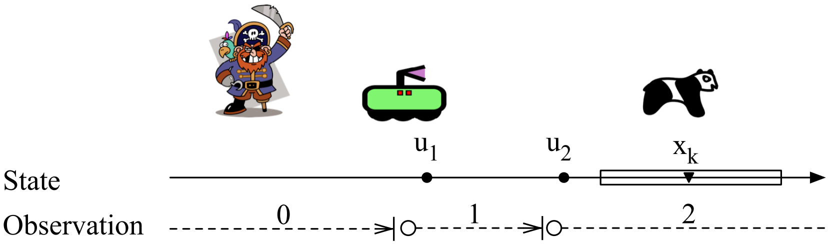


Figure 3.1: Panda tracking in one dimension. Inimitable artwork for the robot and panda is adapted from the original in [1, p. 2].

Notation and model:

In the 1-dim. problem the panda's location is represented as a single coordinate indexed by discrete time. At stage k the location of the panda is $x_k \in \mathbb{R}$. The robot (and adversary) maintains knowledge of the panda's possible location after it moves and between sensing steps by fusing the knowledge accumulated about the panda's possible location, sensor readings, and the movement model. The set of conceivable locations of the panda (a subset of $\mathbb{R}$) is a geometric realization of an information-state or I-state, as formalized and treated in detail by [67]. In this paper, we take the I-state as an interval.

The movement of the panda per time-step is bounded by length $\frac{\delta}{2}$, meaning that the panda can move at most $\frac{\delta}{2}$ in either direction. We use η_k to denote the robot's knowledge of the panda after the observation taken at time k. In evolving from η_k to η_{k+1} the robot's I-state first transits to an intermediate I-state, which we write η^-_{k+1}, representing the state after adding the uncertainty arising from the panda's movement but before observation $k+1$. Since this update occurs before the sensing information is incorporated, we refer to η^-_{k+1} as the *prior* I-state for time $k+1$. Updating I-state η_k involves mapping every $x_k \in \eta_k$ to $[x_k - \frac{\delta}{2}, x_k + \frac{\delta}{2}]$, the resultant I-state, η^-_{k+1}, being the union of the results.

Sensor reading updates to the I-state depend on the values of $u_1(k), u_2(k), \ldots, u_c(k)$, which are under the control of the robot. The sensor reports the panda's location to within one of the $c+1$ non-empty intervals: $(-\infty, u_1(k)], (u_1(k), u_2(k)], (u_2(k), u_3(k)], \ldots, (u_c(k), \infty)$. If we represent the

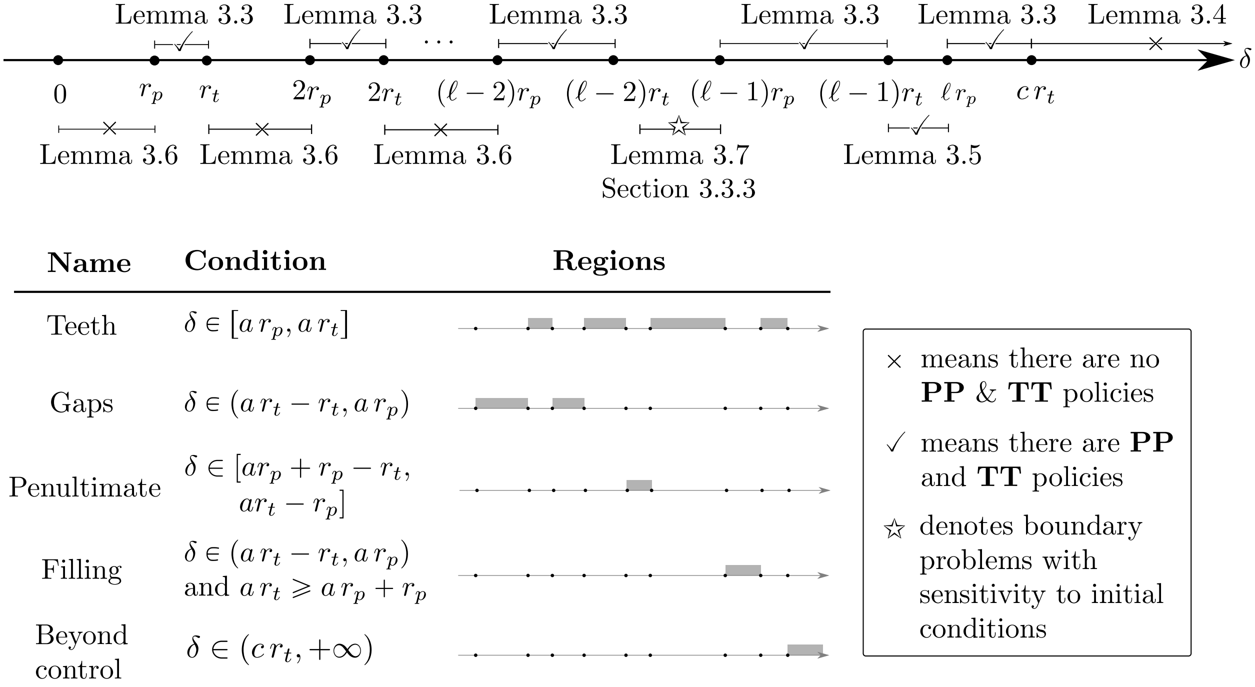

Lemma 3.3
Lemma 3.3
Lemma 3.3
Lemma 3.3
Lemma 3.3
Lemma 3.4
δ
0
r_p
r_t
$2r_p$
$2r_t$
$(\ell-2)r_p$
$(\ell-2)r_t$
$(\ell-1)r_p$
$(\ell-1)r_t$
ℓr_p
$c\,r_t$
Lemma 3.6
Lemma 3.6
Lemma 3.6
Lemma 3.7
Section 3.3.3
Lemma 3.5
Name
Condition
Regions
Teeth
$\delta \in [a\,r_p, a\,r_t]$
Gaps
$\delta \in (a\,r_t - r_t, a\,r_p)$
Penultimate
$\delta \in [ar_p + r_p - r_t, ar_t - r_p]$
Filling
$\delta \in (a\,r_t - r_t, a\,r_p)$ and $a\,r_t \geqslant a\,r_p + r_p$
Beyond control
$\delta \in (c\,r_t, +\infty)$
× means there are no **PP** & **TT** policies
✓ means there are **PP** and **TT** policies
☆ denotes boundary problems with sensitivity to initial conditions

Figure 3.2: Roadmap of the results for 1-dim. panda tracking with c sensing parameters.

observation at time k as a non-empty interval $y(k)$ then the *posterior* I-state η_k is updated as $\eta_k = \eta_k^- \cap y(k)$.

For every stage k the robot chooses a sensing vector $\mathbf{v_k} = [u_1(k), u_2(k), \ldots, u_c(k)]$, $u_i(k) < u_j(k)$ if $i < j$, $u_i(k) \in \mathbb{R}$, so as to achieve the following conditions:

1. *Privacy Preserving Condition (PPC)*: The size of any I-state $\eta_k = [a, b]$ should be at least r_p. That is, for every stage k, $|\eta_k| = b - a \geq r_p$.

2. *Target Tracking Condition (TTC)*: The size of any I-state $\eta_k = [a, b]$ should be at most r_t. That is, for every stage k, $|\eta_k| = b - a \leq r_t$.

3.2 Solving one-dimensional privacy-preserving tracking problems

Given specific problem parameters, we are interested in whether there is always some $\mathbf{v_k}$ that a robot can select to track the panda while satisfying the preceding conditions.

Definition 3.1. A 1-dim. panda tracking problem is a tuple $P_1 = (\eta_0, r_p, r_t, \delta, c)$, in which

1) the initial I-state η_0 describes all the possible initial locations of the panda;

2) the privacy bound r_p gives a lower bound on the I-state size;

3) the tracking bound r_t gives a upper bound on the I-state size;

4) parameter δ describes the panda's (fastest) motion; and

5) the sensor capabilities are given by the number c.

Definition 3.2. The 1-dim. panda tracking problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$ is privacy preservable, written as predicate **PP**(P_1), if starting with $|\eta_0| \in [r_p, r_t]$, there exists some strategy π to determine a $\mathbf{v_k}$ at each time-step, such that the Privacy Preserving Condition holds forever. Otherwise, the problem P_1 is not privacy preservable: $\neg$ **PP**(P_1).

Definition 3.3. The 1-dim. panda tracking problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$ is target trackable, **TT**(P_1), if starting with $|\eta_0| \in [r_p, r_t]$, there exists some strategy π to determine a $\mathbf{v_k}$ at each time-step, such that the Target Tracking Condition holds forever. Otherwise, the problem P_1 is not target trackable: $\neg$ **TT**(P_1).

To save space, we say a problem P_1 and also its strategy π are **PP** if **PP**(P_1). Similarly, both P_1 and its strategy π will be called **TT** if **TT**(P_1). Putting aside trivially infeasible P_1 where $r_p > r_t$, we wish to know which problems are both **PP** and **TT**. Next, we explore the parameter space to classify the various classes of problem instances by investigating the existence of strategies.

3.2.1 Roadmap of technical results

The results follow from several lemmas. The roadmap in Figure 3.2 provides a sense of how the pieces fit together to help the reader keep track of the broader picture.

Our approach begins, firstly, by identifying particular actions which we call *basis actions*: one increases the size of the I-state and the other does the opposite. We show that any question about the existence of a strategy can be framed in terms of sequences comprised of these two actions. Next, we divide the space of strategies into 'teeth' and 'gaps' according to the panda's speed. The teeth regions and the last gap region describe under-constrained problems that have **PP** and **TT** tracking strategies for all initial I-states that satisfy $|\eta_0| \in [r_p, r_t]$. The region with uncertainty that grows so quickly as to be beyond control and the remaining gaps (except the penultimate one) are over-constrained problems that have no **PP** and **TT** tracking strategies for any initial I-states satisfying $|\eta_0| \in [r_p, r_t]$. The second-to-last gap describes boundary problems that transition between the under-constrained problems and over-constrained ones. To proceed further, one needs to consider circumstances where the existence of a suitable strategy depends on the initial information available to the robot. These are a logically distinct class of solution and are, therefore, presented in a major section of their own (Section 3.3.3). Before that segue, a summary of the results established up to that point, including a visual representation of the parameter space, is provided; it may aid the reader to glance ahead to Section 3.3.1 and Figure 3.6.

3.2.2 Basis actions for privacy-preserving tracking

A challenge in dealing with 1-dim. panda tracking is the fact that the space of sensor configurations (i.e., the choices of $\mathbf{v_k}$) is continuous, making it infeasible to search through all choices. To resolve this issue, consider those sensor configurations which split the prior I-states into parts of equal size. Thinking in worst-case terms, a suicidal panda would choose the least convenient place to move to. Thus, compared with unevenly partitioned configurations, evenly partitioned configurations have both a “finer” largest I-state and a “safer” smallest I-state. Moreover, evenly sized splits mean that the size of the I-state after the observation is determined and depends only on the number of intervals.

Let $s(i)$ denote the choice of evenly dividing the prior I-state interval into i parts (the mnemonic is s for *split*). Then the following lemma states that it is sufficient to consider strategies consisting solely of $s(i)$ actions to determine whether violation of either the privacy or tracking constraints is inevitable.

Lemma 3.1. For any problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$, $\mathbf{PP}(P_1) \wedge \mathbf{TT}(P_1)$ if and only if there exists a **PP** and **TT** strategy that only consists of action $s(i)$, where $i \in \{1, 2 \ldots, c+1\}$.

Proof. $\Leftarrow$: Holds trivially. $\Rightarrow$: For **PP** and **TT** strategy π, then we can always construct a **PP** and **TT** strategy π' consisting of actions $s(i)$, where $i \in \{1, 2 \ldots, c+1\}$. Under strategy π, the sensing vector at stage k is $\mathbf{v_k}$, and the prior I-state is divided into i_k parts. The smallest and largest possible sizes of the resulting I-state are denoted as $\min\limits_{\eta \in \mathbf{v_k}} |\eta|$ and $\max\limits_{\eta \in \mathbf{v_k}} |\eta|$, respectively. Then action $\mathbf{v'_k} = s(i_k)$ splits the prior I-state equally into i_k parts. Both the smallest and largest size of the result are equal to the average size:

$$\min_{\eta \in \mathbf{v'_k}} |\eta| = \max_{\eta \in \mathbf{v'_k}} |\eta| = \frac{|\eta_k| + \delta}{i_k}.$$

Since PPC and TTC bounds must be satisfied for all motions of the panda, the I-states arising from $\mathbf{v'_k}$, which are intermediate sized with $\min\limits_{\eta \in \mathbf{v_k}} |\eta| \leq \frac{|\eta_k|+\delta}{i_k} \leq \max\limits_{\eta \in \mathbf{v_k}} |\eta|$, cannot introduce a violation where none existed in π. □

Hence, instead of considering all possible choices for set-points $u_1 < u_2 < \cdots < u_c$, we only need to consider those in $\bigcup\limits_{1 \leq i \leq c+1} s(i+1)$.

The actions can be categorized into two types depending on the number of parts the prior I-state is divided into. As shown in Figure 3.3, there are actions that increase the I-state size, and those that decrease its size. The distinguishing feature, which we denote with a, is the maximum number of r_t's contained within the distance the panda can move in a single time-step, namely $a = \left\lceil \frac{\delta}{r_t} \right\rceil$. From this definition it follows that $\delta \in (ar_t - r_t, ar_t]$. Since $|\eta_k| \in [r_p, r_t]$, if $\delta \in [ar_p, ar_t]$, then action $s(a)$ will always guarantee that the I-state's size is admissible: $|\eta_{k+1}| = \frac{|\eta_k|+\delta}{a} \in [r_p, r_t]$. For $\delta \in (ar_t - r_t, ar_p)$, the size of the I-state that results from action $s(a)$ is $|\eta_{k+1}| = \frac{|\eta_k|+\delta}{a} > |\eta_k|$. The size of the resulting I-state under action $s(a+1)$ is $|\eta_{k+1}| = \frac{|\eta_k|+\delta}{a+1} < |\eta_k|$. Furthermore the size of the resulting I-state decreases with the parameter of the $s(\cdot)$ action. When the number partitions is less than or equal to a, the I-state size will increase; when the number of partitions exceeds a, the size of I-state will decrease.

Next, we reduce the action space into basis actions $s(a)$ and $s(a+1)$, which we find it useful to denote as $\oplus$ and $\ominus$, respectively.

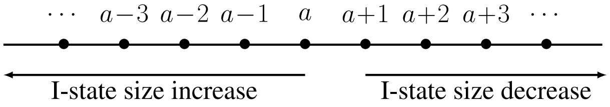


Figure 3.3: The change of the resulting I-state's size when splitting into i equal parts with $\delta \in (ar_t - r_t, ar_p)$.

Lemma 3.2. For any 1-dim. problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$, there exists a **PP** and **TT** strategy consisting solely of actions of $s(i)$ if and only if there exists a **PP** and **TT** strategy which uses actions $\oplus$ and $\ominus$ only.

Proof. $\Leftarrow$: Holds as $\oplus = s(a)$ and $\ominus = s(a+1)$.

$\Rightarrow$: For any **PP** and **TT** strategy π consisting of $s(i)$ actions, we construct π' as follows. For each action $\mathbf{v_k}$ under strategy π, suppose $\mathbf{v_k}$ splits the prior I-state into i_k parts. Construct the new action $\mathbf{v'_k}$ for π', so that

$$\mathbf{v'_k} = \begin{cases} \oplus & \text{if } i_k \leq a, \\ \ominus & \text{if } i_k \geq a+1. \end{cases}$$

For the first case, where $i_k \leq a$, we have $r_p \leq |\eta_k| \leq \max\limits_{\eta \in \mathbf{v'_k}} |\eta| \leq \max\limits_{\eta \in \mathbf{v_k}} |\eta| \leq r_t$ And, thus one has that $\min\limits_{\eta \in \mathbf{v'_k}} |\eta| = \max\limits_{\eta \in \mathbf{v'_k}} |\eta| \in [r_p, r_t]$. For the second case, where $i_k \geq a+1$, this condition also holds similarly. The resulting I-state at each time-step under π' satisfies both bounds, and π' is a **PP** and **TT** strategy.

□

Theorem 3.1. For any tracking problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$, $\mathbf{PP}(P_1) \wedge \mathbf{TT}(P_1)$ if and only if there exists a **PP** and **TT** strategy that consisting only of actions $\oplus$ and $\ominus$.

Proof. Combine the results from Lemma 3.1 and 3.2. □

3.2.3 Characterizing the solvability of the problems

In this section, we follow the roadmap in Figure 3.2.

Lemma 3.3. Let $P_1 = (\eta_0, r_p, r_t, \delta, c)$ be any 1-dim. panda tracking problem, if $\delta \in [ar_p, ar_t]$, where $a \in \mathbb{Z}^+$, $a \leq c$, then $\mathbf{PP}(P_1) \wedge \mathbf{TT}(P_1)$.

Proof. A **PP** and **TT** strategy is given. For any $|\eta_k| \in [r_p, r_t]$ the prior I-state has size $|\eta_{k+1}^-| = |\eta_k| + \delta$. Since $\delta \in [ar_p, ar_t]$, where $a \leq c$, $|\eta_{k+1}^-| \in [ar_p + r_p, ar_t + r_t]$. By taking action $\ominus$, which is possible since $a \leq c$, we get $|\eta_{k+1}| = \frac{1}{a+1}|\eta_{k+1}^-| \in [r_p, r_t]$. That is, if $|\eta_0| \in [r_p, r_t]$ and we take action $\ominus$, then $\forall k, |\eta_k| \in [r_p, r_t]$. Therefore, there exists a strategy (always take action $\ominus$) for P_1, so that the privacy-preserving tracking conditions *PPC* and *TTC* are always both satisfied when $\eta_0 \in [r_p, r_t]$. □

Lemma 3.4. For any 1-dim. panda tracking problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$, if $\delta \in (cr_t, \infty)$, then $\neg$ **TT**(P_1).

Proof. The tracking stipulation is proved to be violated eventually. Given the constraint of c sensing parameters, the prior I-state $|\eta_{k+1}^-| = |\eta_k| + \delta$ can be divided into at most $c + 1$ parts. Among these $c + 1$ posterior I-states, if c of them reach the maximum size r_t, the size of the remaining I-state is $|\eta_k| + \delta - cr_t$. If none of the resulting I-states violate the tracking bound, then the size of the smallest resulting I-state is $|\eta_k| + \delta - cr_t$. Since $\delta > cr_t$, the size of the smallest resulting I-state must increase by some positive constant $\delta - cr_t$. After $\left\lceil \frac{r_t - r_p}{\delta - cr_t} \right\rceil$ stages, the I-state will exceed r_t. So it is impossible to ensure that the tracking bound will not eventually be violated.

□

Lemma 3.5. For 1-dim. panda tracking problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$, if $\delta \in (ar_t - r_t, ar_p)$, where $a \in \mathbb{Z}^+$, $a \leq c$ and $ar_t \geq ar_p + r_p$, then **PP**$(P_1) \wedge$ **TT**(P_1).

Proof. A **PP** and **TT** strategy is given in this proof. Since $\delta \in (ar_t - r_t, ar_p)$ and $ar_t > ar_p + r_p$, $|\eta_{k+1}^-| = |\eta_k| + \delta \in (ar_p, ar_t + r_t) \subset L_1 \cup L_2$, where $L_1 = [ar_p, ar_t]$ and $L_2 = [ar_p + r_p, ar_t + r_t]$. If action $\oplus$ is performed when $|\eta_{k+1}^-| \in L_1$ and $s(a+1)$ is performed when $|\eta_{k+1}^-| \in L_2$, the resulting I-state satisfies $|\eta_{k+1}| \in [r_p, r_t]$. Hence, there is a strategy consisting of $\oplus$ and $\ominus$, for the problem P_1 such that the *PPC* and *TTC* are always both satisfied when $\eta_0 \in [r_p, r_t]$. □

Lemma 3.6. For any 1-dim. panda tracking problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$, if $\delta \in (ar_t - r_t, ar_p)$, where $a \in \mathbb{Z}^+$, $a \leq c$, $ar_t < ar_p + r_p$, then $\neg$ **PP**$(P_1) \vee \neg$ **TT**(P_1) when either: (i) $\delta > ar_t - r_p$ or (ii) $\delta < (a+1)r_p - r_t$.

Proof. In case (i), $r_p > ar_t - \delta$, so the size of the prior I-state satisfies $|\eta^-_{k+1}| = |\eta_k| + \delta > r_p + \delta > ar_t$. That is, if we take action $\oplus$ and divide η^-_{k+1} into a parts, the largest posterior I-state η_{k+1} will violate the tracking stipulation at the next time-step. If we take action $\ominus$ and divide η^-_{k+1} into $a+1$ parts, it can be shown that the smallest I-state will eventually violate the privacy stipulation. Under the action $\ominus$, the smallest size of the resulting posterior I-state $|\eta_{k+1}|_{\text{smallest}} = \min\limits_{\eta \in \mathbf{v_k}} |\eta|$ is no greater than the average size $\frac{|\eta_k|+\delta}{a+1}$. The amount of decrease is $\Delta_- = |\eta_k| - |\eta_{k+1}|_{\text{smallest}} \geq |\eta_k| - \frac{|\eta_k|+\delta}{a+1} \geq \frac{ar_p - \delta}{a+1} > 0$. Hence, eventually after $\left\lceil \frac{(a+1)(r_t - r_p)}{ar_p - \delta} \right\rceil$ steps, the smallest I-state will violate the privacy stipulation and put the panda in danger.

The same conclusion is reached for the case of (ii), when $r_t < (a+1)r_p - \delta$, along similar lines. □

Let the domain of δ described by condition (i) in Lemma 3.6 be $K_1 = (ar_t - r_t, ar_p) \cap (ar_t - r_p, +\infty)$, and that of condition (ii) be $K_2 = (ar_t - r_t, ar_p) \cap (-\infty, (a+1)r_p - r_t)$. If $(a+1)r_t < (a+2)r_p$, then $ar_t - r_p < (a+1)r_p - r_t$. We have $K_1 \cup K_2 = (ar_t - r_t, ar_p)$, so condition (i) and (ii) together describe problems making up all the gaps in Figure 3.2, save for the last and penultimate one. The previous lemma shows that there are no privacy-preserving strategies for those gaps.

Notice that Lemma 3.3 and 3.5 describe under-constrained problems where there exists a straightforward privacy-preserving tracking strategy. Lemma 3.4 and 3.6 prove that there are no privacy-preserving tracking strategies for a set of problems which we might consider to be over-constrained. Next, we give a lemma that describes boundary problems separating under-constrained from over-constrained problems. (Definition 3.5 in Section 3.3.3 formalizes these boundary problems, for here it is sufficient to recognize that these are elements from the penultimate gap in Figure 3.2.)

Lemma 3.7. For any 1-dim. panda tracking problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$, if $a \in \mathbb{Z}^+$, $a \leq c$ and $r_p \leq ar_t - \delta < (a+1)r_p - \delta \leq r_t$, then there is no privacy-preserving strategy for every initial I-state.

Proof. For $|\eta_0| \in (ar_t - \delta, (a+1)r_p - \delta)$, if we take action $\oplus$ and equally divide the prior I-state into a parts, the size of the largest posterior I-state at the next time-step ($t = 1$) is $|\eta_1| = \frac{|\eta_1^-|+\delta}{a} > \frac{ar_t-\delta+\delta}{a} = r_t$, which will violate the tracking stipulation. If we take action $\ominus$ and equally divide the prior I-state into $a+1$ parts, the size of posterior I-state at the next step ($t = 1$) is $|\eta_1| = \frac{|\eta_1^-|+\delta}{a+1} < \frac{(a+1)r_p-\delta}{a+1} = r_p$, which will violate the privacy stipulation. Hence, there are no strategies, which are both **PP** and **TT**, for $|\eta_0| \in (ar_t - \delta, (a+1)r_p - \delta)$. □

The preceding proof shows that, in the case of these problems, there are no strategies which are both **PP** and **TT** for all initial I-states. It achieves this by showing that there are particular initial I-states for which either the PPC or the TTC must be violated. But the problems in Lemma 3.7 are not merely over-constrained instances—something rather more complex is going on in these boundary problems. To illustrate this fact, next, we show that there exist boundary problems that have a **PP** and **TT** strategy but only for certain initial I-states.

Example 3.1. Consider $Q_1 = (\eta_0, r_p = 76, r_t = 101.3, \delta = 227, c = 4)$. It satisfies the constraints in Lemma 3.7 (viz., $r_p \leq ar_t - \delta < (a+1)r_p - \delta \leq r_t$ and $a \leq c$, since $a = 3$) and also has privacy-preserving strategies described by

$$\pi_{Q_1} = (\oplus(\ominus)^3)^*$$

for $|\eta_0| \in [76, 76.9]$. The asterisk in the expression for π_{Q_1} above should be interpreted as a Kleene star. By removing the appropriate prefix of actions, π_{Q_1} can also be extended to work for any $|\eta_0| \in [76, 76.9] \cup [77, 80.6] \cup [81, 95.4] \cup [97, 101.3]$.

This problem instance and the behavior of the strategy is illustrated in Figure 3.4.

Example 3.2. We can adapt Q_1 from Example 3.1 by considering a slightly more lethargic panda, which has $\delta = 223$. This gives a new problem instance $Q_1' = (\eta_0, r_p = 76, r_t = 101.3, \delta = 223, c = 4)$, which has more complex privacy-preserving strategies, again written in regular expression-like form

$$\pi_{Q_1'} = (\oplus \ominus \oplus(\ominus)^2)^*$$

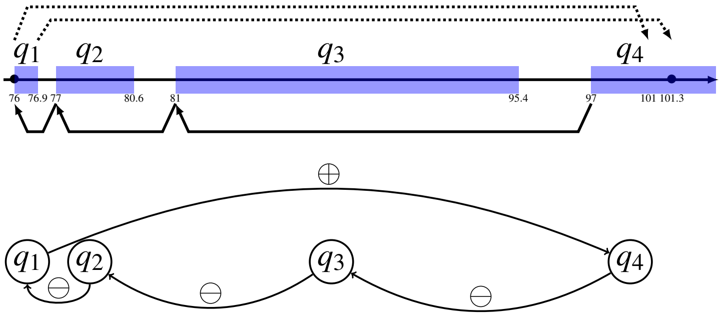


Figure 3.4: The problem instance given in Example 3.1, where there exists a **PP** and **TT** tracking strategy for $|\eta_0| \in q_1 \cup q_2 \cup q_3 \cup q_4$.

for $|\eta_0| \in [76, 79.21]$. As with the case above, after dropping an appropriate prefix of actions, $\pi_{Q'_1}$ will work on initial states: $|\eta_0| \in [76, 79.21] \cup [80, 80.9] \cup [81, 93.86] \cup [96.66, 100.6] \cup [101, 101.3]$.

And this problem instance and how the strategy $\pi_{Q'_1}$ relates is illustrated in Figure 3.5.

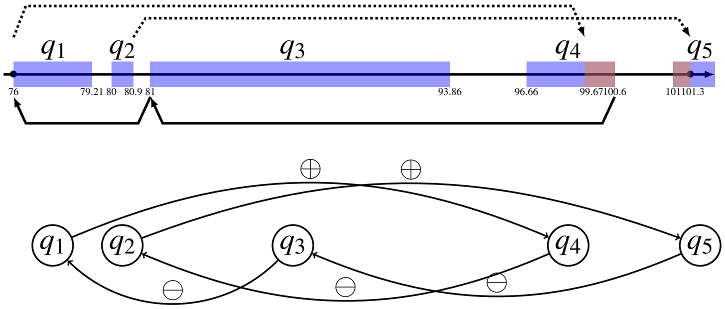


Figure 3.5: The problem instance given in Example 3.2, where there exists a **PP** and **TT** tracking strategy for $|\eta_0| \in q_1 \cup q_2 \cup q_3 \cup q_4 \cup q_5$.

Both Figures 3.4 and 3.5 are instances of problems whose feasibility depends on the size of the initial I-state, $|\eta_0|$. They illustrate that, unlike the under-constrained and over-constrained problems, any conclusions reached about such instances demand a deeper analysis—the details of that analysis are deferred till Section 3.3.3. First, we collect the results presented thus far, clarifying their interplay and giving some interpretation.

3.3 Examing one-dimensional privacy-preserving tracking from design perspective

In this section, we will examine the solvability of the one-dimensional privacy-preserving tracking problems from robot design perspective. We show that as we increase the sensing power by adding more preimages, there will always be problems that cannot be solved. We also identify that there exist problems whose sovability depends on the robot's initial belief.

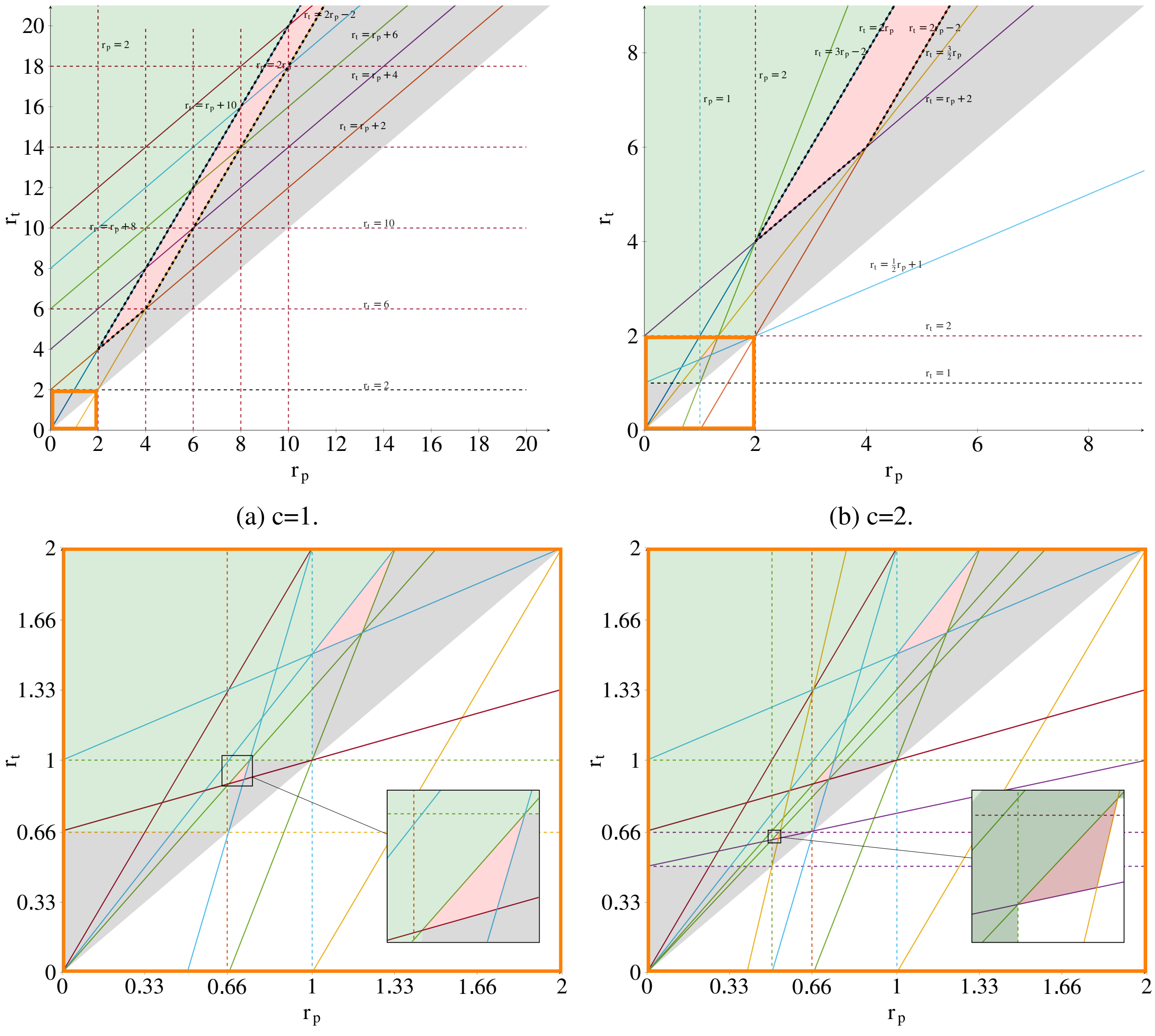
with respect to the robot's sensing power and initial belief, and show that the

3.3.1 Impossibility results of solvability and asymptotic sensing power

To have a clearer sense of how these pieces fit together, we have found it helpful to plot the space of problem parameters and examine how the preceding theorems relate visually. Figure 3.6 contains subfigures for increasingly powerful robots (in terms of sensing) with $c = 1, 2, 3, 4$. The white regions represent the trivial $r_p > r_t$ instances; otherwise the whole strategy space is categorized into the following subregions: under-constrained space (colored green), over-constrained space (colored gray), and boundary space (colored pink). When summarized in this way, the results permit examination of how sensor power affects the existence of solutions.

Preserving the privacy of a target certainly makes some unusual demands of a sensor. O'Kane's quadrant sensor has preimages for each output class that are infinite subsets of the plane, making it possible for his robot to increase its uncertainty if it must do so. But it remains far from clear how one might tackle the same problem with a different sensor. The privacy requirement makes it difficult to reason about the relationship between two similar sensors. For example, an octant sensor appears to be at least as useful as the quadrant sensor, but it makes preserving privacy rather trickier. Since octants meet at the origin at 45°, it is difficult to position the robot so that it does not discover too much. One advantage of the one-dimensional model is that the parameter c allows for a natural modification of sensor capabilities. This leads to three closely related results, each of which helps clarify how certain limitations persist even when c is increased.

Theorem 3.2. (More sensing won't grant omnipotence) The one-dimensional robot is not always able to achieve privacy-preserving tracking, regardless of its sensing power.



(a) c=1.

(b) c=2.

(c) c=3.

(d) c=4.

Figure 3.6: The problem parameter space and the existence of strategies for robots with differing c. The green region depicts under-constrained problems, where a suitable strategy exists no matter the initial I-state. The gray region represents the conditions that are over-constrained. The pink region depicts conditions that serve as the boundary between over-constrained and under-constrained problems. The white region represents trivially infeasible problems. The orange rectangles emphasize the different levels of magnification and highlight conditions where the differences in sensing power come into play. Both r_p and r_t are expressed in units of $\frac{\delta}{2}$.

Proof. The negative results in the over-constrained problems described by Lemmas 3.4 and 3.6 show that there are circumstances where it is impossible to find a tracking strategy satisfying both *PPC* and *TTC*. Though these regions depend on c, no finite value of c causes these regions to be empty. □

Turning to the boundary cases that serve as the transition from over-constrained problems to under-constrained ones, one might think that these boundary regions will shift or reduce when the sensor gets more powerful to handle the constraints. But this explanation is actually erroneous. Observe that the boundary region is more complicated than other regions within the strategy space: in Figure 3.6, the green region is contiguous, whereas the regions marked pink are not. The specific boundary region for Lemma 3.7 under condition $a = 1$, visible clearly as chisel shape in Figures 3.6a and 3.6b, is invariant with respect to c, so remains as the boundary in all circumstances (though outside the visible region in Figures 3.6c and 3.6d, it is present). As c increases, what happens is that the former boundary regions still serve as the boundary, and the regions previously marked as over-constrained are claimed as under-constrained, or become the boundary regions. It is evident that the boundary regions do not shift with more powerful sensors. The following expresses the fact that additional sensing power fails to reduce the number of the boundary regions.

Theorem 3.3. (Boundary region invariance) The number of boundary regions does not decrease by using more powerful sensors.

Proof. We focus on the boundary areas within the square between $(0, 0)$ and $(2, 2)$ as the parts outside this (orange) square do not change as c increases. According to Lemma 3.7, the boundary regions for any specific a are bounded by the following linear inequalities:

$$\begin{aligned} r_t &< \frac{2}{a-1}, && (1)\\ r_p &> \frac{2}{a}, && (2)\\ r_t &\geq \frac{r_p}{a} + \frac{2}{a}, && (3)\\ r_t &\geq (a+1)r_p - 2, && (4)\\ r_t &< \frac{(a+1)}{a} r_p. && (5) \end{aligned}$$

Combining (1)–(3) gives both the bound for r_t as $r_t \in \left[\frac{2(a+1)}{a^2}, \frac{2}{a-1}\right)$, and the bound for r_p as $r_p \in \left[\frac{2}{a}, \frac{2a}{a^2-1}\right]$. The boundary region, thus, is bounded.

Next, we show that (1) and (2) are dominated by (3)–(5). According to (4) and (5), we have $r_p < \frac{2a}{a^2-1}$. Applying this result to (5) produces (1). Similarly, combining (3) and (5) together yields (2). Hence, the boundary regions are fully determined by inequalities (3)–(5). (Figure 3.7 provides a visual example.)

To form a bounded region with three linear inequalities, the boundary region has to be a triangle. The three points of the triangle can be obtained by intersecting pairs of (3)–(5): $\left(\frac{2}{a}, \frac{2a+2}{a^2}\right)$, $\left(\frac{2a}{a^2-1}, \frac{2}{a-1}\right)$, $\left(\frac{2(a+1)}{a^2+a-1}, \frac{2(a+1)^2}{a^2+a-1} - 2\right)$. Since $a \in \{2, 3, \cdots, c\}$, the triangle region will not be empty. Let $\Delta(a)$ denote the triangle with parameter a. Then the smallest y coordinate for $\Delta(a)$ is $\text{minY}(\Delta(a)) = \frac{2a+2}{a^2}$. And the largest y coordinate for $\Delta(a)$ is $\text{maxY}(\Delta(a)) = \frac{2}{a-1}$. For adjacent triangles $\Delta(a)$ and $\Delta(a+1)$, we have $\text{minY}(\Delta(a)) > \text{maxY}(\Delta(a+1))$. Hence, the triangles for different values of a do not overlap. □

The preceding discussion showed that the number of boundary regions increases with c and that, along with the chisel shaped region, there are $c-1$ triangles of decreasing size. This motivates our introduction of a quantitative measure of the robot's power as a function of c.

Definition 3.4. A measure of tracking power, $p(c)$, for the robot with c sensing parameters should satisfy the following two properties: $p(c) > 0, \forall c \in \mathbb{Z}^+$ *(positivity)*, and $p(a) > p(b)$, if $a, b \in \mathbb{Z}^+$ and $a > b$ *(monotonicity)*.

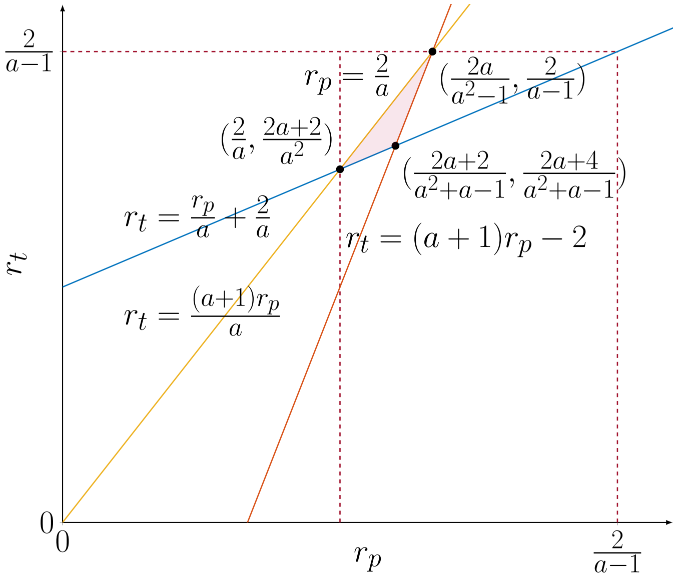


Figure 3.7: Relationships between the linear inequalities in Lemma 3.7.

The plots in Figure 3.6 suggest that one way to quantify change in these regions is to measure changes in the various areas of the parameter space as c increases. As a specific measure of power in the one-dimensional setting, we might consider the proportion of cases (in the r_p vs. r_t plane) that are under-constrained (green) and boundary regions (pink). Though the green and pink areas are unbounded in the full plane, the only changes that occur as c increases are in the square between $(0, 0)$ and $(2, 2)$. Thus, we take $p(c)$ to equal to the total volume of green and pink regions filling within the region $0 \leq r_p \leq r_t$ and $0 \leq r_t \leq 2$. This area satisfies the properties in Definition 3.4 and is indicative of the power of the robot as, intuitively, it can be interpreted as an upper-bound of the solvable cases.

Corollary 3.1. (Asymptotic tracking power) The power $p(c)$ of a robot with c sensing parameters to achieve privacy-preserving tracking in the 1-dim. problem is bounded and $\lim\limits_{c \to \infty} p(c) = \ell$, with $1.5 < \ell < 1.6$.

Proof. Inequalities (3)–(5) in the proof of Theorem 3.3 give $c - 1$ triangles, one for each $a \in \{2, 3, \cdots, c\}$. An analytic expression gives the area of each of these triangles and the series describing the cumulative pink volume $p_{\text{pink}}(c)$ within $0 \leq r_p \leq r_t$ and $0 \leq r_t \leq 2$ can be shown (by the comparison test) to converge as $c \to \infty$. Similarly, the cumulative green volume $p_{\text{green}}(c)$ within $0 \leq r_p \leq r_t$ and $0 \leq r_t \leq 2$ converges. Numerical evaluation gives the value of the limit $\approx 1.54\bar{5}$. □

3.3.2 The greatest tracking precision for a given problem instance

A practical question what one might ask is: for a given set of panda tracking parameters (the panda's motion δ, privacy bound r_p, sensor capability c), what is the highest precision (i.e., tightest tracking bound), for which the problem is solvable? Formally this requires us to find the smallest r_t, which we denote $r_t^\star$, such that an **PP** and **TT** tracking strategy exists $\forall \eta_0, P_1 = (\eta_0, r_p, r_t^\star, \delta, c)$. For small values of c, the lower envelope of the green region in Figure 3.6 shows the values of $r_t^\star$ graphically. By tracing the positions of the under-constrained problems, over-constrained ones, and boundary regions, an answer to this question can be obtained more generally with the following formula (where we have let $c^\star = \lceil \frac{\delta}{r_p} \rceil$):

$$r_t^\star = \begin{cases} 2r_p, & r_p \geq \delta, \\ \delta, & c = 1, r_p < \delta, \\ \dfrac{\delta}{c}, & 1 < c < c^\star, r_p < \delta, \\ \dfrac{(c^\star + 1) r_p}{c^\star}, & c \geq c^\star, r_p \in [\dfrac{\delta}{c^\star}, \dfrac{\delta c^\star}{c^{\star 2} - 1}], r_p < \delta, \\ \dfrac{\delta}{c^\star - 1}, & c \geq c^\star, r_p \in (\dfrac{\delta c^\star}{c^{\star 2} - 1}, \dfrac{\delta}{c^\star - 1}), r_p < \delta. \end{cases}$$

The conditions on the right-hand side are exhaustive.

3.3.3 Boundary problems show a dependence on the robot's initial I-state

The results of the preceding section provide an analysis for the green and gray areas in Figure 3.6, which correspond to the teeth, gaps, filling, and final interval in the roadmap diagram (Figure 3.2). Only the penultimate interval, resulting in pink regions in Figure 3.6, is left. In this section we analyze these last remaining problem instances, that is the boundary problems, and they are shown to have a different structure from the others (which is why we have opted to treat them separately). We show that there are no **PP** and **TT** tracking strategies for all initial I-states in these instances but, as will be uncovered, there do exist some boundary problems that have **PP** and **TT** tracking strategies for some initial I-states.

First, we describe the common characteristics of boundary problems, introducing the notion of an "impossibility zone", thereafter we give a detailed treatment of the each of the classes of problems that arise (Lemmas 3.9–3.12), The last subsection dealing with boundary problems provides some broader interpretation of the technical results.

3.3.3.1 The boundary problems in detail

The boundary problem described is defined as follows:

Definition 3.5. The 1-dim. panda tracking problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$ is a *boundary problem*, if $\delta \in (ar_t - r_t, ar_p)$, $r_p \leq ar_t - \delta < (a+1)r_p - \delta \leq r_t$, and $a \in \{1, 2, \ldots, c\}$.

In boundary problem P_1, if the size of the I-state is $ar_t - \delta$, then it will transit to the I-state with size r_t under action $\ominus$. The I-state with size $(a+1)r_p - \delta$ will transit to the I-state with size r_p. Between $ar_t - \delta$ and $(a+1)r_p - \delta$, there is a zone $I_0^\times = (ar_t - \delta, (a+1)r_p - \delta)$ shown in Figure 3.8, wherein all actions will violate the privacy or tracking bound in the next time-step. We call $I_0^\times$ the *impossibility zone*. To its left, $\oplus$ is the only action that can be taken, otherwise the privacy constraint will be violated in the next time-step. Similarly, $\ominus$ is the only action that can be taken to the right of the impossibility zone. The robot's actions are forced, and making any other choice means that the privacy or tracking constraints will be violated immediately afterward. Thus we may conclude that if there is a **PP** and **TT** strategy for some η_0, then the privacy-preserving tracking strategy is unique.

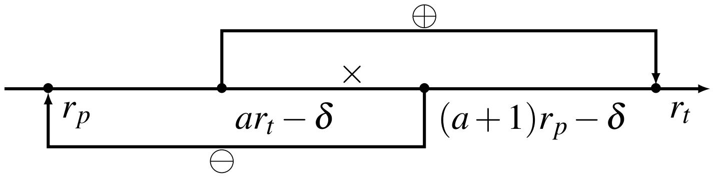


Figure 3.8: The boundary problem instances have the property that there is a central zone from which no action can safely be taken thereafter.

Next, we will examine the I-state transitions under action $\oplus$ and $\ominus$, so as to understand the structure of the privacy-preserving tracking strategy in the boundary problem. Let the resulting size of I-state after taking action $\oplus$ and $\ominus$ be $f_+(x)$ and $f_-(x)$. For a perturbation Δ_x, the two functions satisfy the following:

1. If $\Delta_x > 0$, then $f_+(x + \Delta_x) - f_+(x) = \frac{\Delta_x}{a} > 0$.

2. If $\Delta_x > 0$, then $f_-(x + \Delta_x) - f_-(x) = \frac{\Delta_x}{a+1} > 0$.

These properties help in understanding the transition of I-states under action $\oplus$ and $\ominus$. Both f_+ and f_- are monotonically increasing functions, preserving the order of their inputs. If the same action is performed at the both endpoints of some interval $[x, x + \Delta_x]$, then the interval transits to $[f_+(x), f_+(x + \Delta_x)]$ under action $\oplus$, and $[f_-(x), f_-(x + \Delta_x)]$ under action $\ominus$. Extending the notation naturally, we will write the interval $[f_+(x), f_+(x + \Delta_x)]$ as $f_+([x, x + \Delta_x])$. The size of the new interval will decrease to $\frac{1}{a}$ of the original interval under action $\oplus$, and $\frac{1}{a+1}$ of the original one under action $\ominus$. That is, $|f_+([x, x + \Delta_x])| = \frac{\Delta_x}{a}$ and $|f_-([x, x + \Delta_x])| = \frac{\Delta_x}{a+1}$.

Following these transition properties, we are able to divide $[r_p, r_t]$ into subintervals. Let p be the maximum number of sequential $\oplus$'s that can be performed before violating the tracking constraint r_t for all I-states, and m be the maximum number of sequential $\ominus$'s that before violating the privacy constraint r_p for all I-states. Then, as shown in Figure 3.9, the interval $[r_p, r_t]$ can be divided into three parts: $\bigcup\limits_{1 \le j \le p} I_j^+$, the impossibility zone $I_0^\times$, and $\bigcup\limits_{1 \le j \le m} I_j^-$. For each interval $I_j^+ = (l_j^+, r_j^+]$, we have $l_{j+1}^+ = r_j^+$ and $l_j^+ = f_+^{-1}(r_j^+)$, where $f_+^{-1}(x)$ denotes the size of the I-state that transits to a new one of size x after taking action $\oplus$. The "+" in I_j^+ means that only action $\oplus$ can be taken, and j is the number of sequential $\oplus$'s that can be taken before violating the tracking constraints. Similarly, for any interval $I_j^- = [l_j^-, r_j^-]$, we have $r_{j+1}^- = l_j^-$ and $r_j^- = f_-^{-1}(l_j^-)$. The "-" in I_j^- means that only action $\ominus$ can be taken, and there are at most j sequential $\ominus$'s to be taken before violating the privacy constraints.

Following from the I-state transition properties and as Lemma 3.8 states, the maximum number of $\oplus$'s and $\ominus$'s are constrained.

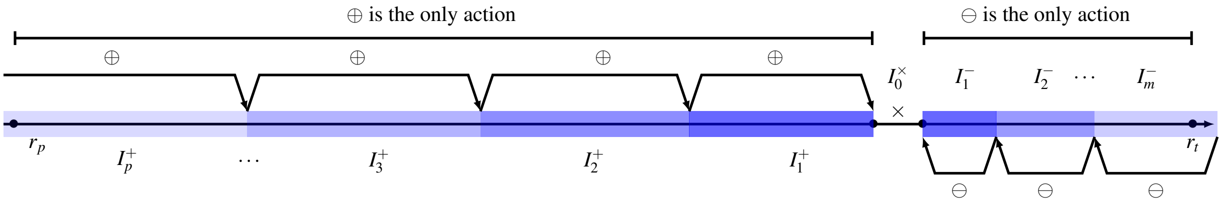


Figure 3.9: m ⊕'s and p ⊖'s can be taken in $[r_p, ar_t - \delta]$ and $[(a+1)r_p - \delta, r_t]$.

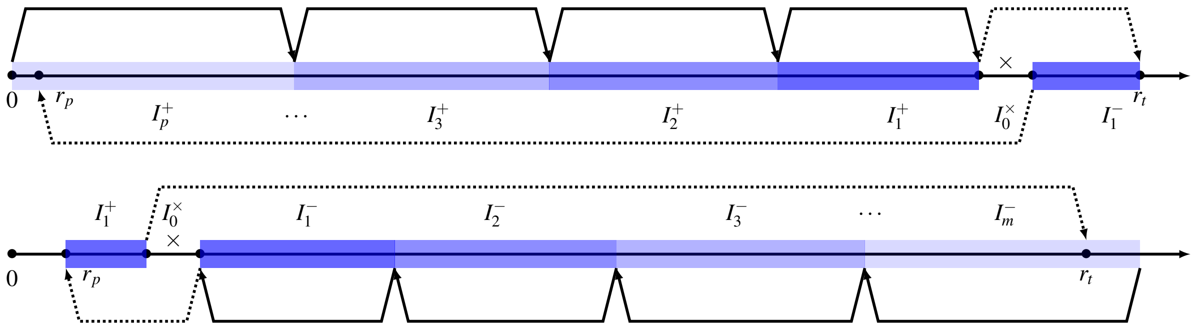


Figure 3.10: All boundary problem instances have either a single ⊖ action (top) or single ⊕ action (bottom).

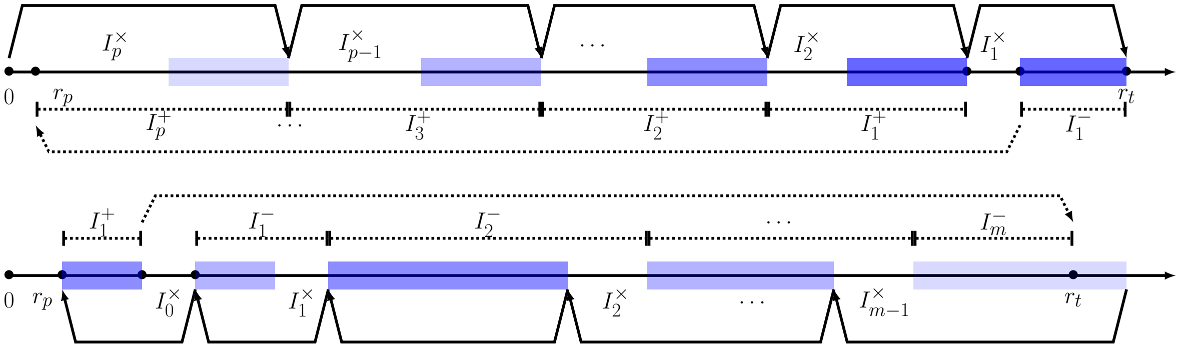


Figure 3.11: Two cases of the boundary problem after the propagating the impossibility zone with $\oplus^*$ or $\ominus^*$. The top instance fits the description in the text.

Lemma 3.8. In the boundary problem P_1, either $m = 1$ or $p = 1$.

Proof. The proof is by contradiction by assuming that both $m > 1$ and $p > 1$. Since interval I_1^+ transits to a new one containing I_1^- under ⊕, we can conclude that $|I_1^+| > |f_+(I_1^+)| > |I_1^-|$. But interval I_1^- transits to a new one containing I_1^+ after action $\ominus$. Hence, $|I_1^-| > |f_-(I_1^-)| > |I_1^+|$, which contradicts the prior assertion. Therefore the assumption is incorrect and either $m = 1$ or $p = 1$. □

There are two cases, shown in Figure 3.10, depending on whether $p = 1$ or $m = 1$. We see that the unique privacy-preserving tracking strategy consists of $\oplus$ and $\ominus$ in the following regular expression form $\oplus^*(\ominus(\oplus)^{p|p-1})^*$ or $\ominus^*(\oplus(\ominus)^{m|m-1})^*$, where the prefixes on the action outside the parentheses have length no more than the same one inside.

In boundary problems, the impossibility zone, $I_0^\times$, contains the I-states that will violate the privacy or tracking bound in one time-step for any action. The next step is to define $I_j^\times$, where $j \in \mathbb{N}\cup\{0\}$, such that $I_j^\times$ represents the set of I-states that will violate the privacy or tracking bound in $j+1$ steps. Then $I_j^\times$ is defined (recursively) as:

$$I_j^\times = \begin{cases} f_+^{-1}(I_{j-1}^\times) \cap [r_p, r_t] & \text{if } f_-^{-1}(I_{j-1}^\times) \cap [r_p, r_t] = \varnothing, \\ f_-^{-1}(I_{j-1}^\times) \cap [r_p, r_t] & \text{if } f_+^{-1}(I_{j-1}^\times) \cap [r_p, r_t] = \varnothing. \end{cases}$$

The two conditions are mutually exclusive because actions are forced for boundary problems: we have either $f_-^{-1}(i_{j-1}^\times) \cap [r_p, r_t] = \varnothing$ or $f_+^{-1}(i_{j-1}^\times) \cap [r_p, r_t] = \varnothing$, since at least one action is illegal, taking $I_{j-1}^\times$ out of $[r_p, r_t]$. But the two conditions need not be jointly exhaustive and, if neither condition holds for some j, then no such $I_j^\times$ is defined.

For every $I_j^\times$ we are justified in calling it an interval, for it is a single interval, and $\exists k \in \{1, 2, \dots\}$, such that $I_j^\times \subseteq I_k^+$ or I_k^-. Both of these properties are easily shown to hold inductively (though the $m = 1$ and $p = 1$ cases, see Figure 3.10, each demand a slightly different inductive basis).

To find all the impossibility zones, we need to trace through the I-state transitions until no new impossibility zones are produced, i.e., until neither condition holds. Notice that the definition is in terms of the inverse maps of f_+ and f_- so, instead of following the I-state transition forwards, we must proceed backwards to identify the new impossibility zone $I_{j+1}^\times$ that transits to $I_j^\times$. We call this process of tracing inverses back-propagation. In the first case ($m = 1$), the propagation advances backward using the inverse of $\oplus^*(\ominus(\oplus)^{p|p-1})^*$, and for ($p = 1$), the second case, by using the inverse of $\ominus^*(\oplus(\ominus)^{m|m-1})^*$. Since the two cases are symmetric, from this point onwards we examine the first only, though all properties also hold for the second and can be derived in an analogous manner.

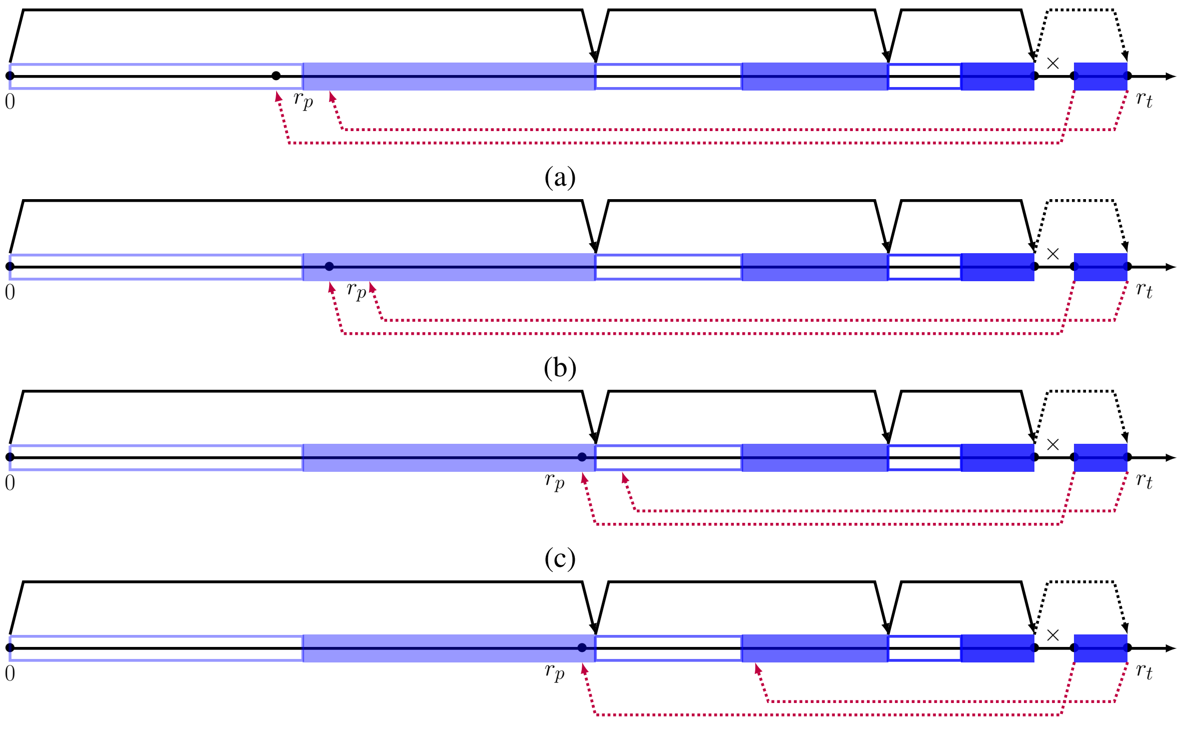
The back-propagation of impossibility zones is periodic and each period can be divided into two phases. The first phase is the back-propagation from I_1^- to I_p^+ or I_{p-1}^+ under the inverse of action sequence $\oplus^*$. The second is the back-propagation from I_p^+ or I_{p-1}^+ to I_1^- following the inverse of action $\ominus$. The first phase is straightforward, since the fraction of the impossibility zone in I_j^+ during the propagation is the same as that of $I_1^- \cup I_0^\times$ in the same period. A visual example of back-propagation in the first phase of the first period appears in Figure 3.11. Depending on the size of interval $[r_p, r_t]$, there are either p or $p+1$ impossibility zones. The second phase is important, since it determines the fraction in I_1^- in the next period. To satisfy the constraint that $|f_-(I_1^-)| < |I_1^-| < |I_{p-1}^+|$, there are four possible transitions as shown in Figure 3.12. Each transition maps $f_-(I_1^-)$ into different parts of $I_p^+ \cup I_{p-1}^+$, which results in a different fraction in the next period. As one continues to follow the periodic transition there are two outcomes: (i) if the fraction of the impossibility zone in I_1^- converges to a value less than 1, then we have found all the impossibility zones and the I-states that are not in the impossibility zone have a **PP** and **TT** strategy; (ii) if the impossibility zone fills I_1^-, then the impossibility zones taint the whole $[r_p, r_t]$ and no regions with privacy-preservable tracking strategies remain.

3.3.3.2 The impossibility zone's propagation

To track of the fraction of the impossibility zone in I_1^- at each period, we denote the fraction at period t as z_t. In the first period, there is no impossibility zone in I_1^- and $z_1 = 0$. In Lemmas 3.9–3.12, we will compute z_t for each possible critical transition.

Lemma 3.9. For the boundary problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$, if $r_p \in I_p^\times$, then there are no **PP** and **TT** strategies for any $\eta_0 \in [r_p, r_t]$.

Proof. The impossibility zone is proved to ultimately taint the entire interval $[r_p, r_t]$. If $r_p \in I_p^\times$, then $f_-^{-1}(I_1^-)$ is mapped into two parts: the impossibility part $B_1 = f_-^{-1}(I_1^-) \cap I_p^\times$ and the non-



(a)

(b)

(c)

(d)

Figure 3.12: Four possibilities of the I-state transition for I_1^- in the first case of the boundary problem. These correspond to Lemmas 3.9–3.12.

impossibility part $f_-^{-1}(I_1^-) \cap (I_p^+ \setminus B_1)$. Following this mapping, the fraction of the impossibility zone $I_{p+1}^\times$ in I_1^- in the next period, $t = 2$, is $z_2 = \frac{(a+1)|B_1|}{|I_1^-|}$. In the first phase of period $t = 2$, the impossibility zone $I_{p+1}^\times$ will be back-propagated to I_p^+ as $I_{2p+1}^\times$, which taints part of the non-impossibility part of the previous period and gets mapped to I_1^- in the next period. There is a relationship between the sizes: $|I_{2p+1}^\times| = a^p(a+1)|B_1|$. Let B_2 be the additional impossibility interval that will be back-propagated in the next period. But B_1 and B_2 are contiguous, since B_2 is the impossibility zone to the left of I_p^+. Then $z_3 = (a+1)\frac{|B_1|+|B_2|}{|I_1^-|}$, where $|B_2| = a^p(a+1)|B_1|$. Following this back-propagation pattern in general, at period t, the additional impossibility zone to be back-propagated to I_1^- is B_{t-1}, where $|B_{t-1}| = a^p(a+1)|B_{t-2}|$ and, again, B_{t-1} is contiguous with B_{t-2}. Therefore, after some (finite) time κ, z_κ will reach 1, and the impossibility zone will have tainted the whole interval $[r_p, r_t]$. □

To sum up visually, Lemma 3.9 shows that there are no **PP** and **TT** strategies in the boundary problem described by Figure 3.12a.

In the following I-state transitions, there are now only $p-1$ impossibility zones that can be found during the first phase of the first period. Let these impossibility zones be $L = \bigcup_{0 \leq j \leq p-1} I_j^{\times}$.

Lemma 3.10. For the boundary problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$, if $r_p \in I_p^+$ and $f_-(r_t) \in I_p^+$, then P_1 is I-state dependent: there are **PP** and **TT** strategies for $|\eta_0| \in (I_p^+ \setminus L)$, while there are no such strategies for $|\eta_0| \in L$.

Proof. The fraction of the impossibility zone within I_1^- reaches a maximum fraction less than 1, and a **PP** and **TT** strategy for $|\eta_0| \in ([r_p, r_t] \setminus L)$ is given in this proof. If $r_p \in I_p^+$ and $f_-(r_t) \in I_p^+$, then $f_-(I_1^-)$ is mapped into the $[r_p, r_t] \setminus L$, and no impossibility will back-propagate to I_1^- at period $t = 2$ or afterward. Hence, the fraction of the impossibility zone in I_1^- remains zero and, therefore, there is no **PP** and **TT** strategy for L, but a privacy-preserving tracking strategy for $[r_p, r_t] \setminus L$ exists. The privacy-preserving tracking strategy for $[r_p, r_t] \setminus L$ is as follows: take action $\oplus$ in $[r_p, r_t] \setminus (L \cup I_1^-)$, and action $\ominus$ in I_1^-. The resulting I-state remains within $[r_p, r_t] \setminus L$ forever. □

Lemma 3.10 says that there are **PP** and **TT** strategies in the boundary problem described by Figure 3.12b.

Lemma 3.11. For the boundary problem $P_1 = (\eta_0, r_p, r_t, \delta, c)$, there are no **PP** and **TT** strategies for any $\eta_0 \in [r_p, r_t]$ when $r_p \in (I_p^+ \setminus L)$ and $f_-(r_t) \in I_{p-1}^{\times}$.

Proof. The proof is similar to Lemma 3.9. □

Lemma 3.11 says that there are no **PP** and **TT** strategies in the boundary problem described by Figure 3.12c.

For the fourth type of I-state transition, since $r_p \in I_p^+$ and $f_-(r_t) \in (I_{p-1}^+ \setminus I_{p-1}^{\times})$, it follows that $f_-(I_1^-)$ is mapped to three parts: the first non-impossibility zone $f_-(I_1^-) \cap (I_p^+ \setminus L)$, impossibility zone $I_p^{\times}$, and the second non-impossibility zone $f_-(I_1^-) \cap (I_{p-1}^+ \setminus L)$. Following the forward I-state transition, part of first one non-impossibility zone will transit to part of the second one and vice versa. It turns out that the fraction of impossibility zones in I_1^- stays the same, remaining less than

1, when the size of the first and the second non-impossibility zone are comparable in size. And there is, thus, a **PP** and **TT** strategy for some initial I-states in this condition. Otherwise, there is no privacy-preserving tracking strategy for all of $[r_p, r_t]$. This conclusion is reached by first defining $v = f_-(I_1^-) \cap I_p^+$ as the first non-impossibility part, and $w = f_-(I_1^-) \cap I_{p-1}^+$ as the second non-impossibility part. Details are in the following proof for Lemma 3.12.

Lemma 3.12. For the boundary problem P_1 with $r_p \in I_p^+$ and $f_-(r_t) \in (I_{p-1}^+ \setminus I_{p-1}^\times)$:

1. If
$$\frac{1}{a^p(a+1)} \leq \frac{|w|}{|v|} \leq a^{p-1}(a+1),$$
then P_1 is initial I-state dependent: there is a **PP** and **TT** strategy for the initial I-states $\eta_0 \in [r_p, r_t] \setminus L$, while there are no **PP** and **TT** strategies for $I_j^\times \in L$.

2. If either $\frac{|w|}{|v|} > a^{p-1}(a+1)$ or $\frac{|w|}{|v|} < \frac{1}{a^p(a+1)}$, then there are no **PP** and **TT** strategies for $|\eta_0| \in [r_p, r_t]$.

Proof. First we prove that if $\frac{|w|}{|v|} > a^{p-1}(a+1)$, the impossibility zones propagate until they taint all of $[r_p, r_t]$. By tracking I-state transitions the impossibility zone in the middle, $I_{p-1}^\times$, is back-propagated to I_1^- as $I_p^\times$ in one period. In period $t = 2$, the tainted fraction in I_1^- is $z_2 = \frac{(a+1)|I_{p-1}^\times|}{|I_1^-|}$. In the first phase of this period, the impossibility region will propagate to I_{p-1}^+ so that I_{p-1}^+ will be partitioned into three parts with the ratio: $|v| : |I_{p-1}^\times| : |w|$, where the middle part is the new impossibility zone $I_{2p-1}^\times$. This ratio is precisely the same as the partitioning from period $t = 1$. Since $\frac{|w|}{|v|} > a^{p-1}(a+1)$ and $|I_{2p-1}^\times| = (a+1)a^{p-1}$, we have $I_{2p-1}^\times \cap w \neq \varnothing$. That is, part of w will be tainted with impossibility, which will then back-propagate to I_1^- at period $t = 3$. Let $B_1 = I_{2p-1}^\times \cap w$ be that which will back-propagate to I_1^- in the next period. Then $z_3 = (a+1)\frac{|B_1|+|I_{p-1}^\times|}{|I_1^-|}$. In the first phase of period 3, there will be a new impossibility zone B_2 in v to back-propagate toward I_1^- at next period. In addition, $|B_2| = (a+1)a^p|B_1|$. Then $z_4 = (a+1)\frac{|B_2|+|B_1|+|I_{p-1}^\times|}{|I_1^-|}$. In the first phase of period 4, there will be a new impossibility zone B_3 in w, where $|B_3| = (a+1)a^{p-1}|B_2|$. Then $z_5 = (a+1)\frac{|B_3|+|B_2|+|B_1|+|I_{p-1}^\times|}{|I_1^-|}$. Following this pattern, at period t, the additional impossibility zone to be back-propagated to I_1^- is B_{t-2}, where $|B_{t-2}| \geq a^{p-1}(a+1)|B_{t-3}|$. All the impossibility zones in I_1^- form a contiguous interval. Therefore, at some finite κ, the impossibility zone will eventually taint the whole interval $[r_p, r_t]$, and z_κ will reach 1. The case with $\frac{|w|}{|v|} < \frac{1}{a^p(a+1)}$ has a similar proof showing that there are no **PP** and **TT** strategies for P_1.

If $\frac{1}{a^p(a+1)} \leq \frac{|w|}{|v|} \leq a^{p-1}(a+1)$, we then have $I_{2p-1}^\times \cap w = \varnothing$ and $I_{2p-1}^\times$ is the last non-empty impossibility zone. The fraction tainted stays as $z_2 = \frac{(a+1)|I_{p-1}^\times|}{I_1^-} < 1$. Thus, for the initial I-states that do not belong to the impossibility zone, there is a privacy-preserving tracking strategy. Let $L' = \bigcup_{0 \leq j \leq 2p-1} I_j^\times$. The privacy-preserving tracking strategy for $[r_p, r_t] \setminus L'$ is as follows: take action $\oplus$ in $[r_p, r_t] \setminus (L' \cup I_1^-)$, and action $\ominus$ in I_1^-. The resulting I-state will always stay within $[r_p, r_t] \setminus L'$.

□

Lemma 3.12 says that the boundary problem in Figure 3.12d contains both initial I-state dependent cases and also instances which have no privacy-preserving tracking strategy.

3.3.3.3 Interpretation and further observations

The preceding analysis of the boundary problem completes our characterization of all panda tracking problems. The earlier examination of Figure 3.6 can now be supplemented with the observation that the region marked in pink, already (from Lemma 3.7) known not to contain strategies for all initial I-states, actually consists of a mixture of colors. If we imagine an axis with the initial I-state sizes normal to the page, then the pink region is partly gray (Lemmas 3.9 and 3.11) and partly a mixture of green and gray (Lemmas 3.10 and 3.12), with the latter mixture having at least some gray on every line segment departing the page.

This analysis also illuminates some new possibilities for the problem. Under the assumption of a strong poacher who knows all the information obtained by the robot immediately, there is no hope to save the panda from the situations that are not privacy-preservable or target trackable. But if we relax this admittedly very strong model of the adversary, some additional versions of the problem can be salvaged.

Assume that there is some maximum amount of time, τ, that the poacher is willing to spend hunting before giving up. Then the impossibility zones $I_t^{\times}, t \geq \tau$ become safe. Some problems, previously without privacy-preserving strategies, will become I-state dependent, and the I-state dependent problems will have more initial I-states that have privacy-preserving strategies.

Another model to consider is the case where the robot can purposefully forget some limited initial information, say, replacing η_0 with η_0'. Then the poacher can have all the information available, as before, excepting η_0. In this model all the I-state dependent problems can become privacy-preservable and target trackable by simply disguising the impossible initial I-states, which is easily achieved by expanding the I-state until it becomes a privacy-preservable and target trackable one.

3.4 Beyond one-dimensional tracking

The inspiration for this work was the 2-dimensional case. This section lifts the impossibility result to higher dimensions.

3.4.1 Mapping from high dimension to one dimension

In the n-dimensional privacy-preserving tracking problem, the state for the panda becomes a point in $\mathbb{R}^n$. The panda can move with a maximum distance of $\frac{\delta}{2}$ in any direction in $\mathbb{R}^n$ within a single time-step, so that the panda's actions fill an n-dimensional ball. The privacy and tracking bound are also generalized from an interval of size r_p and r_t, to an n-dimensional ball of diameter r_p and r_t respectively. That is, the I-state should contain a ball of diameter r_p and be contained in a ball of diameter r_t, so as to achieve privacy-preserving tracking. The robot inhabits the n-dimensional space as well and attention must be paid to its orientation.

It is unclear what would form the appropriate higher dimensional analogue of parameter c, so we only consider n-dimensional tracking problems for robots equipped with a generalization of the original quadrant sensor. The sensor's orientation is determined by that of the robot and it indicates which of the 2^n possible orthogonal cells the panda might be in. Adopting notation and definitions analogous to those earlier, we use a tuple for n-dimensional tracking problems—a subscript makes the intended dimensionality clear.

The following lemma shows that there is a mapping which preserves the tracking property from n-dimensional problem to 1-dimensional problems.

Lemma 3.13. Given some 1-dim. panda tracking problem $P_1 = (\eta_0, r_p, r_t, \delta, n)$, there exists an n-dim. panda tracking problem $P_n = (\theta_0, r_p, r_t, \delta, 1^n)$ where, if $\mathbf{TT}(P_n)$, then $\mathbf{TT}(P_1)$.

Proof. The approach to this proof has elements of a strategy stealing argument and simulation of one system by another. The robot faced with a 1-dim. problem constructs an n-dim. problem and uses the (hypothetical) strategy for this latter problem to select actions. The crux of the proof is that the 1-dim. robot can report back observations that are apposite for the n-dim. case. Figure 3.13, below, gives a visual overview.

For some $P_1 = (\eta_0, r_p, r_t, \delta, c)$, with $c = n$, we construct $P_n = (\theta_0, r_p, r_t, \delta, 1^n)$ as follows. Without sacrifice of generality, assume that in P_1 the initial I-state $\eta_0 = \{x \mid \eta_0^{\text{min}} \leq x \leq \eta_0^{\text{max}}\}$ is centered at the origin, so $\eta_0^{\text{min}} = -\eta_0^{\text{max}}$. (This simplifies the argument and a suitable translation of coordinate system rectifies the situation otherwise.) Then we choose θ_0 as the closed ball at the origin with radius η_0^{max}.

We show how, given some π_n on $P_n = (\theta_0, r_p, r_t, \delta, 1^n)$, we can use it to define a π_1 for use by the 1-dim. robot. The robot forms θ_0 and also has η_0. It picks an arbitrary unit-length vector $\hat{\mathbf{v}} = v_1\mathbf{e_1} + v_2\mathbf{e_2} + \cdots + v_n\mathbf{e_n}$, unknown to the source of π_n, which is the subspace that the 1-dim. panda lives in. For subsequent steps, the robot maintains $\theta_1^-, \theta_1, \theta_2^-, \theta_2, \ldots, \theta_k^-, \theta_k, \theta_{k+1}^-, \ldots$ along with the I-states in the original 1-dim. problem $\eta_1^-, \eta_1, \eta_2^-, \eta_2, \ldots, \eta_k^-, \eta_k, \eta_{k+1}^- \ldots$. For any step k, the η_k can be seen as measured along $\hat{\mathbf{v}}$ within the higher dimensional space. Given θ_{k-1}, θ_k^- is constructed using Minkowski sum operations as before, though now in higher dimension. Given θ_k^-, strategy π_n determines a new pose for the n-dim. robot and, on the basis of this location and orientation, the n sensing planes slice through θ_k^-. Though the planes demarcate 2^n cells, the line along $\hat{\mathbf{v}}$ is cut into no more than $n+1$ pieces as the line can pierce each plane at most once (with any planes containing $\hat{\mathbf{v}}$ being ignored). Since the 1-dim. robot has $c = n$, it picks the $u_1, \ldots, u_n$ by measuring the locations that the sensing planes intersect the line $\mathbf{x} = \alpha\hat{\mathbf{v}}$, $\alpha \in \mathbb{R}$. (If fewer than n

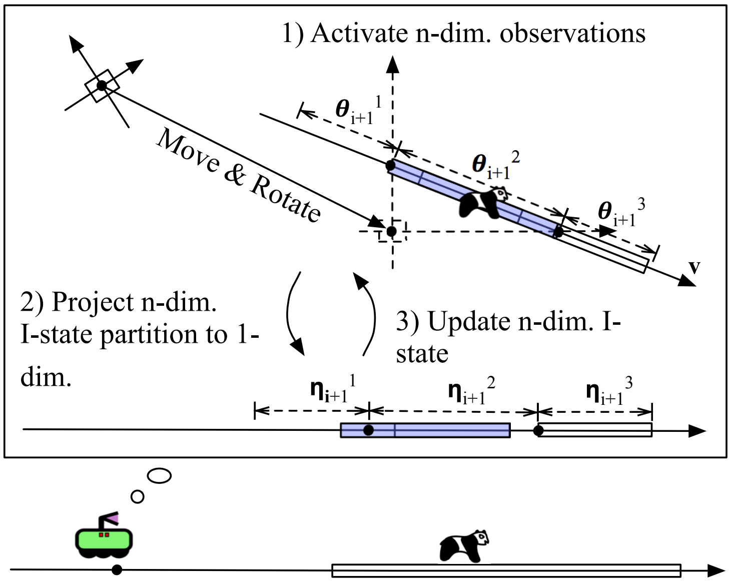


Figure 3.13: Constructing a 1-dim. strategy π_1 from some n-dim. strategy π_n.

intersections occur, owing to planes containing the line, the extra u_i's are simply placed outside the range of the I-state.) After the 1-dim. panda's location is determined, the appropriate orthogonal cell is reported as the n-dim. observation, and θ_k^- leads to θ_k via the intersection operation. This process comprises π_1. It continues indefinitely because θ_k must always entirely contain η_k along the line through $\hat{\mathbf{v}}$ because, after all, a cantankerous n-dim. panda is free to choose to always limit its movements to that line.

If π_n is **TT**, then so too is the resulting strategy π_1 since the transformation relating $\theta_k \cap \{\alpha\hat{\mathbf{v}} : \alpha \in \mathbb{R}\}$ with η_k preserves length and, thus, θ_k fitting within a ball of diameter r_t implies that $|\eta_k| < r_t$. □

3.4.2 Impossibility in high-dimensional privacy-preserving tracking

Now we are ready to connect the pieces together for the main result:

Theorem 3.4. (Impossibility) It is not possible to achieve privacy-preserving panda tracking in n dimensions for every problem with $r_p < r_t$.

Proof. To extend the lemmas that have shown this result for $n = 1$ to cases for $n > 1$, suppose such a solution existed for $P_n = (\theta_0, r_p, r_t, \delta, 1^n)$. Then according to Lemma 3.13, every 1-dim. panda tracking problem $P_1 = (\eta_0, r_p, r_t, \delta, n)$ is **TT**, since they can be mapped to an n-dim. **TT** panda tracking problem. But this contradicts the non-**TT** instances in Lemma 3.4, so no such strategy can exist for every non-trivial problem ($r_p < r_t$) in two, three, and higher dimensions. □

Theorem 3.4 (via Lemma 3.13) relates an n-dim. robot equipped with a generalized quadrant sensor (that is what the 1^n denotes) to a 1-dim. robot with a sensor capability value of n. The n-dim. sensor has 2^n separate output classes (or preimages), yet the 1-dim. sensor has only $n + 1$. It appears, at first sight, that the reduction is to a *less* capable sensor, which seems paradoxical as some information goes missing. But the 1-dim. robot selects the boundaries of the preimages in a far more flexible way than any higher dimensional robot would achieve with actions that move (and reorient) the quadrant sensor's origin.

3.5 Summary

In this chapter, we studied the privacy-preserving tracking problems by reexamining the panda tracking scenario introduced in [1], focusing on how various parameters specifying a problem instance, including the capabilities of the robot and the panda, affect the existence of solutions. Our approach has been to study nontrivial instances of the problem in one dimension. This allows for an analysis of strategies by examining whether the sensing operations involved at each step increase or decrease the degree of uncertainty in a directly quantifiable way. Only if this uncertainty can be precisely controlled forever, can we deem the problem instance solved.

In examining the space of tracking and privacy stipulations, we characterize the solvability of the privacy-preserving problems with respect to robot design, including the robot's initial belief, the panda's movement, and the robot's sensing capability. First, the existence of strategies is shown to be a function of the robot's initial belief and panda's movement. There exist regions without solution, where it is impossible for the robot to actively track the panda as well as protect its privacy for certain nontrivial tracking and privacy bounds. Additionally, we have uncovered regions where solution feasibility is sensitive to the robot's initial belief, which we call I-state dependent cases (or conditions). The simple one-dimensional setting also permits exploration of how circumstances change as the robot's sensing power increases. Perhaps surprisingly, the number of these I-state dependent strategy conditions does not decrease as the robot's sensing becomes more powerful. Finally, we connect the impossibility result back to O'Kane's setting by mapping between high-dimensional and one-dimensional versions, proving that the 2D planar panda tracking problem does not have any privacy-preserving tracking strategy for every non-trivial tracking and privacy stipulation.

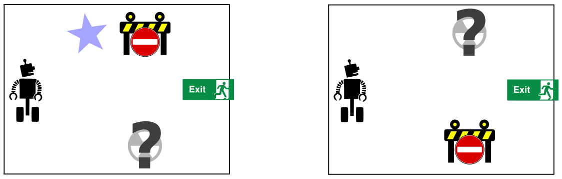
4. FINDING PLANS SUBJECT TO STIPULATIONS ON WHAT THEY DIVULGE[*]

In this chapter, we will generalize the privacy-preserving tracking work in two aspects: First, we consider a robot actively taking actions to interact with the world. The state of the world, which also belongs to the information to be estimated, can be changed by the robot, while in the previous work the target's location cannot be influenced by the robot's tracking strategy. Second, we create a knowledge gap between the robot and the observer by introducing an information disclosure policy to conflate the information perceived by the observer and a hierarchy of prior knowledge for the observer. In doing so, we develop algorithms to jointly search for plans and information disclosure policy for the robot to reach its goal as well as constraining the information learned by an observer.

To illustrate the privacy-preserving planning problem, we create a simple scenario in Figure 4.1 which though simplistic, is rich enough to depict several aspects of the problem. The task requires that a robot determine whether some facility's processing of raw radioactive material meets international treaty requirements or not. The measurement procedure itself depends on the type of facility as the differing physical arrangements of 'pebble bed' and 'breeder' reactors necessitate different actions. First, the robot must actively determine the facility type (checking for the presence of the telltale blue light in the correct spot). Then it can go to a location to make the measurement, with the specific measurement location corresponding with the facility type. But the facility type is deemed sensitive information and the robot must ascertain the radioactivity state while ensuring that the facility type is not disclosed.

What makes this scenario interesting is that the task is rendered infeasible immediately if one prescribes a policy to ensure that the robot never gains sensitive information. Over and above the (classical) question of how to balance information-gathering and progress-making actions, the robot must control what it divulges, strategically increasing uncertainty as needed, precisely lim-

[*] Part of this chapter is reprinted with permission from "Finding Plans Subject to Stipulations on What Information They Divulge" by Yulin Zhang, Dylan Shell, and Jason M. OKane, 2019. *Algorithmic Foundations of Robotics XIII*. 14:106–124. Copyright 2018 by *Springer Nature*.



Pebble bed facility

Breeder reactor

Figure 4.1: **Nuclear Site Inspection** A robot inspects a nuclear facility by taking a measurement at the location marked with a '**?**', the specific position depending on the facility type. But the type of the facility is sensitive information that it must not be divulged to any external observers.

iting and reasoning about the 'knowledge gap' between the external observer and itself. To solve such problems, the robot needs a carefully constructed plan and must establish a policy characterizing what information it divulges, the former achieving the goals set for the robot, the latter respecting all stipulated constraints—and, of course, each depending on the other.

4.1 The model: worlds, robots and observers

Figure 5.2 illustrates the three-way relationships underlying the setting we examine. Most fundamentally, a robot executes a *plan* to achieve some goal in the *world*, and the coupling of these two elements generates a stream of observations and actions. Both the plan and the action–observation stream are disclosed, though potentially only partially, to a third party, we term the *observer*. The observer uses the stream, its knowledge of the plan, and also other known structure to infer properties about the interaction. Additionally, a stipulation is provided specifying particular properties that can be learned by the observer. We formalize these elements in terms of p-graphs and label maps (see [12]).

4.1.1 P-graph and its interaction language

We will start with the definition of p-graphs [12] and related properties:

Definition 4.1 (p-graph)**.** A *p-graph* is an edge-labelled directed bipartite graph with $\mathtt{G} = (V_y \cup V_u, Y, U, V_0)$, where

1) the finite vertex set $V(\mathtt{G}) := V_y \cup V_u$, whose elements are also called *states*, comprises two disjoint subsets: the *observation vertices* V_y and the *action vertices* V_u,

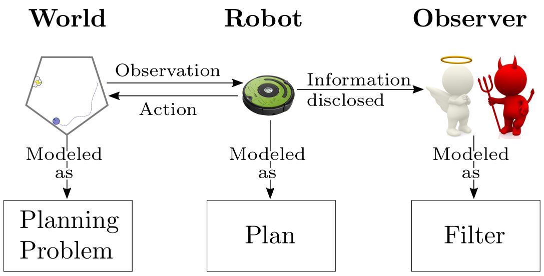


Figure 4.2: An overview of the setting: the robot is modeled abstractly as realizing a plan to achieve some goal in the world and a third party observes, modeled as a filter. All three, the world, plan, and filter have concrete representations as p-graphs.

2) each edge e originating at an observation vertex bears a set of observations $Y(e) \subseteq Y$, containing *observation labels*, and leads to an action vertex,

3) each edge e originating at an action vertex bears a set of actions $U(e) \subseteq U$, containing *action labels*, and leads to an observation vertex, and

4) a non-empty set of states V_0 are designated as *initial states*, which may be either exclusively action states ($V_0 \subseteq V_u$) or exclusively observation states ($V_0 \subseteq V_y$).

An *event* is an action or an observation. Respectively, they make up the sets U and Y, which are called the p-graph's *action space* and *observation space*. We will also have occasion to write $Y(\mathtt{G})$ and $U(\mathtt{G})$ for the observation space and action space of $\mathtt{G}$. Though that is a slight abuse of notation, the initial states will be written $V_0(\mathtt{G})$, similarly.

Intuitively, a p-graph abstractly represents a (potentially non-deterministic) transition system where transitions are either of type "action" or "observation" and these two alternate. The following definitions make this idea precise.

Definition 4.2 (transitions to)**.** For a given p-graph $\mathtt{G}$ and two states $v, w \in V(\mathtt{G})$, a sequence of events $\ell_1, \ldots, \ell_k$ *transitions in* $\mathtt{G}$ *from* v *to* w if there exists a sequence of states $v_1, \ldots, v_{k+1}$, such that $v_1 = v$, $v_{k+1} = w$, and for each $i = 1, \ldots, k$, there exists an edge $v_i \xrightarrow{E_i} v_{i+1}$ for which $\ell_i \in E_i$, and E_i is a subset of $Y(\mathtt{G})$ if v_i is in V_y, or a subset of $U(\mathtt{G})$ if v_i is in V_u.

Concisely, we let the predicate $\textsc{TransTo}(v \xrightarrow{s} w)^{\mathtt{G}}$ hold if there is some way of tracing s

on $\mathtt{G}$ from v to w, i.e., it is True iff v transitions to w under execution s. Note, when $\mathtt{G}$ has non-deterministic transitions, v may transition to multiple vertices under the same execution. We only require that w be one of them.

Definition 4.3 (executions and interaction language)**.** An *execution* on a p-graph $\mathtt{G}$ is a finite sequence of events s, if there exists some $v \in V_0(\mathtt{G})$ and some $w \in V(\mathtt{G})$ for which $\textsc{TransTo}(v \xrightarrow{s} w)^{\mathtt{G}}$. The set of all executions on $\mathtt{G}$ is called the *interaction language* (or, briefly, just *language*) of $\mathtt{G}$ and is written $\mathcal{L}^I(\mathtt{G})$.

Given any edge e, if $U(e) = L_e$ or $Y(e) = L_e$, we speak of e *bearing* the set L_e.

Definition 4.4 (joint-execution)**.** A *joint-execution* on two p-graphs $\mathtt{G}_1$ and $\mathtt{G}_2$ is a sequence of events s that is an execution of both $\mathtt{G}_1$ and $\mathtt{G}_2$, written as $s \in \mathcal{L}^I(\mathtt{G}_1) \cap \mathcal{L}^I(\mathtt{G}_2)$. The p-graph producing all the joint-executions of $\mathtt{G}_1$ and $\mathtt{G}_2$ is their tensor product graph with initial states $V_0(\mathtt{G}_1) \times V_0(\mathtt{G}_2)$, which we denote $\mathtt{G}_1 \otimes \mathtt{G}_1$.

A vertex from $\mathtt{G}_1 \otimes \mathtt{G}_2$ is as a pair (v_1, v_2), where $v_1 \in V(\mathtt{G}_1)$ and $v_2 \in V(\mathtt{G}_2)$. Given a set of vertices $V \subseteq V(\mathtt{G}_1 \otimes \mathtt{G}_2)$, taking the first elements from all the tuples gives a set that we write as $\pi_{\mathtt{G}_1}(V)$. All the second elements is $\pi_{\mathtt{G}_2}(V)$, similarly. Next, the relationship between the executions and vertices is established.

Definition 4.5. The set of vertices reached by execution s in $\mathtt{G}$, denoted $\mathcal{V}_{\mathtt{G}}(s)$, are the vertices to which the execution $s \in \mathcal{L}^I(\mathtt{G})$ transitions, starting at an initial state. Symbolically, $\mathcal{V}_{\mathtt{G}}(s) \coloneqq \{v \in V(\mathtt{G}) \mid \exists v_0 \in V_0(\mathtt{G}), \textsc{TransTo}(v_0 \xrightarrow{s} v)^{\mathtt{G}}\}$. Further, the set of executions reaching vertex v in $\mathtt{G}$ is written as $\mathcal{S}_v^{\mathtt{G}} \coloneqq \{s \in \mathcal{L}^I(\mathtt{G}) \,|\, v \in \mathcal{V}_{\mathtt{G}}(s)\}$.

The mnemonic here being that $\mathcal{V}$ describes sets of vertices, $\mathcal{S}$ describes sets of strings/executions. The collection of sets $\{\mathcal{S}_{v_0}^{\mathtt{G}}, \mathcal{S}_{v_1}^{\mathtt{G}}, \dots, \mathcal{S}_{v_i}^{\mathtt{G}} \dots\}$ can be used to form an equivalence relation $\underset{\mathtt{G}}{\sim}$ over executions, under which $s_1 \underset{\mathtt{G}}{\sim} s_2$ if and only if $\mathcal{V}_{\mathtt{G}}(s_1) = \mathcal{V}_{\mathtt{G}}(s_2)$. This equivalence relation partitions the executions in $\mathcal{L}^I(\mathtt{G})$ into a set of non-empty equivalence classes: $\mathcal{L}^I(\mathtt{G})/\underset{\mathtt{G}}{\sim} = \{[r_0]_{\mathtt{G}}, [r_1]_{\mathtt{G}}, [r_2]_{\mathtt{G}},$

$\dots\}$, where each equivalence class is $[r_i]_{\texttt{G}} = \{s \in \mathcal{L}^I(\texttt{G}) \mid r_i \underset{\texttt{G}}{\sim} s\}$ and r_i is a representative execution in $[r_i]_{\texttt{G}}$. The intuition is that any two executions that transition to identical sets of vertices are, in an important sense, indistinguishable.

In what follows, we shall consider systems where the vertices of a p-graph constitute the state that is stored, acted upon, and/or represented. In this sense, the vertices are akin to a 'sufficient statistic'.

Definition 4.6 (state-determined)**.** A p-graph $\texttt{G}$ is in a state-determined presentation, or is in state-determined form, if $\forall s \in \mathcal{L}^I(\texttt{G}), |\mathcal{V}_{\texttt{G}}(s)| = 1$.

An algorithm to expand any p-graph $\texttt{G}$ into a state-determined presentation $\text{SDE}(\texttt{G})$ is given as Algorithm 1. The key is to make sure that there is only a single starting state and that the labels on different outgoing edges of the same vertex have empty intersections. The language of p-graphs is not affected by state-determined expansion, i.e., $\mathcal{L}^I(\texttt{G}) = \mathcal{L}^I(\text{SDE}(\texttt{G}))$.

Note that, in the preceding discussion, the covering $\{\mathcal{S}^{\texttt{G}}_{v_0}, \mathcal{S}^{\texttt{G}}_{v_1}, \dots, \mathcal{S}^{\texttt{G}}_{v_i} \dots\}$ turned into a partition when we considered *all* the vertices reached by a string (since $\mathcal{V}_{\texttt{G}}(s_1) = \mathcal{V}_{\texttt{G}}(s_2)$), not just whether a vertex *can* be reached by a string. Any string s covered by both $\mathcal{S}^{\texttt{G}}_{v_i}$ and $\mathcal{S}^{\texttt{G}}_{v_j}$ means that, whatever $\mathcal{V}_{\texttt{G}}(s)$ may be, both $v_i \in \mathcal{V}_{\texttt{G}}(s)$ and $v_j \in \mathcal{V}_{\texttt{G}}(s)$. It is easy to show that $\mathcal{V}_{\texttt{G}}(s)$ is the collection of all those v_k's whose $\mathcal{S}^{\texttt{G}}_{v_k}$ contain s. One may start with vertices and ask about the executions reaching those vertices. (Later, this will be part of how an observer makes inferences about the world.)

Definition 4.7. Given any set of vertices $B \subseteq V(\texttt{G})$ in p-graph $\texttt{G}$, the set of executions that reach exactly (i.e. reach and reach only) B is $\mathbb{S}^{\texttt{G}}_B := (\cap_{v\in B}\mathcal{S}^{\texttt{G}}_v) \setminus \cup_{v\in(V(\texttt{G})\setminus B)}\mathcal{S}^{\texttt{G}}_v$.

Above, the $\cap_{v\in B}\mathcal{S}^{\texttt{G}}_v$ represents the set of executions that reach every vertex in B. By subtracting the ones that also reach the vertices outside B, $\mathbb{S}^{\texttt{G}}_B$ describes the set of executions that reach exactly B. In Figure 4.3, the executions reaching w_3 are represented as $\mathcal{S}^{\texttt{G}}_{w_3} = \{a_1o_1, a_2o_1\}$. But the executions reaching and reaching only $\{w_3\}$ are $\mathbb{S}^{\texttt{G}}_{\{w_3\}} = \{a_1o_1\}$ since a_2o_1 also reaches w_4.

Algorithm 1: SDE($\mathtt{G}$)

```
Build initial vertex $v_0'$ in $\mathtt{G}'$, associate $v_0'$ with all $v_0 \in V_0(\mathtt{G})$: $A(v_0') \leftarrow V_0(\mathtt{G})$
Initialize queue $Q \leftarrow \{v_0'\}$
while $Q$ not empty do
    $s' \leftarrow Q$.pop
    // Refine each label and determine which states each refinement maps to:
    $L \leftarrow$ all outgoing edge labels of $s'$
    $L' \leftarrow$ RefineLabels($L$) // See Alg.1 in [59].
    $d_{Lab}[.] \leftarrow \varnothing$ // Empty a map
    for $l' \in L'$ do
        For every outgoing edges of $s'$, record which states you reach with Representative($l'$) by adding them to
          $d_{Lab}[l']$
    end for// Produce new states as need:
    for $(l_a, V_a) \in d_{Lab}$ do
        $flag \leftarrow$False
        for $t \in V(\mathtt{G}')$ do
            if $V_a = A(t)$ then
                add $s' \xrightarrow{l_a} t$ in $\mathtt{G}'$
                $flag \leftarrow$True
            end if
        end for
        if $flag =$False then
            Create new state $t'$, add $s' \xrightarrow{l_a} t'$ in $\mathtt{G}'$, $A(t') \leftarrow V_a$
        end if
    end for
end while
return $\mathtt{G}'$
```

Specifically, the equivalence class $[r_i]_{\mathtt{G}}$ contains the executions that reach exactly $\mathcal{V}_{\mathtt{G}}(r_i)$, so we have $[r_i]_{\mathtt{G}} = \mathbb{S}^{\mathtt{G}}_{\mathcal{V}_{\mathtt{G}}(r_i)}$.

4.1.2 Planning problems and plans

In the p-graph formalism, planning problems and plans are defined as follows [12].

Definition 4.8 (planning problems and plans)**.** A *planning problem* is a p-graph $\mathtt{W}$ along with a goal region $V_{\text{goal}} \subseteq V(\mathtt{W})$; a *plan* is a p-graph $\mathtt{P}$ equipped with a *termination region* $V_{\text{term}} \subseteq V(\mathtt{P})$.

Planning problem $(\mathtt{W}, V_{\text{goal}})$ is solved by some plan $(\mathtt{P}, V_{\text{term}})$ if the plan always terminates (i.e., reaches V_{term}) and only terminates at a goal. Said with more precision:

Definition 4.9 (solves)**.** A plan $(\mathtt{P}, V_{\text{term}})$ *solves* a planning problem $(\mathtt{W}, V_{\text{goal}})$ if there is some integer which bounds length of all joint-executions, and for each joint-execution and any pair of

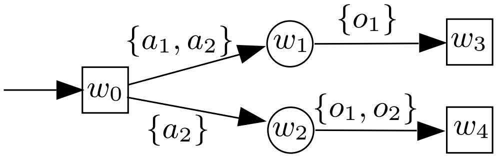


Figure 4.3: An example showing the difference between 'reaches' and 'reaches exactly' as distinguished in notation as $\mathcal{S}^{\mathtt{G}}_{w}$ and $\mathbb{S}^{\mathtt{G}}_{\{w\}}$.

nodes $(v \in V(\mathtt{P}), w \in V(\mathtt{W}))$ reached by that execution simultaneously, the following conditions hold:

1) if v and w are both action nodes and, for every label borne by each edge originating at v, there exist edges originating at w bearing the same action label;

2) if v and w are both observation nodes and, for every label borne by each edge originating at w, there exist edges originating at v bearing the same observation label;

3) if $v \in V_{\text{term}}$ and then $w \in V_{\text{goal}}$;

4) if $v \notin V_{\text{term}}$ then some extended joint-execution exists, continuing from v and w, that does reach the termination region.

In the above, properties 1) and 2) describe a notion of safety; property 3) of correctness; and 4) of liveness. In the previous definition, there is an upper bound on joint-execution length. We say that plan $(\mathtt{P}, V_{\text{term}})$ is *c-bounded* if, $\forall s \in \mathcal{L}^{I}(\mathtt{P})$, $|s| \leq c$.

4.1.3 Information disclosure policy, divulged plan, and observer

The observer sees a stream of the robot's actions and observations, and uses them to build estimates (or to compute general properties) of the robot's interaction with the world. But the observer's access to this information will usually be imperfect—either by design, as a consequence of real-world imperfections, or some combination of both. Conceptually, this is a form of partial observability in which the stream of symbols emitted as part of the robot's execution is distorted into to the symbols seen by the observer (see Figure 4.4). For example, if some pairs of actions are indistinguishable from the perspective of the observer, this may be expressed with a function

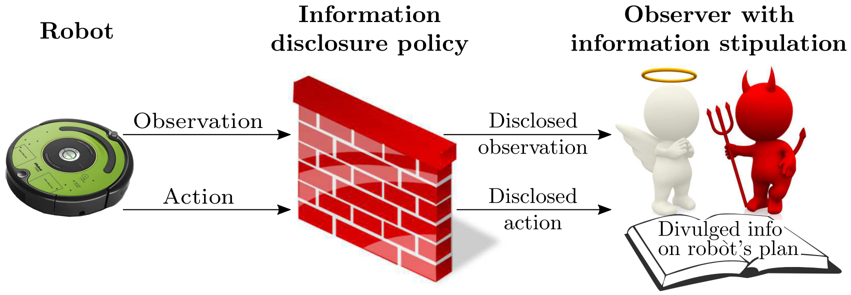

Robot
Information disclosure policy
Observer with information stipulation
Observation
Action
Disclosed observation
Disclosed action
Divulged info on robot's plan

Figure 4.4: The information disclosure policy, divulged plan and information stipulation. Even when the observer is a strong adversary, the disclosure policy and divulged plan can limit the observer's capabilities effectively.

that maps those pairs of actions to the same value. In this paper, this barrier is what we have been referring to (informally, thus far) with the phrase *information disclosure policy*. It is formalized as a mapping from the events in the robot's true execution in the world p-graph to the events received by the observer.

Definition 4.10 (Information disclosure policy)**.** An *information disclosure policy* is a label map h on p-graph $\mathtt{G}$, mapping from elements in the combined observation and action space $Y(\mathtt{G}) \cup U(\mathtt{G})$ to some set of events X.

The word "policy" hints at two interpretations: first, as something given as a predetermined arrangement (that is, as a rule); secondly, as something to be sought (as in finding a policy to solve a decision problem). Both senses apply in the present work; the exact transformation describing the disclosure of information will be used first (in Section 4.2) as a specification and then, later (in Section 4.4.3) as something which planning algorithms can produce. How the information disclosure policy is realized in some setting depends on which sense is apt: it can be interpreted as describing observers (showing that for those observers unable to tell y_i from y_j, the stipulations can be met), or it can inform robot operation (the stipulations require that the robot obfuscate u_ℓ and u_m via means such as explicit concealment, sleight-of-hand, misdirection, etc.)

The observer, in addition, may also have imperfect knowledge of robot's plan, which is leaked or communicated from the side-channel. The *disclosed plan* is also modeled as a p-graph, which may be weaker than knowing the actual plan. A variety of different types of divulged plan are introduced later (in Section 4.3.1) to model different prior knowledge available to an observer; as we will show, despite their differences, they can be treated in a single unified way.

The next step is to provide formal definitions for the ideas just described. In the following, we refer to h as the map from the set $Y \cup U$ to some set X, and refer to its preimage h^{-1} as the map from X to subsets of $Y \cup U$. The notation for a label map h and its preimage h^{-1} is extended in the usual way to sequences and sets. We consider sets of events, executions (being sequences), and sets of executions, and they are also extended to p-graphs in the obvious way, by applying the function to all edges:

Events Given any set of events $L \subseteq Y \cup U$, its image is $h[L] = \{h(\ell) \mid \ell \in L\}$. And conversely, for set $L' \subseteq X$, its preimage is $h^{-1}[L'] = \{\ell \in Y \cup U \mid h(\ell) \in L'\}$.

Executions Given any execution $s = \ell_0 \ell_1 \dots \ell_k$, where $\ell_i \in Y \cup U$, its image is $h(s) = h(\ell_0)h(\ell_1) \dots h(\ell_k)$, and for any execution $s' = \ell'_0 \ell'_1 \dots \ell'_k$, where $\ell'_i \in X$, its preimage is $h^{-1}(s') = \{s \mid h(s) = s'\}$.

Sets of executions Given any set of executions A, where $\forall s \in A, s \in (Y \cup U)^*$, its image is $h[A] = \{h(s) \mid s \in A\}$. Conversely for any set of executions A', where $\forall s' \in A', s' \in X^*$, its preimage is $h^{-1}[A'] = \{s \mid h(s) \in A'\}$.

P-graphs Given any p-graph $\mathtt{G} = (V_u \cup Y_u, Y, U, V_0)$, its image $h\langle \mathtt{G} \rangle = (V_u \cup Y_u, h[Y], h[U], V_0)$ is produced by replacing the set of events L on each edge e with $h[L]$. Analogously, given p-graph $\mathtt{G} = (V_u \cup Y_u, X_y, X_u, V_0)$, its preimage $h^{-1}\langle \mathtt{G} \rangle = (V_u \cup Y_u, h^{-1}[X_y], h^{-1}[X_u], V_0)$ is constructed by replacing the set of events L' on each edge e with $h^{-1}[L']$.

The function h can either preserve information (when it is a bijection) or it can lose information (by mapping multiple inputs to a single output). The loss of information is felt in $Y \cup U$ by the extent to which some $z \in Y \cup U$ grows under $h^{-1} \circ h$. In contrast, starting from $x \in X$, the uncertainty, measurable via set cardinality under h^{-1}, is washed out again when pushed forward to X. This idea is formalized with the following lemmas:

Lemma 4.1. For any event $\ell \in Y \cup U$, $h^{-1} \circ h(\ell) \supseteq \{\ell\}$. Similarly, we have $\forall L \subseteq Y \cup U, h^{-1} \circ h[L] \supseteq L$ and $\forall s = \ell_0\ell_1 \ldots \ell_n \in (Y \cup U)^*$, $s \in h^{-1} \circ h(s)$.

Proof. First, we are going to prove $\forall \ell \in Y \cup U$, $h^{-1} \circ h(\ell) \supseteq \{\ell\}$. Since h is a function, there are two cases for the images of the events in $\mathtt{G}$: First, $\forall \ell_1, \ell_2 \in U \cup Y$, $h(\ell_1) \neq h(\ell_2)$. In this case, no two events are mapped to the same output. In other words, each image element has a unique preimage, $\{\ell\} = h^{-1}(h(\ell))$. Secondly, $\exists \ell_1, \ell_2 \in U(\mathtt{G}) \cup Y(\mathtt{G})$, $h(\ell_1) = h(\ell_2)$. Then we have $h^{-1}(h(\ell_1)) = h^{-1}(h(\ell_2)) = \ell_1 \cup \ell_2$, $\{\ell_1\} \subset h^{-1}(h(\ell_1))$, and $\{\ell_2\} \subset h^{-1}(h(\ell_2))$. Hence, $h^{-1} \circ h(\ell) \supseteq \{\ell\}$.

Next, following the result of $h^{-1} \circ h(\ell) \supseteq \{\ell\}$, we have that $h^{-1} \circ h[L] = \cup_{\ell_i \in L} h^{-1} \circ h[\ell_i] \supseteq L$ for any $L \subseteq Y \cup U$.

Finally, we will prove $s \in h^{-1} \circ h(s)$ by induction for all $s = \ell_0\ell_1 \ldots \ell_n \in (Y \cup U)^*$. Let $s^k = \ell_0\ell_1 \ldots \ell_k$ be the prefix of s with length $k+1$, where $0 \leq k < n$. When $k = 0$, s^0 only contains an action or observation and, we have $s^0 \in h^{-1} \circ h(s^0)$. Suppose $s^k = \ell_0\ell_1 \ldots \ell_k \in h^{-1} \circ h(s^k)$ holds for k. The inductive step: $h^{-1} \circ h(s^{k+1}) = \cup_{\ell'_0\ell'_1\ldots\ell'_k \in h^{-1}\circ h(s^k)} \cup_{\ell'_{k+1} \in h^{-1}\circ h(\ell_{k+1})} \ell'_0\ell'_1 \ldots \ell'_k\ell'_{k+1} \ni \ell_0\ell_1 \ldots \ell_{k+1}$, which is since $\ell_0\ell_1 \ldots \ell_k \in h^{-1} \circ h(s^k)$ and $\ell_{k+1} \in h^{-1} \circ h(s^{k+1})$. Hence, $s_{k+1} \in h^{-1} \circ h(s^{k+1})$. Therefore, $s \in h^{-1} \circ h(s), \forall s \in (Y \cup U)^*$. □

Lemma 4.2. For any $\ell' \in X$, $h \circ h^{-1}(\ell') = \{\ell'\}$. Similarly, we have $\forall L' \subseteq X$, $h \circ h^{-1}[L'] = L'$ and $\forall s = \ell'_0\ell'_1 \ldots \ell'_k \in X^*$, $h \circ h^{-1}(s) = \{s\}$.

Proof. Firstly, we will prove $h \circ h^{-1}(\ell') = \{\ell'\}$ holds for any $\ell' \in X$. Let $h^{-1}(\ell') = \{l \in Y \cup U | h(l) = \ell'\}$. Then $\forall \ell \in h^{-1}(\ell')$, we have $h(\ell) = \ell'$. Therefore, $h \circ h^{-1}(\ell') = \{\ell'\}$.

Following from $h \circ h^{-1}(\ell') = \{l'\}$, we have $h \circ h^{-1}[L'] = L'$ for any $L' \subseteq X$.

Thirdly, we will prove $h \circ h^{-1}(s) = \{s\}$ by induction for any $s = \ell'_0\ell'_1 \ldots \ell'_n \in X^*$. Let s^k be the prefix of s with length $k+1$, where $0 \leq k < n$. When $k = 0$, s^0 only contains an action or observation and, we have $\{s^0\} = h \circ h^{-1}(s^0)$. Suppose $\{s^k\} = h \circ h^{-1}(s^k)$ holds for k. Then $h \circ h^{-1}(s^{k+1}) = \cup_{\ell''_0\ell''_1\ldots\ell''_k \in h\circ h^{-1}(s^k)} \cup_{\ell''_{k+1} \in h\circ h^{-1}(\ell''_{k+1})} \ell''_0\ell''_1 \ldots \ell''_{k+1} = \{\ell'_0\ell'_1 \ldots \ell'_{k+1}\}$. Hence, $h \circ h^{-1}(s^{k+1}) = \{s^{k+1}\}$. Therefore, $\forall s \in X^*, h \circ h^{-1}(s) = \{s\}$. □

Definition 4.11 (I-state graph)**.** For planning problem $(\mathtt{W}, V_{\text{goal}})$, plan $(\mathtt{P}, V_{\text{term}})$ and information disclosure policy $h : Y(\mathtt{W}) \cup U(\mathtt{W}) \to X$, an observer's *I-state graph* $\mathtt{I}$ is a p-graph, whose inputs are from the image space of h (i.e., $Y(\mathtt{I}) \cup U(\mathtt{I}) = X$), with $\mathcal{L}^I(\mathtt{I}) \supseteq h[\mathcal{L}^I(\mathtt{W})]$. The action space and observation space of $\mathtt{I}$ are also written as $X_u = U(\mathtt{I})$ and $X_y = Y(\mathtt{I})$.

The observer's I-state graph is a p-graph with events in the image space X. By having $\mathcal{L}^I(\mathtt{I}) \supseteq h[\mathcal{L}^I(\mathtt{W})]$, we are requiring that strings generated in the world can be safely traced on $\mathtt{I}$. Whether X_u and X_y are disjoint or not depends on their initial disjointedness and the structure of h.

Using the notation above, we will frequently speak of $h^{-1}\langle \mathtt{I} \rangle$. Next, are some basic properties of the vertices and executions of $\mathtt{I}$.

Lemma 4.3 (Properties of $\mathtt{I}$)**.** Given $\mathtt{I}$ and h, the following hold:

i. $\mathtt{I} = h \circ h^{-1}\langle \mathtt{I} \rangle$.

ii. $\forall s' \in \mathcal{L}^I(\mathtt{I}), \forall s \in h^{-1}(s'), \mathcal{V}_{\mathtt{I}}(s') = \mathcal{V}_{h^{-1}\langle \mathtt{I} \rangle}(s)$.

iii. $\mathcal{L}^I(\mathrm{h}^{-1}\langle \mathtt{I} \rangle) = h^{-1}[\mathcal{L}^I(\mathtt{I})]$.

iv. $\forall B \subseteq V(\mathtt{I}), h^{-1}[\mathbb{S}_B^{\mathtt{I}}] = \mathbb{S}_B^{h^{-1}\langle \mathtt{I} \rangle}$.

Property i. According to Lemma 4.2, each event set L in $\mathtt{I}$ will not change when we apply operation $h \circ h^{-1}$ on $\mathtt{I}$. Therefore, we have $\mathtt{I} = h\langle h^{-1}\langle \mathtt{I} \rangle \rangle$ by replacing every set of events L in $\mathtt{I}$ with $h \circ h^{-1}[L]$. □

Property ii. We need to prove that s' and its preimage s reach the same* set of vertices in $\mathtt{I}$ and $h^{-1}\langle \mathtt{I} \rangle$ respectively. According to the construction of $h^{-1}\langle \mathtt{I} \rangle$, we have $\forall s' \in \mathcal{L}^I(\mathtt{I}), \forall s \in h^{-1}(s'), \mathcal{V}_{\mathtt{I}}(s') \subseteq \mathcal{V}_{h^{-1}\langle \mathtt{I} \rangle}(s)$. Next, we will prove $\forall s' \in \mathcal{L}^I(\mathtt{I}), \forall s \in h^{-1}(s'), \mathcal{V}_{\mathtt{I}}(s') \supseteq \mathcal{V}_{h^{-1}\langle \mathtt{I} \rangle}(s)$ by contradiction. Suppose $\exists s' \in \mathcal{L}^I(\mathtt{I}), \exists s \in h^{-1}(s'), \mathcal{V}_{\mathtt{I}}(s') \not\supseteq \mathcal{V}_{h^{-1}\langle \mathtt{I} \rangle}(s)$. Then we have $\mathcal{V}_{\mathtt{I}}(s') \subset \mathcal{V}_{h^{-1}\langle \mathtt{I} \rangle}(s)$. If s is the preimage of only s', then we should have, according to the construction of $h^{-1}\langle \mathtt{I} \rangle$, $\mathcal{V}_{\mathtt{I}}(s') = \mathcal{V}_{h^{-1}\langle \mathtt{I} \rangle}(s)$ instead. Hence, s is the preimage of at least two different executions s' and s'', which contradicts with the fact that h is a function. Therefore, $\forall s' \in \mathcal{L}^I(\mathtt{I}), \forall s \in h^{-1}(s'), \mathcal{V}_{\mathtt{I}}(s') = \mathcal{V}_{h^{-1}\langle \mathtt{I}\rangle}(s)$.

$\forall s' \in \mathcal{L}^I(\mathtt{I}), \forall s \in h^{-1}(s')$, s' reach the same set of vertices as those reached by s in $h^{-1}\langle \mathtt{I}\rangle$. Since each vertex in $\mathcal{V}_{\mathtt{I}}(s')$ is isomorphic to the same one in $\mathcal{V}_{h^{-1}\langle \mathtt{I}\rangle}(s)$, we have $\mathcal{V}_{\mathtt{I}}(s')$ is identical to $\mathcal{V}_{h^{-1}\langle \mathtt{I}\rangle}(s)$. □

Property iii. $\Longrightarrow$: Given any execution s from p-graph $h^{-1}\langle \mathtt{I}\rangle$, we will prove $h(s)$ is an execution from p-graph $\mathtt{I}$. According to Lemma 4.3.*i*, $\mathtt{I} = h\langle h^{-1}\langle \mathtt{I}\rangle\rangle$. Thus, $h(s)$ is an execution on $\mathtt{I} = h\langle h^{-1}\langle \mathtt{I}\rangle\rangle$. And we have $\mathcal{L}^I(h^{-1}\langle \mathtt{I}\rangle) \subseteq h^{-1}[\mathcal{L}^I(\mathtt{I})]$.

$\Longleftarrow$: Given any execution $s \in h^{-1}[\mathcal{L}^I(\mathtt{I})]$, we will prove $s \in \mathcal{L}^I(h^{-1}\langle \mathtt{I}\rangle)$. For any $s \in h^{-1}[\mathcal{L}^I(\mathtt{I})]$, we have $h(s)$ is an execution from p-graph $\mathtt{I}$. According to Lemma 4.3.*ii.* the set of vertices reached by $h(s)$ in p-graph $\mathtt{I}$ is isomorphic to the set of vertices reached by $s' \in h^{-1}(h(s))$ in $h^{-1}\langle \mathtt{I}\rangle$. Hence, s is an execution in $h^{-1}\langle \mathtt{I}\rangle$. Therefore, $\mathcal{L}^I(h^{-1}\langle \mathtt{I}\rangle) \supseteq h^{-1}[\mathcal{L}^I(\mathtt{I})]$. □

Property iv. $\Longrightarrow$: Given any execution $s \in h^{-1}[\mathbb{S}^{\mathtt{I}}_B]$, then we have $h(s) \in \mathbb{S}^{\mathtt{I}}_B$ and $\mathcal{V}_{\mathtt{I}}(h(s)) = B$. According to Lemma 4.3.*ii*, we have $\mathcal{V}_{h^{-1}\langle \mathtt{I}\rangle}(s) = \mathcal{V}_{\mathtt{I}}(h(s)) = B$. Hence, $s \in \mathbb{S}^{h^{-1}\langle \mathtt{I}\rangle}_B$.

$\Longleftarrow$: Given any execution $s \in \mathbb{S}^{h^{-1}\langle \mathtt{I}\rangle}_B$, then we have $\mathcal{V}_{h^{-1}\langle \mathtt{I}\rangle}(s) = B$. According to Lemma 4.3.*ii*, $\mathcal{V}_{\mathtt{I}}(h(s)) = \mathcal{V}_{h^{-1}\langle \mathtt{I}\rangle}(s) = B$. Hence, $h(s) \in \mathbb{S}^{\mathtt{I}}_B$ and therefore, we now have $s \in h^{-1}[\mathbb{S}^{\mathtt{I}}_B]$. □

Next we present a core definition of the paper. The crucial aspect to be formalized is the connection from the interaction of the robot and world, via the stream of symbols generated, to the state tracked by the observer. Inference runs from observer back to the world, but causality proceeds from the robot–world to observer (glance again at Figure 5.2). We begin, accordingly, with that latter direction.

Definition 4.12 (compatible world states)**.** Given observer I-state graph $\mathtt{I}$, robot's plan $(\mathtt{P}, V_{\text{term}})$, world graph $(\mathtt{W}, V_{\text{goal}})$, and label map h, the world state w is *compatible* with the set of I-states

[*] Since, by $h^{-1}\langle \mathtt{I}\rangle$ we refer to the graph $\mathtt{I}$ with each of the edge labels replaced by preimages under h, there is a one-to-one correspondence between the two graphs via a natural isomorphism. For convenience we speak of the "same" vertex rather than being explicit about the associated bijection and, further, we have used '$=$' rather than '$\cong$'.

$$B \subseteq V(\mathtt{I}) \text{ if } \exists s \in \mathcal{L}^I(\mathtt{W}) \text{ such that } s \in \underbrace{h^{-1}[\mathbb{S}^{\mathtt{I}}_B]}_{(1)} \cap \underbrace{\mathcal{L}^I(\mathtt{P})}_{(2)} \cap \underbrace{\mathcal{S}^{\mathtt{W}}_w}_{(3)}.$$

Informally, each of the three terms can be interpreted as:

(1) An observer with I-state graph $\mathtt{I}$ may ask which sequences are responsible for having arrived at states B. The answer is the set $\mathbb{S}^{\mathtt{I}}_B$, an equivalence class of strings identical up to those states, the executions contained therein being indistinguishable up to states in $\mathtt{I}$. Those strings are in the image space X, so, to obtain an answer in the world $Y \cup U$, their preimages must be taken. Every execution in $h^{-1}[\mathbb{S}^{\mathtt{I}}_B]$ leads the observer to B. Note that information is degraded both by h and $\mathtt{I}$. Figure 4.5 provides a visual example that shows how information can be degraded by a label map h, an I-state graph $\mathtt{I}$, and both together. The leftmost sub-figure gives a scenario by providing a world p-graph $\mathtt{W}$, a plan $\mathtt{P}$, and divulged plan information $\mathtt{D}$ — all three are identical p-graphs. The next sub-figure (second from the left) shows an I-state graph with the same structure as $\mathtt{W}$ and an identity label map. Every I-state corresponds to a single world state in this case. In the third sub-figure, there is an I-state graph with the same structure as $\mathtt{W}$, thus clearly possessing sufficient structure to account for the world states. But here a label map conflates some actions and some observations. A consequence is that the world states w_1 and w_2 are indistinguishable given I-state i_1 and plan $\mathtt{P}$. In the rightmost sub-figure both h and $\mathtt{I}$ degrade information and do so independently. In this case, w_3 and w_4 are indistinguishable owing to the label map, w_5 and w_6 are indistinguishable owing to the collapsed structure in $\mathtt{I}$.

(2) The set of executions that may be executed by the robot is represented by $\mathcal{L}^I(\mathtt{P})$. If the observer knows that the robot's plan takes, say, the high road, this information allows the observer to remove executions involving the robot along the low road.

(3) The set of executions reaching world state w is represented by $\mathcal{S}^{\mathtt{W}}_w$. Two world states $w, w' \in V(\mathtt{W})$ are essentially indiscernible or indistinguishable if $\mathcal{S}^{\mathtt{W}}_w = \mathcal{S}^{\mathtt{W}}_{w'}$, as the sets capture the intrinsic uncertainty of world $\mathtt{W}$.

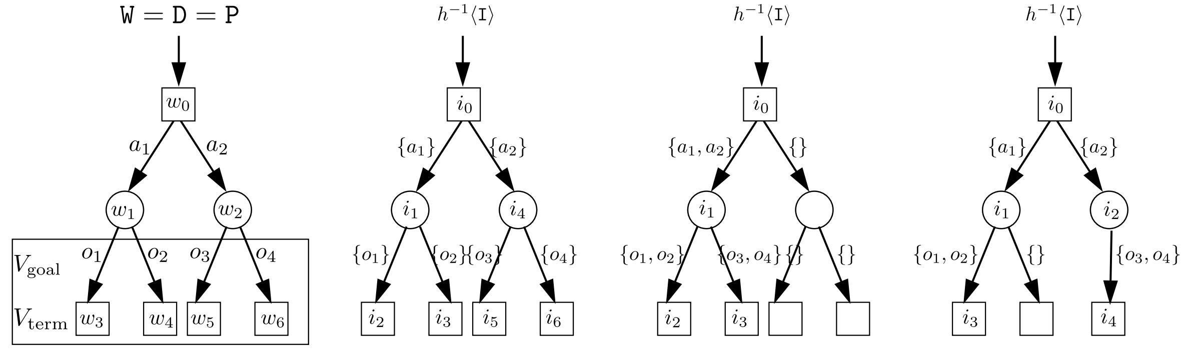


Figure 4.5: Both the label map and the I-state graph can degrade information. Leftmost: a scenario where the world p-graph W, a plan P, and divulged plan information D are all identical. Second from the left: an I-state graph I with the same structure as W and an identity label map h. Second from the right: an I-state graph with I the same structure as W and a label map h which conflates some actions/observations. Rightmost: both h and I degrade information independently.

When an observer is in B, and w is compatible with B, there exists some execution, a certificate, that the world could plausibly be in w subject to (1) the current information summarized in I; (2) the robot's plan; (3) the structure of the world. The set of *all* world states that are compatible with B is denoted $\mathcal{W}_B^{\mathtt{I},\mathtt{P}}$, which is the observer's estimate of the world states when known information about W, P and I have all been incorporated.

A typical observer may know less about the robot's future behavior than the robot's full plan. Weaker knowledge of how the robot will behave can be expressed in terms of some p-graph D, such that $\mathcal{L}^I(\mathtt{D}) \supseteq \mathcal{L}^I(\mathtt{P})$. Here the mnemonic is that it is the divulged information about the robot's plan, which one might imagine as leaked or communicated via a side-channel. Another decision is that the information divulged to the observer about the robot's execution is in the preimage space. This modeling decision may seem strange at first blush, so we provide some explanation and justification for it. As the observer will only see things in the image space, it may seem that granting access to information in the preimage or the image space would have little difference. But, since inference occurs by pulling back observed events to preimage space and then taking an intersection, there actually can be an appreciable difference. As this paper is interested in a worst-case adversarial conditions, we are interested in what strong ne'er-do-well observers might infer and thus study the problem where the adversary gains the maximum possible. From this setting

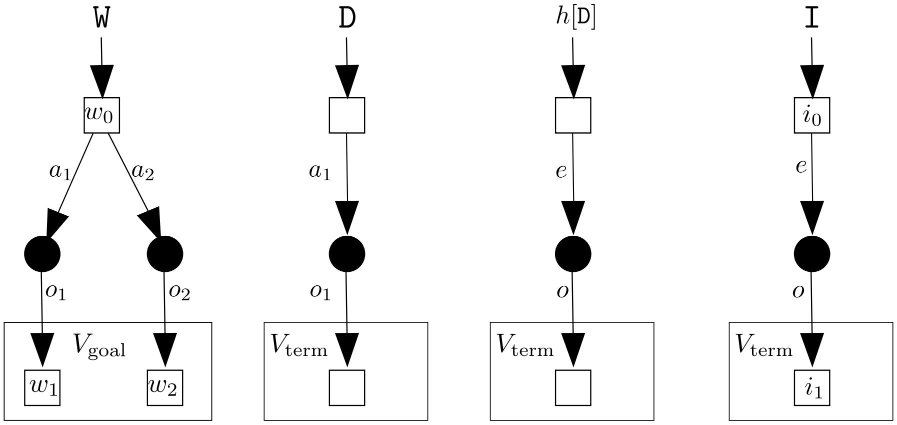


Figure 4.6: Find the estimated world states when given world graph $\mathtt{W}$, I-state graph $\mathtt{I}$, divulged graph $\mathtt{D}$ or $h\langle\mathtt{D}\rangle$, label map $h = \{a_1, a_2 \mapsto a; o_1, o_2 \mapsto o\}$.

the other variant can be easily posed as well (one simply needs to consider $h^{-1} \circ h\langle\mathtt{D}\rangle$ to simulate the knowledge of image space information).

To clarify the previous statements, made only informally, we must be more formal with the image space inference process:

Definition 4.13. Given an I-state graph $\mathtt{I}$, divulged information about the robot's behavior $h\langle\mathtt{D}\rangle$ in the image space, and label map h, the set of estimated world states for I-states $B \subseteq V(\mathtt{I})$ is $\mathcal{W}_B^{\mathtt{I},h\langle\mathtt{D}\rangle} = \{w \in V(\mathtt{W}) | h^{-1}[\mathbb{S}_B^{\mathtt{I}}] \cap \mathcal{L}^I(h^{-1} \circ h\langle\mathtt{D}\rangle) \cap \mathcal{S}_w^{\mathtt{W}} \neq \varnothing\}$.

Lemma 4.4. Given any p-graph $\mathtt{D}$, $\mathcal{L}^I(\mathtt{D}) \subseteq \mathcal{L}^I(h^{-1} \circ h\langle\mathtt{D}\rangle)$.

Proof. According to Lemma 4.1, for event $\ell \in \mathcal{L}^I(\mathtt{D})$, we have $\{\ell\} \subseteq \mathcal{L}^I(h^{-1} \circ h(\ell))$. Then the set of events bearing in each edge of p-graph $h^{-1} \circ \langle\mathtt{D}\rangle$ is a superset of the corresponding edge in p-graph $\mathtt{D}$. Therefore, $\mathcal{L}^I(\mathtt{D}) \subseteq \mathcal{L}^I(h^{-1} \circ h\langle\mathtt{D}\rangle)$. □

Theorem 4.1. Given I-state graph $\mathtt{I}$, divulged information $\mathtt{D}$, world graph $\mathtt{W}$, and label map h, the set of estimated world states for any set of I-states $B \subseteq V(\mathtt{I})$ is $\mathcal{W}_B^{\mathtt{I},\mathtt{D}}$. By replacing $\mathtt{D}$ with its image graph $h\langle\mathtt{D}\rangle$, the set of estimated world states for B is $\mathcal{W}_B^{\mathtt{I},h\langle\mathtt{D}\rangle}$. $\forall B \subseteq V(\mathtt{I})$, $\mathcal{W}_B^{\mathtt{I},h\langle\mathtt{D}\rangle} \supseteq \mathcal{W}_B^{\mathtt{I},\mathtt{D}}$.

Proof. According to Lemma 4.4, $\mathcal{L}^I(\mathtt{D}) \subseteq \mathcal{L}^I(h^{-1} \circ h\langle\mathtt{D}\rangle)$. Thus $\forall B \subseteq V(\mathtt{I}), \forall w \subseteq V(\mathtt{W})$, we have $h^{-1}[\mathbb{S}_B^{\mathtt{I}}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_w^{\mathtt{W}} \subseteq h^{-1}[\mathbb{S}_B^{\mathtt{I}}] \cap \mathcal{L}^I(h^{-1} \circ h\langle\mathtt{D}\rangle) \cap \mathcal{S}_w^{\mathtt{W}}$. If $h^{-1}[\mathbb{S}_B^{\mathtt{I}}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_w^{\mathtt{W}} \neq \varnothing$, then

$h^{-1}[\mathbb{S}^{\mathtt{I}}_{B}] \cap \mathcal{L}^{I}(h^{-1} \circ h\langle \mathtt{D} \rangle) \cap \mathcal{S}^{\mathtt{W}}_{w} \neq \varnothing$. Thus, if $v \in \mathcal{W}^{\mathtt{I},\mathtt{D}}_{B}$, then $v \in \mathcal{W}^{\mathtt{I},h\langle \mathtt{D} \rangle}_{B}$. Hence, $\forall B \subseteq V(\mathtt{I})$, we have $\mathcal{W}^{\mathtt{I},h\langle \mathtt{D} \rangle}_{B} \supseteq \mathcal{W}^{\mathtt{I},\mathtt{D}}_{B}$. □

An example to illustrate Theorem 4.1 is shown in Figure 4.6. Given W and I, $\mathcal{W}^{\mathtt{I},\mathtt{D}}_{\{i_0\}} = \{w_1\}$, while $\mathcal{W}^{\mathtt{I},h\langle \mathtt{D} \rangle}_{\{i_0\}} = \{w_1, w_2\}$. Hence, $\mathcal{W}^{\mathtt{I},h\langle \mathtt{D} \rangle}_{B} \supseteq \mathcal{W}^{\mathtt{I},\mathtt{D}}_{B}$.

Definition 4.12 requires the simple substitution of the second term in the intersection with $\mathcal{L}^{I}(\mathtt{D})$. When only D is given, one can only approximate $\mathcal{W}^{\mathtt{I},\mathtt{P}}_{B}$ with $\mathcal{W}^{\mathtt{I},\mathtt{D}}_{B}$:

Definition 4.14 (estimated world states)**.** Given an I-state graph I, divulged plan p-graph D, world p-graph W, and label map h, the set of estimated world states for I-states $B \subseteq V(\mathtt{I})$ is $\mathcal{W}^{\mathtt{I},\mathtt{D}}_{B} := \left\{ w \in V(\mathtt{W}) \;\middle|\; (\mathbb{S}^{h^{-1}\langle \mathtt{I} \rangle}_{B} \cap \mathcal{L}^{I}(\mathtt{D}) \cap \mathcal{S}^{\mathtt{W}}_{w}) \neq \varnothing \right\}$.

Note that $h^{-1}[\mathbb{S}^{\mathtt{I}}_{B}]$ has been replaced with $\mathbb{S}^{h^{-1}\langle \mathtt{I} \rangle}_{B}$, via Lemma 4.3.*iii*.

Via reasoning that is entirely analogous (hence we minimize discussion in detail), the observer may estimate the robot's plan states:

Definition 4.15 (estimated plan states)**.** Given an I-state graph I, divulged plan graph D, world graph W, and label map h, the set of estimated world states for I-states $B \subseteq V(\mathtt{I})$ is $\mathcal{D}^{\mathtt{I},\mathtt{W}}_{B} := \left\{ d \in V(\mathtt{D}) \;\middle|\; (\mathbb{S}^{h^{-1}\langle \mathtt{I} \rangle}_{B} \cap \mathcal{L}^{I}(\mathtt{W}) \cap \mathcal{S}^{\mathtt{D}}_{d}) \neq \varnothing \right\}$.

These estimated plan states are of 'second order,' as they represent the observer's knowledge about what the robot knows.

The last remaining element in Figure 4.4 that needs to be addressed is the stipulation of information. We do that next.

4.1.4 Information stipulations

We prescribe properties of the information that an observer may extract from its input by imposing constraints on the sets of estimated world states. The observer, filtering a stream of inputs sequentially, forms a correspondence between its I-states and world states. We write propositional formulas with semantics defined in terms of this correspondence—in this model the stipulations

$$\begin{aligned} &\text{Formula} \rightarrow \text{Clause}_1 \wedge \ldots \wedge \text{Clause}_n \\ &\text{Clause} \rightarrow \text{Literal}_1 \vee \ldots \vee \text{Literal}_m \\ &\text{Literal} \rightarrow \text{Symbol} \mid \neg\text{Symbol} \\ &\text{Symbol} \rightarrow v_0, v_1, v_2, \ldots \end{aligned}$$

$$\text{[VALUE]}\ \frac{}{\langle v_i \rangle \Downarrow \text{eval}(v_i \overset{?}{\in} \mathcal{W}_B^{\mathtt{I,D}})} \qquad \frac{\text{[NOT]} \quad \langle v_i \rangle \Downarrow w}{\langle \neg v_i \rangle \Downarrow \text{the negation of } w}$$

$$\frac{\text{[OR]} \quad \langle \ell_1 \rangle \Downarrow w_1 \quad \langle \ell_2 \rangle \Downarrow w_2}{\langle \ell_1 \vee \ell_2 \rangle \Downarrow \text{the logical or of } w_1 \text{ and } w_2} \qquad \frac{\text{[AND]} \quad \langle c_1 \rangle \Downarrow w_1 \quad \langle c_2 \rangle \Downarrow w_2}{\langle c_1 \wedge c_2 \rangle \Downarrow \text{the logical and of } w_1 \text{ and } w_2}$$

Figure 4.7: The syntax and natural semantics of the information stipulations, where c_i, ℓ_i, v_i, represent a clause, literal, and symbol, respectively, and w_i is the result of the evaluation. The transition $\langle e \rangle \Downarrow w$ denotes a transition, where e is any expression defined by the grammar and w is the value yielded by the expression.

are written to hold over *every* reachable set of associated states.[†]

First, however, we must delineate the scope of the estimated world states to be constrained. Some states, in inherently non-deterministic worlds, may be inseparable because they are reached by the same execution. In Figure 4.3, both w_3 and w_4 will be reached (non-deterministically) by execution $a_2 o_1$. Since this is intrinsic to the world, even when the observer has perfect observations, they remain indistinguishable. In the remainder of this paper, we will assume that the world graph $\mathtt{W}$ is in state-determined form, and we may affix stipulations to the world states knowing that no two vertices will be non-deterministically reached by the same execution. Similarly, when applicable, $\mathtt{D}$ will also be presumed to be in state-determined form with the stipulations written in terms of these states.[‡]

Second, we write propositional formulae to constrain the observer's estimate. The formula $\mathbf{\Phi}$ is written in conjunctive normal form, consisting of symbols, literals and clauses as shown in Fig. 4.7. Firstly, a basic, atomic symbol v_i is associated with each world state $v_i \in V(\mathtt{W})$ or plan state $v_i \in V(\mathtt{D})$. If v_i is contained in the observer's estimates $\mathcal{W}_B^{\mathtt{I,D}}$ or $\mathcal{D}_B^{\mathtt{I,W}}$, we will evaluate the

[†] We foresee other variants which are straightforward to modifications to consider; but we report only on our current implementation.

[‡] Recall that every p-graph has a state-determined presentation [12].

corresponding symbol v_i as $True$. Otherwise, it evaluates as $False$. With each symbol grounded, we can evaluate literals and clauses compositionally, using logic operators NOT, AND, OR. These are defined naturally, eventually enabling evaluation of $\boldsymbol{\Phi}$ on the observer's estimate $\mathcal{W}_B^{\mathtt{I},\mathtt{D}}$ and $\mathcal{D}_B^{\mathtt{I},\mathtt{W}}$. Suppose, for example, we wish to require that state v_1 be included in the observer's estimates of the world whenever v_2 is; this would be expressed via $\boldsymbol{\Phi} = \neg v_2 \vee v_1$. Evaluation of such formulas takes place as follows. For a set of I-states B reached under some operation of the robot in the world, v_i is connected with v_i in that v_i evaluates to True for B iff $v_i \in \mathcal{W}_B^{\mathtt{I},\mathtt{D}}$, where $\mathcal{W}_B^{\mathtt{I},\mathtt{D}}$ is the set of estimated world states for I-states B. The opposite condition, where $v_i \notin \mathcal{W}_B^{\mathtt{I},\mathtt{D}}$, is written naturally as $\neg v_i$. Standard connectives $\neg$, $\wedge$, $\vee$ enable composite expressions for complex stipulations to be built and recursively evaluated.

Let the predicate $\mathrm{satfd}(B, \boldsymbol{\Phi})$ denote whether the stipulation $\boldsymbol{\Phi}$ holds for I-states B. Then a plan $\mathtt{P}$ satisfies the stipulations, if and only if

$$\forall s \in \mathcal{L}^I(\mathtt{P}) \cap \mathcal{L}^I(\mathtt{W})\ B = \mathcal{V}_{\mathtt{I}}(h(s))\ \ \mathrm{satfd}(B, \boldsymbol{\Phi}).$$

4.2 Verifying plans and stipulations: the Check problem

Given everything involved, an important initial step is to successfully recognize a solution to the problem, including determining whether the constraints have been met.

Problem: Check$\big((\mathtt{W}, V_{\text{goal}}), (\mathtt{P}, V_{\text{term}}), (\mathtt{D}, \mathtt{I}), h, \boldsymbol{\Phi}\big)$

Input: A planning problem $(\mathtt{W}, V_{\text{goal}})$, a plan $(\mathtt{P}, V_{\text{term}})$, a divulged plan p-graph $\mathtt{D}$, an I-state graph $\mathtt{I}$, an information disclosure policy h and an information stipulation $\boldsymbol{\Phi}$.

Output: True if plan $(\mathtt{P}, V_{\text{term}})$ solves the problem $(\mathtt{W}, V_{\text{goal}})$, and $\forall s \in \mathcal{L}^I(W^\dagger) \cap \mathcal{L}^I(\mathtt{P}), B = \mathcal{V}_{\mathtt{I}}(h(s))$, the information stipulation $\boldsymbol{\Phi}$ is always evaluated as True on $\mathcal{W}_B^{\mathtt{I},\mathtt{D}}$ and $\mathcal{D}_B^{\mathtt{I},\mathtt{W}}$ (i.e. $\mathrm{satfd}(B, \boldsymbol{\Phi}) = True$); False otherwise.

One thing demands further explanation: $\mathtt{W}^\dagger$ is used as a replacement for the world $\mathtt{W}$; this deals with a technical nuance, a sort of inference gained for free upfront, most easily handled by transforming the inputs. Given a world and a disclosed p-graph, oftentimes certain parts of the p-graph can be determined to be irrelevant *a priori*—for instance, if we know, via $\mathtt{D}$, that the robot is executing a plan, then all non-goal cul-de-sacs can be excised (yielding a $\mathtt{W}^{\dagger}$ with $\mathcal{L}^{I}(\mathtt{W}^{\dagger}) \subseteq \mathcal{L}^{I}(\mathtt{W})$).

4.2.1 Does the plan solve the planning problem?

To determine whether the plan $(\mathtt{P}, V_{\text{term}})$ solves the planning problem $(\mathtt{W}, V_{\text{goal}})$, safety, correctness and liveness must all be checked. The procedure is shown in Algorithm 2. A breath-first search (BFS) of the product graph $\mathtt{W} \otimes \mathtt{P}$ permits these three properties to be verified. Examination of safety and correctness follow directly from their definitions. Liveness is violated if there is a loop in the product graph prior to terminating states being reached, or when a joint-execution cannot be extended to reach the termination region.

4.2.2 Are the stipulations satisfied?

An important preliminary step to determine whether the stipulation is satisfied is to establish the correspondence from sets of observer I-states to estimated world states. To accomplish this, for sets B of I-states, we compute $\mathcal{W}_{B}^{\mathtt{I},\mathtt{D}}$ and $\mathcal{D}_{B}^{\mathtt{I},\mathtt{W}}$.

First, one examines those sets B of I-states that can arise by dint of the observer perceiving the image of executions under h. According to Definition 4.7, those sets correspond to equivalence classes of the images of executions. We can obtain exactly the sets of I-states of interest by expanding $\mathtt{I}$ into its state-determined form $\textsc{Sde}(\mathtt{I})$, the expansion process produces a single new state for each equivalence class. Following this, the preimage p-graph $h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle$ will also be in state-determined form.

Next, we find the estimated world states for each vertex in $\textsc{Sde}(\mathtt{I})$, by simply realizing Definition 4.14 and 4.15 constructively: a world state w corresponds with an observer I-state $v \in V(\textsc{Sde}(\mathtt{I}))$ if there exists a joint-execution in $h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle$, $\mathtt{W}$ and $\mathtt{D}$ that reaches I-state v, w, and some plan state. Note that $\textsc{Sde}(\mathtt{I})$ and $h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle$ share the same set of vertices. This correspondence can be established easily via a graph $\mathtt{T} := \mathtt{W} \otimes \mathtt{D} \otimes h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle$ of triples (see Figure 4.8). In $\mathtt{T}$ each vertex $(v^{\mathtt{W}}, v^{\mathtt{D}}, v^{h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle}) \in V(T)$ represents the fact that world state $v^{\mathtt{W}}$ and plan state $v^{\mathtt{D}}$ are associated with the observer I-state $v^{h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle}$. Iterating over all vertices in

Algorithm 2: CheckSoln($\mathtt{W}, V_{\text{goal}}, \mathtt{P}, V_{\text{term}}$)

```
$Q \leftarrow []$ and visited $\leftarrow []$
for $w \in V_0(\mathtt{W})$ do
    for $v \in V_0(\mathtt{P})$ do
        if $v \in V_{\text{term}}$ and $w \notin V_{\text{goal}}$ then
            return False // Not correct
        else if $v \notin V_{\text{term}}$ then
            $Q$.append($(w, v)$)
            visited.append($(w, v)$)
        end if
    end for
end for
while $Q$ not empty do
    $(w, v) \leftarrow Q$.pop
    $N_w \leftarrow \mathtt{W}$.outNeighbors($w$)
    $N_v \leftarrow \mathtt{P}$.outNeighbors($v$)
    if $N_w$ is empty and $N_v$ not empty then
        return False // Not safe and not live
    end if
    if $w$ is action vertex and $N_v \not\subseteq N_w$ then
        return False // Not safe on action
    else if $w$ is observation vertex and $N_w \not\subseteq N_v$ then
        return False // Not safe on observation
    end if
    for $w \in N_w$ do
        for $v \in N_v$ do
            if $w \notin V_{\text{goal}}$ and $v \in V_{\text{term}}$ then
                return False // Not correct or live
            else if $w \in V_{\text{goal}}$ and $v \in V_{\text{term}}$ then
                continue // Terminating vertex
            else if $(w, v) \in$ visited then
                return False // Loop detected, not finite
            else
                $Q$.append($(w, v)$)
                visited.append($(w, v)$)
            end if
        end for
    end for
end while
return True // Passing all the tests
```

$V(T)$, we thus find the set of world states $\mathcal{W}^{\texttt{I},\texttt{D}}_{\{v\}}$ and plan states $\mathcal{D}^{\texttt{I},\texttt{W}}_{\{v\}}$ corresponding with I-state v. The correspondence is both complete and tight: completeness follows from the exhaustiveness of the enumeration; tightness from the fact that an association is formed only when some actual execution s can simultaneously take the world to a state and the observer to an I-state.

For any vertex $v \in V(h^{-1}\langle \text{SDE}(\texttt{I})\rangle)$, we can use $\mathcal{W}^{\texttt{I},\texttt{D}}_{\{v\}}$ and $\mathcal{D}^{\texttt{I},\texttt{W}}_{\{v\}}$ to evaluate the information stipulations. Each vertex v in $\text{SDE}(\texttt{I})$ can be marked as either $\text{satfd}(\{v\}, \mathbf{\Phi}) = True$ when the stipulation holds, or $\text{satfd}(\{v\}, \mathbf{\Phi}) = False$ otherwise.

Next, we make another product graph $\texttt{W} \otimes \texttt{P} \otimes h^{-1}\langle \text{SDE}(\texttt{I})\rangle$ to mark whether the I-states in $\text{SDE}(\texttt{I})$ are reached by some execution in the joint execution $\mathcal{L}^{I}(\texttt{W}) \cap \mathcal{L}^{I}(\texttt{P})$. Finally, the objective is to check whether every I-state reached by the plan P (since what is really happening is P rather than D) satsifies the stipulations.

4.3 Discussion regarding the observer's model

Above, we hinted that observers may differ depending on the prior knowledge that has been revealed to them; next we bring this idea into sharper focus. The information associated with an observer is contained in a pair $(\texttt{D}, \texttt{I})$: D captures the observer's prior knowledge about all possible executions that the robot may produce according to its plan, the I-state graph I that acts as a filter, succinctly tracking state from a stream of inputs, and knowledge of robot's plan. Then the I-state graph I induces $\underset{\texttt{I}}{\sim}$ over its set of executions and hence over the joint-executions with the

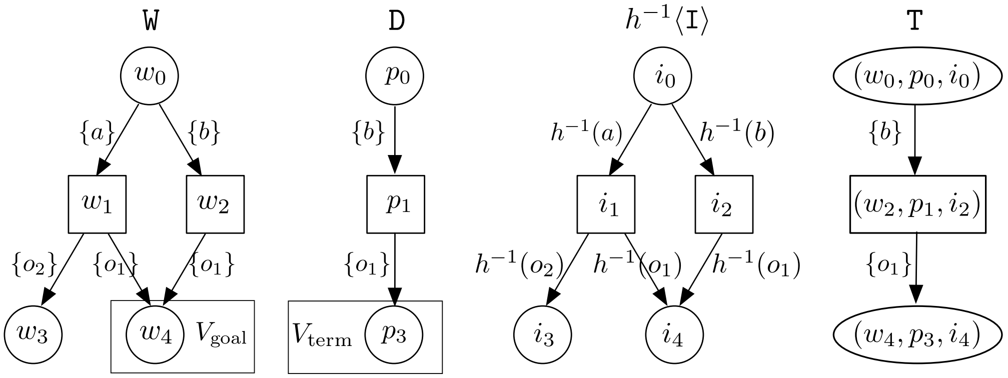


Figure 4.8: An example of product graph T formed from some W, D, and $h^{-1}\langle\texttt{I}\rangle$.

world, or, more precisely, the image of those through h. (Recall that the coarseness of h limits the fidelity of the observer, cf. Figure 4.5.) By comparing the fineness of the relations induced by two I-state graphs, one obtains a sense of the relative coarseness of the two I-state graphs. As the present paper describes methods motivated by applications to robotic privacy, we model the most capable adversary, taking the *finest observer*, that is, one whose equivalence classes are as small as possible. In this section, we will start by describing various possibilities for $\mathtt{D}$, and show that $\mathtt{D}$ can be equivalently represented as a p-graph. Thereafter, we will give the definition and a construction of the finest observer.

4.3.1 Observer's prior knowledge about robot's plan

The first element in the observer pair is $\mathtt{D}$, information disclosed about the plan, and presumed to be known *a priori*, to the observer. Depending on how much the observer knows, there are multiple possibilities here, from most- to least-informed:

I. The observer knows the exact plan $\mathtt{P}$ to be executed.

II. The plan to be executed is among a finite collection of plans $\{\mathtt{P}_1, \mathtt{P}_2, \dots, \mathtt{P}_\mathtt{n}\}$.

III. The observer may only know that the robot is executing *some* plan, that is, the robot is goal directed and aims to achieve some state in V_{goal}.

IV. The observer knows nothing about the robot's execution other than that it is on $\mathtt{W}$.

Definition 4.14 and 4.15 detail how the observer's knowledge of the world state ($\mathcal{W}_B^{\mathtt{I},\mathtt{D}}$) and plan state ($\mathcal{D}_B^{\mathtt{I},\mathtt{W}}$) from I-states B depend on $\mathbb{S}_B^{h^{-1}\langle\mathtt{I}\rangle} \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{L}^I(\mathtt{W})$, a set of executions that arrive at B in the I-state graph $\mathtt{I}$. Because the observer uses $\mathtt{D}$ to refine $\mathbb{S}_B^{h^{-1}\langle\mathtt{I}\rangle}$, when $\mathcal{L}^I(\mathtt{P}) \subsetneq \mathcal{L}^I(\mathtt{D})$ the gap between the two sets of executions represents a form of uncertainty. The ordering of the four cases, thus, can be stated precisely in terms language inclusion.

A p-graph exists whose language expresses knowledge for each of these cases:

Case I. When $\mathtt{D} = \mathtt{P}$, the interpretation is straightforward: the observer tracks the states of the plan given the stream of observations (as best as possible, as the operation is under h).

Case II. If instead a set of plans $\{\mathtt{P}_1, \mathtt{P}_2, \dots, \mathtt{P}_\mathtt{n}\}$ is given, we must construct a single p-graph, $\mathtt{D}$, so that $\mathcal{L}^I(\mathtt{D}) = \mathcal{L}^I(\mathtt{P}_1) \cup \cdots \cup \mathcal{L}^I(\mathtt{P}_\mathtt{n})$. This is achieved via the union of p-graphs $D = \mathtt{P}_1 \uplus \mathtt{P}_2 \uplus \cdots \uplus \mathtt{P}_\mathtt{n}$, cf. [12, p. 244].

Case III. If the robot is known only to be executing some plan, we must consider the set of all plans, $\mathtt{P}^\infty := \{\mathtt{P}_1, \mathtt{P}_2, \mathtt{P}_3, \dots, \}$. As the notation hints, there can be an infinite number of such plans, so the approach of unioning plans won't work. Fortunately, another structure, $\mathtt{P}^*$, exists such that $\mathcal{L}^I(\mathtt{D}) = \mathcal{L}^I(\mathtt{P}^*) = \mathcal{L}^I(\mathtt{P}^\infty)$. Here $\mathtt{P}^*$, a finite p-graph, is called the *plan closure*. We will show the construction of $\mathtt{P}^*$ and prove $\mathcal{L}^I(\mathtt{P}^*) = \mathcal{L}^I(\mathtt{P}^\infty)$ below.

Case IV. When taking $\mathtt{D} = \mathtt{W}$ the executions are, again, intersected with $\mathcal{L}^I(\mathtt{D})$ but as they already came from $\mathcal{L}^I(\mathtt{W})$, this shows why the observer is the least informed in the hierarchy.

Now, the question is how to represent $\mathtt{P}^\infty$ with a p-graph $\mathtt{P}^*$ and show that they share the same language.

To start, we describe construction of $\mathtt{P}^*$. The initial step is to convert $\mathtt{W}$ to its state-determined form $\mathtt{W}' = \textsc{Sde}(\mathtt{W})$ (this is an operation described in [12, pp. 244–245], also provided as pseudocode in Algorithm 1). Then, to decide whether a vertex in $\mathtt{W}'$ exists in some plan, we iteratively color each vertex green, red, or gray. Being colored green means that the vertex exists in some plan, red means that the vertex does not exist in any plan, and gray indicates that its status has yet to be decided. To start with, we initially color the goal vertices green, and non-goal leaf vertices (with no edges to other vertices) red. Using the iconography of [12], we show action vertices as squares and observation vertices as circles. Then gray vertices of each type change their color by iterating the following steps:

- □ → 🟩: $\exists$ some action a reaching 🟢, which is not an initial state.
- □ → 🟥: $\forall$ action a reaching 🔴.
- ○ → 🟢: $\forall$ observation o reaching 🟩, which is not an initial state.
- ○ → 🔴: $\exists$ some observation o reaching 🟥.

The iteration ends when no vertex changes its color. The subgraph that consisting of only green vertices and their corresponding edges is $\mathtt{P}^*$. And $\mathtt{P}^*$ then contains only the vertices that exist in some plan leading to the goal states. For further detail of this algorithm for building $\mathtt{P}^*$, we refer the reader to Algorithm 3.

Next, we prove that the $\mathtt{P}^*$ constructed from this procedure has the same language as $\mathtt{P}^\infty$. The proof shows that any green vertex is on some plan, by showing that we we can construct a plan π, that will lead to a goal state within a finite number of steps form any such vertex.

Lemma 4.5. $\mathcal{L}^I(\mathtt{P}^*)$=$\mathcal{L}^I(\mathtt{P}^\infty)$.

Proof. $\supseteq$: For any $s = s_0 s_1 s_2 \dots s_k \in \mathcal{L}^I(\mathtt{P}^\infty)$, according to the definition of $\mathtt{P}^\infty$, s is in the execution of some plan $\mathtt{P}'$. Though s_k may not be a goal, using $\mathtt{P}'$, s can be extended: $\exists s' = s_0 s_1 \dots s_k t_0 t_1 \dots t_n \in \mathcal{L}^I(\mathtt{P}')$, $k > 0, n \geq 0$ to reach an element of V_{goal}. Then $\mathcal{V}_{\mathtt{P}'}(s')$ comprises vertices associated with those in $\mathtt{W}'$ marked green in V'_{goal}. And, tracing the execution s' on $\mathtt{P}'$ backwards on $\mathtt{W}'$, we find every vertex green back to a start vertex. But this means they are in $\mathtt{P}^*$, and hence $s' \in \mathcal{L}^I(\mathtt{P}^*)$, means $s \in \mathcal{L}^I(\mathtt{P}^*)$ as well.

$\subseteq$: For any execution $s = s_0 s_1 s_2 \dots s_k \in \mathcal{L}^I(\mathtt{P}^*)$, s reaches V'_{goal}, or s is a prefix of some execution reaching V'_{goal} in $\mathtt{W}'$. We show that there is a plan that can produce s. The execution s does not include enough information to describe a plan because: (1) it may not reach V'_{goal} itself, and (2) it gives an action after some observation that was revealed, but not every possible observation. To address this shortfall, we will capture some additional information during the construction of $\mathtt{P}^*$, which we save in π. This provides an action that makes some progress, for states that can result from other observations. Now, using s as a skeleton, construct plan where once a transition outside of s occurs, either owing to an unaccounted-for observation or having reached the end of s, the plan reverts to using the actions that π prescribes. (See Fig. 4.9 for a visual example.) This is always possible because states arrived at in $\mathtt{W}'$ under s are green. This implies that all states in $\mathtt{W}$ are also assured to reach a goal states. The resulting plan can produce s, so some plan produces s, hence $s \in \mathcal{L}^I(\mathtt{P}^\infty)$. □

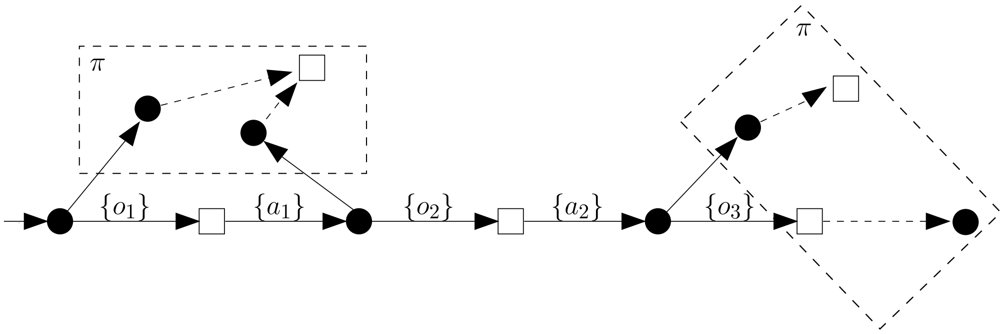


Figure 4.9: The construction of a plan generating execution s using π, computed as part of Algorithm 3, as an 'ambient' plan.

Thus, one may use $\mathtt{D} = \mathtt{P}^*$, for Case III in Section 4.3.1.

Now using the D as appropriate for each case, one may examine whether a given plan and disclosure policy solves the planning problem (i.e., achieves desired goals in the world) while meeting the stipulations on information communicated. Hence, we see that describing disclosed information via a p-graph D is in fact rather expressive. This section has also illustrated the benefits of being able to use both interaction language and graph presentation views of the same structure.

4.3.2 Construction of the finest observer

Next, we define the most capable observer in the particular sense of being equipped with disclosed information that enables it to track aspects most minutely.

Given the disclosed plan D as a prior knowledge, the observer builds estimates about the plan states and world states. As we may place stipulations on both plan states and world states, for privacy considerations, we are interested in the strongest observer—the one that maintains the tightest estimate for both disclosed plan states and world states.

Definition 4.16 (finest observer given D)**.** Given world graph W and the divulged plan D, an I-state graph $\widetilde{I}$ is a *finest observer* if for any I-state graph I, we have $\forall s \in \mathcal{L}^I(\mathtt{W})$, $\mathcal{W}^{\widetilde{I},\mathtt{D}}_{h(s)} \subseteq \mathcal{W}^{\mathtt{I},\mathtt{D}}_{h(s)}$ and $\mathcal{D}^{\widetilde{I},\mathtt{W}}_{h(s)} \subseteq \mathcal{D}^{\mathtt{I},\mathtt{W}}_{h(s)}$.

Above we have discussed checking whether a plan meets some stipulations (Section 4.2), but soon we will turn attention to finding suitable plans (Section 4.4). We will examine how one can

Algorithm 3: $\mathtt{P}^*\textsc{Construction}(\mathtt{W}, V_{\text{goal}})$

```
Initialize queues red, green, gray as empty
$\mathtt{W}' \leftarrow \textsc{Sde}(\mathtt{W})$, and initialize $V'_{\text{goal}}$ as the associated vertices of $V_{\text{goal}}$
Initialize plan $\pi$ as empty
for $v \in V(\mathtt{W}')$ do
    if $v \in V'_{\text{goal}}$ then
        green.append($v$)
    else if $v$ has no edges to other vertices then
        red.append($v$)
    else
        gray.append($v$)
    end if
end for
$Q$.extend(InNeighbor(red $\cup$ green)$\setminus$(red $\cup$ green))
while $Q$ not empty do
    $v \leftarrow Q$.pop
    flag $\leftarrow$True
    if $v$ is a ⚪ then
        if one of its outgoing neighbors is 🟥 then
            red.append($v$)
        else if all of its outgoing neighbors are 🟩 then
            green.append($v$)
        else
            flag $\leftarrow$False
        end if
    else if $v$ is a ⬜ then
        if one of its outgoing neighbors under label $a$ is 🟢 then
            green.append($v$) and $\pi[v] = a$
        else if all of its outgoing neighbors are 🔴 then
            red.append($v$)
        else
            flag $\leftarrow$False
        end if
    end if
    if flag then
        $Q$.extend(InNeighbor($v$)$\setminus\{$red $\cup$ green$\}$)
    end if
end while
$\mathtt{P}^* \leftarrow$ subgraph($\mathtt{W}'$, green)
return $\mathtt{P}^*$ (and also $\pi$, if desired)
```

seek and then disclose that sought plan, so that it will satisfy some given stipulations. There is no natural way for stipulations in such cases to restrict knowledge of plan states, because formulae cannot name plan states which are not yet known. Hence, we must restrict our interest is in the strongest observer that (always) maintains the tightest estimate for the world states.

Definition 4.17 (finest observer given $\mathtt{P}$)**.** Given world graph $\mathtt{W}$ and robot's plan $\mathtt{P}$, an I-state graph $\widetilde{I}$ is a *finest observer* if for any I-state graph $\mathtt{I}$, we have $\forall s \in \mathcal{L}^I(\mathtt{W})$, $\mathcal{W}^{\widetilde{I},\mathtt{P}}_{h(s)} \subseteq \mathcal{W}^{\mathtt{I},\mathtt{P}}_{h(s)}$.

Any observer attempting to keep more states that $V(\mathtt{W})$ is merely tracking repetitions and, given the form of the stipulations, this grants the observer no additional capabilities. Hence, the finest observer is well defined and also conveniently represented:

Lemma 4.6. Three closely related claims:

- For every pair of executions $s_1, s_2 \in h^{-1}[\mathbb{S}_B^{h\langle \mathtt{W}\rangle}] \cap \mathcal{L}^I(\mathtt{W})$, then we have either $h(s_1) = h(s_2)$ or $\mathcal{V}_{\mathtt{W}}(s_1) = \mathcal{V}_{\mathtt{W}}(s_2)$.

- For every pair of executions $s_1, s_2 \in h^{-1}[\mathbb{S}_B^{h\langle \mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{D})$, then we have either $h(s_1) = h(s_2)$ or $\mathcal{V}_{\mathtt{D}}(s_1) = \mathcal{V}_{\mathtt{D}}(s_2)$.

- For every pair of executions $s_1, s_2 \in h^{-1}[\mathbb{S}_B^{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{W}) \cap \mathcal{L}^I(\mathtt{D})$, then we have either $\underline{h(s_1) = h(s_2)}$, or we have $\underline{\mathcal{V}_{\mathtt{W}}(s_1) = \mathcal{V}_{\mathtt{W}}(s_2) \text{ and } \mathcal{V}_{\mathtt{D}}(s_1) = \mathcal{V}_{\mathtt{D}}(s_2)}$.

Proof. For $\forall s_1, s_2 \in h^{-1}[\mathbb{S}_B^{h\langle \mathtt{W}\rangle}]$, we have $\mathcal{V}_{h\langle \mathtt{W}\rangle}(h(s_1)) = \mathcal{V}_{h\langle \mathtt{W}\rangle}(h(s_2)) = B$. Suppose $h(s_1) \neq h(s_2)$ and $\mathcal{V}_{\mathtt{W}}(s_1) \neq \mathcal{V}_{\mathtt{W}}(s_2)$. Let $w_1 \in \mathcal{V}_{\mathtt{W}}(s_1)$ and $w_1 \notin \mathcal{V}_{\mathtt{W}}(s_2)$. Then we have $w_1 \in \mathcal{V}_{h\langle \mathtt{W}\rangle}(h(s_1))$. In order to satisfy $\mathcal{V}_{h\langle \mathtt{W}\rangle}(h(s_1)) = \mathcal{V}_{h\langle \mathtt{W}\rangle}(h(s_2))$, we need to find another execution $s' \in h^{-1}[\mathbb{S}_B^{h\langle \mathtt{W}\rangle}]$ such that $h(s') = h(s_2)$ and $w_1 \in \mathcal{V}_{\mathtt{W}}(s')$, which contradicts with the condition $s_1, s_2 \in h^{-1}[\mathbb{S}_B^{h\langle \mathtt{W}\rangle}]$.

Similarly, one proves that $\forall s_1, s_2 \in h^{-1}[\mathbb{S}_B^{h\langle \mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{D})$, then we have either $h(s_1) = h(s_2)$ or $\mathcal{V}_{\mathtt{D}}(s_1) = \mathcal{V}_{\mathtt{D}}(s_2)$.

Now, $\forall s_1, s_2 \in h^{-1}[\mathbb{S}_B^{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{W}) \cap \mathcal{L}^I(\mathtt{D})$, we have $s_1, s_2 \in h^{-1}[\mathbb{S}_{B'}^{h\langle \mathtt{W}\rangle}] \cap \mathcal{L}^I(\mathtt{W})$ and $s_1, s_2 \in h^{-1}[\mathbb{S}_{B''}^{h\langle \mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{D})$. Then we have either $h(s_1) = h(s_2)$ or, otherwise, both $\mathcal{V}_{\mathtt{W}}(s_1) = \mathcal{V}_{\mathtt{W}}(s_2)$ and $\mathcal{V}_{\mathtt{D}}(s_1) = \mathcal{V}_{\mathtt{D}}(s_2)$. □

Lemma 4.7. $h\langle\mathtt{W}\rangle$ is a finest observer given $\mathtt{P}$.

Proof. This lemma will be proved by showing that $\forall s \in \mathcal{L}^I(\mathtt{W})$, $B = \mathcal{V}_{h\langle\mathtt{W}\rangle}(h(s))$ and $B' = \mathcal{V}_{\mathtt{I}}(h(s))$, we have $\mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}} \subseteq \mathcal{W}_{B'}^{\mathtt{I},\mathtt{P}}$.

If $\mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}}$ contains only one world state w, then $\exists s \in h^{-1}[\mathbb{S}_B^{h\langle\mathtt{W}\rangle}] \cap \mathcal{L}^I(\mathtt{P}) \cap \mathcal{S}_w^{\mathtt{W}}$. We also have $s \in h^{-1}[\mathbb{S}_{B'}^{\mathtt{I}}] \cap \mathcal{L}^I(\mathtt{P}) \cap \mathcal{S}_w^{\mathtt{W}}$ for some $B' \subseteq V(\mathtt{I})$. Hence, w is also contained in $\mathcal{W}_{B'}^{\mathtt{I},\mathtt{P}}$.

If $\mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}}$ contains at least two world state w_1, w_2, then $\exists s_1 \in h[\mathbb{S}_B^{h\langle\mathtt{W}\rangle}] \cap \mathcal{L}^I(\mathtt{P}) \cap \mathcal{S}_{w_1}^{\mathtt{W}}$ and $s_2 \in h[\mathbb{S}_B^{h\langle\mathtt{W}\rangle}] \cap \mathcal{L}^I(\mathtt{P}) \cap \mathcal{S}_{w_2}^{\mathtt{W}}$. According to Lemma 4.6, $h(s_1) = h(s_2)$ or $\mathcal{V}_{\mathtt{W}}(s_1) = \mathcal{V}_{\mathtt{W}}(s_2)$. If $h(s_1) = h(s_2)$, then it is trivial that $s_1, s_2 \in h^{-1}[\mathbb{S}_{B'}^{\mathtt{I}}]$. We have $s_1 \in h[\mathbb{S}_{B'}^{\mathtt{I}}] \cap \mathcal{L}^I(\mathtt{P}) \cap \mathcal{S}_{w_1}^{\mathtt{W}}$ and $s_2 \in h[\mathbb{S}_{B'}^{\mathtt{I}}] \cap \mathcal{L}^I(\mathtt{P}) \cap \mathcal{S}_{w_2}^{\mathtt{W}}$. Hence, $w_1, w_2 \in \mathcal{W}_{B'}^{\mathtt{I},\mathtt{P}}$. If $\mathcal{V}_{\mathtt{W}}(s_1) = \mathcal{V}_{\mathtt{W}}(s_2)$, then $s_1, s_2 \in h[\mathbb{S}_{B'}^{\mathtt{I}}] \cap \mathcal{L}^I(\mathtt{P}) \cap \mathbb{S}_{\{w_1,w_2\}}^{\mathtt{W}}$. Hence, $w_1, w_2 \in \mathcal{W}_{B'}^{I}$ also holds.

Therefore, $\mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}} \subseteq \mathcal{W}_{B'}^{\mathtt{I},\mathtt{P}}$. Hence, $h\langle\mathtt{W}\rangle$ is a finest observer. □

Lemma 4.8. $h\langle\mathtt{W} \otimes \mathtt{D}\rangle$ is a finest observer given $\mathtt{D}$.

Proof. This lemma will be proved by showing that $\forall s \in \mathcal{L}^I(\mathtt{W}) \cap \mathcal{L}^I(\mathtt{D})$, $B = \mathcal{V}_{h\langle\mathtt{W}\otimes\mathtt{D}\rangle}(h(s))$, $B' = \mathcal{V}_{\mathtt{I}}(h(s))$, such that $\mathcal{W}_B^{h\langle\mathtt{W}\otimes\mathtt{D}\rangle,\mathtt{D}} \subseteq \mathcal{W}_{B'}^{\mathtt{I},\mathtt{D}}$.

If there exists an execution $s \in h^{-1}[\mathbb{S}_B^{h\langle\mathtt{W}\otimes\mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_{w_1}^{\mathtt{W}}$, such that $\mathcal{W}_B^{h\langle\mathtt{W}\otimes\mathtt{D}\rangle,\mathtt{D}}$ contains only one world state w. Then we also have $s \in h^{-1}[\mathbb{S}_{B'}^{\mathtt{I}}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_{w_1}^{\mathtt{W}}$. Hence, $w_1 \in \mathcal{W}_{B'}^{\mathtt{I},\mathtt{D}}$.

When there exists an image execution reaching B and B' in $h\langle\mathtt{W}\otimes\mathtt{D}\rangle$ and $\mathtt{I}$, such that $\mathcal{W}_B^{\mathrm{h}\langle\mathtt{W}\otimes\mathtt{D}\rangle,\mathtt{D}}$ contains at least two world states w_1 and w_2, we have $\exists s_1 \in h^{-1}[\mathbb{S}_B^{h\langle\mathtt{W}\otimes\mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_{w_1}^{\mathtt{W}}, \exists s_2 \in h^{-1}[\mathbb{S}_B^{h\langle\mathtt{W}\otimes\mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_{w_2}^{\mathtt{W}}$. According to Lemma 4.6, either $h(s_1) = h(s_2)$ or $\mathcal{V}_{\mathtt{W}}(s_1) = \mathcal{V}_{\mathtt{W}}(s_2)$. Next, we will show that if $\exists s_1 \in h^{-1}[\mathbb{S}_B^{h\langle\mathtt{W}\otimes\mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_{w_1}^{\mathtt{W}}, \exists s_2 \in h^{-1}[\mathbb{S}_B^{h\langle\mathtt{W}\otimes\mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_{w_2}^{\mathtt{W}}$ such that either $h(s_1) = h(s_2)$ or $\mathcal{V}_{\mathtt{W}}(s_1) = \mathcal{V}_{\mathtt{W}}(s_2)$, then $w_1, w_2 \in \mathcal{W}_{B'}^{\mathtt{I},\mathtt{D}}$. If $h(s_1) = h(s_2)$, then it is trivial that $s_1, s_2 \in h^{-1}[\mathbb{S}_{B'}^{\mathtt{I}}]$. And we have $s_1 \in h^{-1}[\mathbb{S}_{B'}^{\mathtt{I}}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_{w_1}^{\mathtt{W}}$ and $s_2 \in h^{-1}[\mathbb{S}_{B'}^{\mathtt{I}}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_{w_2}^{\mathtt{W}}$. Hence, $w_1, w_2 \in \mathcal{W}_{B'}^{\mathtt{I},\mathtt{D}}$. If $\mathcal{V}_{\mathtt{W}}(s_1) = \mathcal{V}_{\mathtt{W}}(s_2)$, then $s_1, s_2 \in h^{-1}[\mathbb{S}_{B'}^{\mathtt{I}}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathbb{S}_{\{w_1,w_2\}}^{\mathtt{W}}$. Hence, $w_1, w_2 \in \mathcal{W}_{B'}^{\mathtt{I},\mathtt{D}}$ also holds when at least two world states are consistent with the finest observer's belief. Hence, $\mathcal{W}_B^{h\langle\mathtt{W}\otimes\mathtt{D}\rangle,\mathtt{D}} \subseteq \mathcal{W}_{B'}^{\mathtt{I},\mathtt{D}}$ whenever there is one or more estimated world states in $\mathcal{W}_B^{h\langle\mathtt{W}\otimes\mathtt{D}\rangle,\mathtt{D}}$. □

4.4 Searching for plans and disclosure policy: the SEEK problems

Naturally, the question of most interest is how to find plans and/or disclosure policies, not merely how to verify them. We formulate this problem in three varieties:

Problem: $\textsc{Seek}_{\boldsymbol{x}}\big((\mathtt{W}, V_{\text{goal}}), \boldsymbol{x}, (\widetilde{I}, \mathtt{D}), h, \boldsymbol{\Phi}\big)$

$\textsc{Seek}_{\boldsymbol{\lambda}}\big((\mathtt{W}, V_{\text{goal}}), \mathtt{P}, (\widetilde{I}, \mathtt{D}), \boldsymbol{\lambda}, \boldsymbol{\Phi}\big)$

$\textsc{Seek}_{\boldsymbol{x},\boldsymbol{\lambda}}\big((\mathtt{W}, V_{\text{goal}}), \boldsymbol{x}, (\widetilde{I}, \boldsymbol{x}), \boldsymbol{\lambda}, \boldsymbol{\Phi}\big)$

Vars. to solve for:

$\boldsymbol{x}$ is a plan

$\boldsymbol{\lambda}$ is a label map

Input: A planning problem $(\mathtt{W}, V_{\text{goal}})$, a finest observer $\widetilde{I}$, a divulged plan p-graph $\mathtt{D}$, information disclosure policy h and information stipulation $\boldsymbol{\Phi}$.

Output: A plan $\boldsymbol{x} = (\mathtt{P}, V_{\text{term}})$ and/or label map $\boldsymbol{\lambda} = h$ such that plan $(\mathtt{P}, V_{\text{term}})$ solves the problem $(\mathtt{W}, V_{\text{goal}})$, and $\forall s \in \mathcal{L}^I(W^\dagger) \cap \mathcal{L}^I(\mathtt{P}), B = \mathcal{V}_{\mathtt{I}}(h(s))$, the information stipulation $\boldsymbol{\Phi}$ is always evaluated as True on $\mathcal{W}_B^{\mathtt{I},\mathtt{D}}$ (i.e. $\text{satfd}(B, \boldsymbol{\Phi}) = True$), else False.

Of the three versions of SEEK, the first searches for a plan, the second for a label map, and the third for both, jointly. We consider each in turn.

4.4.1 Finding a plan given some predetermined $\mathtt{D}$

For $\textsc{Seek}_{\boldsymbol{x}}$, first we must consider the search space of plans. Prior work [12] showed that, although planning problems can have stranger solutions than people usually contemplate, there is a core of well-structured plans (called homomorphic solutions) that suffice to determine solvability. As an example, there may exist plans which loop around the environment before achieving the goal, but, they showed that in seeking plans, one need only consider plans that short-circuit the loops.

The situation is rather different when a plan must satisfy more than mere goal achievement: information stipulations may actually *require* a plan to loop in order to ensure that the disclosed stream of events is appropriate for the observer's eyes. (A concrete example appears in Fig. 4.15(c).) The argument in [12] needs modification for our problem—a different construction can save the result even under disclosure constraints. This fact is key to be able to implement a solution.

In this paper, without loss of generality, we focus on finding plans in state-determined form. Next, we will examine the solution space closely.

Definition 4.18. A plan $\mathtt{P}$ is *congruent* on the world graph $\mathtt{W}$, if and only if for every pair of executions $s_1, s_2 \in \mathcal{L}^I(\mathtt{P})$ we have $s_1 \underset{\mathtt{P}}{\sim} s_2 \implies s_1 \underset{\mathtt{W}}{\sim} s_2$.

In other words, a plan that respects the equivalence classes of the world graph is defined as a congruent plan. (The definition of congruent plan is a generalization of homomorphic solution (see Definition 6.7 in [12]), by considering a non-deterministic world.) Next, our search space may be narrowed further still.

Lemma 4.9. Given any plan $(\mathtt{P}, V_{\text{term}})$, there exists a plan $(\mathtt{P}', V'_{\text{term}})$ that is congruent on the world graph $\mathtt{W}$ and $\mathcal{L}^I(\mathtt{P}') = \mathcal{L}^I(\mathtt{P})$.

Proof. We give a construction from $\mathtt{P}$ of $\mathtt{P}'$ as a tree, and show that it meets the conditions. To construct $\mathtt{P}'$, perform a BFS on $\mathtt{P}$. Starting from $V_0(\mathtt{P})$, build a starting vertex v_0 in $\mathtt{P}'$, keep a correspondence between it and $V_0(\mathtt{P})$. Mark v_0 as unexpanded. Now, for every unexpanded vertex v in $\mathtt{P}'$, mark the set of all outgoing labels for its corresponding vertices in $\mathtt{P}$ as L_v, create a new vertex v' in $\mathtt{P}'$ for each label $l \in L_v$, build an edge from v to v' with label l in $\mathtt{P}'$, and mark it as expanded. Repeat this process until all vertices in $\mathtt{P}'$ have been expanded. Mark the vertices corresponding to vertices in V_{term} as V'_{term}. In the new plan $(\mathtt{P}', V'_{\text{term}})$, no two executions reach the same vertex. That is, $\forall s_1, s_2 \in \mathcal{L}^I(\mathtt{P}'), s_1 \underset{\mathtt{P}'}{\not\sim} s_2$. Hence, $\mathtt{P}'$ is congruent on $\mathtt{W}$. In addition, since no new executions are introduced and no executions in $\mathtt{P}$ are eliminated during the construction of $\mathtt{P}'$, we have $\mathcal{L}^I(\mathtt{P}') = \mathcal{L}^I(\mathtt{P})$. □

Theorem 4.2. For problem $\textsc{Seek}_{\boldsymbol{x}}\big((\mathtt{W}, V_{\text{goal}}), \boldsymbol{x}, (\mathtt{I}, \mathtt{D}), h, \boldsymbol{\Phi}\big)$, if there exists a solution $(\mathtt{P}, V_{\text{term}})$, then there exists a solution $(\mathtt{P}', V'_{\text{term}})$ that is both c-bounded and congruent on $\mathtt{W}$, where $c = |V(\mathtt{W})| \cdot |V(\mathtt{D})| \cdot |V(\mathtt{I})|$.

Proof. Suppose $\textsc{Seek}_{\boldsymbol{x}}$ has a solution $(\mathtt{P}, V_{\text{term}})$. Then the existence of a solution $(\mathtt{P}', V'_{\text{term}})$ which is congruent on $\mathtt{W}$ is implied by Lemma 4.9. Moreover, we have $\textsc{Check}\big((\mathtt{W}, V_{\text{goal}}), (\mathtt{P}, V_{\text{term}}), \mathtt{D}, \mathtt{I}, h, \boldsymbol{\Phi}\big) \implies \textsc{Check}\big((\mathtt{W}, V_{\text{goal}}), (\mathtt{P}', V'_{\text{term}}), \mathtt{D}, \mathtt{I}, h, \boldsymbol{\Phi}\big)$, following from two observations:

($i.$) if $(\mathtt{P}, V_{\text{term}})$ solves $(\mathtt{W}, V_{\text{goal}})$ then the means of construction ensures $(\mathtt{P}', V'_{\text{term}})$ does as well, and

($ii.$) in checking $\mathbf{\Phi}$, the set of estimated world states $\mathcal{W}^{\mathtt{I},\mathtt{D}}_{\{v\}}$ does not change for each vertex $v \in V(\textsc{Sde}(\mathtt{I}))$, since the triple graph is independent of the plan to be searched. The set of I-states to be evaluated by $\mathbf{\Phi}$ in $\textsc{Sde}(\mathtt{I})$ is $\cup_{s' \in h[\mathcal{L}^I(\mathtt{P}) \cap \mathcal{L}^I(\mathtt{W})]} \mathcal{V}_{\textsc{Sde}(\mathtt{I})}(s')$. Since $\mathcal{L}^I(\mathtt{P}) = \mathcal{L}^I(\mathtt{P}')$, the set of I-states to be evaluated is no altered and the truth of $\mathbf{\Phi}$ along the plan is preserved.

The final step is to prove that if there exists a congruent solution $(\mathtt{P}', V'_{\text{term}})$, then there exits a solution $(\mathtt{P}'', V''_{term})$ that is c-bounded. First, build a product graph $\mathtt{T}$ of $\mathtt{W}$, $\mathtt{D}$, and $h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle$, with vertex set $V(\mathtt{W}) \times V(\mathtt{D}) \times V(h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle)$. Then trace every execution s in $\mathtt{P}'$ on $\mathtt{T}$. If s visits the same vertex $(v^{\mathtt{W}}, v^{\mathtt{D}}, v^{h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle})$ multiple times, then $v^{\mathtt{W}}$, $v^{\mathtt{D}}$, and $v^{h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle}$ have to be action vertices, for otherwise $\mathtt{P}'$ can loop forever and is not a solution (since $\mathtt{P}'$ is finite on $\mathtt{W}$). Next, record the action taken at the last visit of $(v^{\mathtt{W}}, v^{\mathtt{P}}, v^{h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle})$ as a_{last}. Finally, build a new plan $(\mathtt{P}'', V'_{\text{term}})$ by bypassing unnecessary transitions on $\mathtt{P}'$ as follows. For each vertex $(v^{\mathtt{W}}, v^{\mathtt{P}}, v^{h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle})$ that is visited multiple times, $\mathtt{P}''$ takes action a_{last} when $(v^{\mathtt{W}}, v^{\mathtt{P}}, v^{h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle})$ is first visited. $\mathtt{P}''$ terminates at the goal states without violating any stipulations, since it takes a shortcut in the executions of $\mathtt{P}'$ but—crucially—without visiting any new observer I-states. In addition, $\mathtt{P}''$ will visit each vertex in $\mathtt{T}$ at most once, and the maximum length of its executions is $|V(\mathtt{W})| \times |V(\mathtt{D})| \times |V(h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle)|$. Since $\mathtt{P}''$ preserves the structure of $\mathtt{P}'$ during this construction, $\mathtt{P}''$ is also congruent. □

The intuition, and the underlying reason for considering congruent plans, is that modifying the plan will not affect the stipulations if the underlying languages are preserved. The bound on the length then takes this further, modifying the language by truncating long executions in the triple graph, thereby shortcutting visits to I-states that do not affect goal achievement.

Accordingly, it suffices to look for congruent plans in the (very specific) form of trees, since any plan has a counterpart that is congruent and in the form of a tree (for additional detail, refer to Lemma 4.9). Theorem 4.2 states that the depth of the tree is at most $c = |V(\mathtt{W})| \cdot |V(\mathtt{D})| \cdot |V(h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle)|$. Therefore, we can limit the search space to trees of a specific bounded depth. To search for a c-bounded solution, first we mark the vertex $(v^{\mathtt{W}}, v^{\mathtt{D}}, v^{h^{-1}\langle\textsc{Sde}(\mathtt{I})\rangle})$ as: (i) a goal state if $v^{\mathtt{W}}$ is a goal state in the world graph; (ii) as satisfying $\mathbf{\Phi}$ when all the world states and plan states appearing together with $v^{h^{-1}\langle \textsc{Sde}(\mathtt{I})\rangle}$ together satisfy $\mathbf{\Phi}$. Then we will conduct an AND–OR search [68] on the triple graph to search for a tree with bounded depth, in which:

- each action vertex serves as an OR node, and an action should be chosen for each action vertex in the tree such that it will eventually terminate at the goal states and all the vertices satisfy $\mathbf{\Phi}$ along the way;
- each observation vertex is treated as an AND node, and for each of its child vertices, there exists an action choice that satisfies $\mathbf{\Phi}$ and eventually terminate at the goal states.

4.4.2 Finding a label map given some predetermined $\mathtt{D}$

To solve SEEK$_{\boldsymbol{\lambda}}$, the key idea is to treat the label map as a partitioning where the events sharing the same image are grouped together while the ones with different images form parts of groups that must different. The label map is then a collection of non-overlapping groups of events: a partition of the set of events. Like the incremental procedure used to obtain a plan, as we search an incremental form of label map must be kept. Specifically, a partition for a subset of actions and observations seen so far, which we term a partial label map (also, below, a partial partition), is built up incrementally. In order to visualize the search, we present an observer's belief tree for SEEK$_{\boldsymbol{\lambda}}$, which encodes the observer's belief transitions under different label map choices. The objective is to search for a subtree, which consists of non-conflicting partial label maps.

Firstly, we will use the product graph $h\langle \mathtt{W} \otimes \mathtt{D}\rangle$ as the finest observer. Then the estimated world states and plan states are formalized with the following theorem:

Theorem 4.3. Given any $s \in \mathcal{L}^I(\mathtt{W}) \cap \mathcal{L}^I(\mathtt{D})$, the set of I-states reached by $h(s)$ in $h\langle \mathtt{W}\rangle$ is $B = \mathcal{V}_{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}(h(s))$. Then $\mathcal{W}_B^{h\langle \mathtt{W}\otimes\mathtt{D}\rangle,\mathtt{D}} = \pi_{\mathtt{W}}(\mathcal{V}_{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}(h(s)))$ and $\mathcal{D}_B^{h\langle \mathtt{W}\otimes\mathtt{D}\rangle,\mathtt{W}} = \pi_{\mathtt{D}}(\mathcal{V}_{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}(h(s)))$, where $\pi_{\mathtt{W}}(V)$ takes the first elements from tuples in V and gives a a set of world states, and $\pi_{\mathtt{D}}(V)$ takes the second elements from tuples in V and gives a a set of plan states.

Proof. According to the definition, $\mathcal{W}_B^{h\langle \mathtt{W}\otimes\mathtt{D}\rangle,\mathtt{D}} = \{w \in V(h\langle \mathtt{W} \otimes \mathtt{D}\rangle) | h^{-1}[\mathbb{S}_B^{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{D}) \cap \mathcal{S}_w^{\mathtt{W}} \neq \varnothing\} = \pi_{\mathtt{W}}(\cup_{s\in h^{-1}[\mathbb{S}_B^{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}]\cap\mathcal{L}^I(\mathtt{W})\cap\mathcal{L}^I(\mathtt{D})}\mathcal{V}_{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}(h(s)))$. Since $\forall s_1, s_2 \in h^{-1}[\mathbb{S}_B^{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}] \cap \mathcal{L}^I(\mathtt{W}) \cap \mathcal{L}^I(\mathtt{D})$, we have either $\mathcal{V}_{\mathtt{W}}(s_1) = \mathcal{V}_{\mathtt{W}}(s_2)$ or $h(s_1) = h(s_2)$. If $\mathcal{V}_{\mathtt{W}}(s_1) = \mathcal{V}_{\mathtt{W}}(s_2)$, then need to consider the set of executions whose image are equal and defined as s'. Since $s' \in \mathbb{S}_B^{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}$, s' reaches and reaches only vertices in B. Therefore, $\mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{D}}$ is the set of all world states included in B. Hence, $\mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{D}} = \pi_{\mathtt{W}}(\mathcal{V}_{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}(h(s)))$. Similarly, $\mathcal{D}_B^{h\langle \mathtt{W}\rangle,\mathtt{W}} = \pi_{\mathtt{D}}(\mathcal{V}_{h\langle \mathtt{W}\otimes\mathtt{D}\rangle}(h(s)))$ □

Theorem 4.3 provides a shortcut to find the estimated world states and plan states of any set of observer I-states $B \subseteq V(h\langle \mathtt{W}\otimes\mathtt{D}\rangle)$, by projecting to its first $\pi_{\mathtt{W}}(B)$ and second components $\pi_{\mathtt{D}}(B)$. The observer I-states B satisfies the stipulation $\mathbf{\Phi}$, iff the stipulation $\mathbf{\Phi}$ is satisfied on both $\pi_{\mathtt{W}}(B)$ and $\pi_{\mathtt{D}}(B)$.

Note that not all I-states B will be visited by robot's plan $\mathtt{P}$. We only want to make sure that the I-states that are reached by strings in $\mathtt{P}\otimes\mathtt{W}$ satisfy the stipulations. To find those I-states, we will construct a product graph of $\mathtt{W}\otimes\mathtt{D}$ and $\mathtt{P}$. The vertex in $\mathtt{W}\otimes\mathtt{D}$ is marked as reachable by the plan $\mathtt{P}$ if it is paired with some vertex in $\mathtt{P}$ in the product graph. Then for any I-states $B \subseteq \mathtt{W}\otimes\mathtt{D}$, B is reachable if there exists a vertex $v \in B$ is reachable by $\mathtt{P}$. This can be done in a preprocessing step, without the label map.

A brief aside is necessary to describe requisite data structures. First, we represent the label map h as a partition over all events according to the equivalence relation induced under h. Given any label map h, its equivalence relation is defined as follows: $h(\ell_1) = h(\ell_2) \Leftrightarrow \ell_1 \underset{h}{\sim} \ell_2$. Using $\underset{h}{\sim}$, we may partition K of the events. (We will use both $\underset{K}{\sim}$ and $\underset{h}{\sim}$ to denote their equivalence relation, the former being more representationally explicit in bearing a collection of sets.) Next, we claim that the partition is equivalent to the label map function in terms of the estimation of world states and plan states for any given I-states. In the formula to estimate the world and plan states (see Definition 4.14 and 4.15), the key to using the label map is to find the strings that share the same image, which is exactly what is distinguished up to partitioning. Hence, searching for a partition of the events is identical to searching for a label map function. For a partition of only some events in p-graph $\mathtt{W}\otimes\mathtt{D}$, we term these partial partitions. The partial partitions describe partition choices for the events that are mentioned, but place no constraints for the events that have not appeared. We say that a partial partition K_1 conflicts with K_2 if $\ell_1 \underset{K_1}{\sim} \ell_2$ and $\ell_1 \underset{K_2}{\not\sim} \ell_2$.

Now, in searching for a label map, we expand the graph $\mathtt{W} \otimes \mathtt{D}$ to include choices of label maps. This yields a richer structure that we call an observer's belief tree. In the observer's belief tree shown in Fig. 4.10, each vertex is interpreted as a subset of world states. This tree, being a sort of generalized or enhanced AND–OR tree, captures the observer's belief under different choices of label map. It may be constructed incrementally as follows. Starting from $\mathcal{B}^0 = V_0(\mathtt{W} \otimes \mathtt{D})$, we expand an OR node, where each outgoing edge leads to a partition for all outgoing events at vertices in $\mathcal{B}^0$ and a single partition should be chosen. During the expansion, we will avoid expanding the partition which conflicts with its ancestor choices, since doing so must produce an invalid label map choice. Given a non-conflicting partition $\mathbb{P}_i = \{G_1, G_2 \dots\}$, we will treat it as an AND node and each outgoing edge bearing a set of events $G_j \in \mathbb{P}_i$ in the partition. Following the edge with events G_j, the states in $\mathcal{B}^0$ transition to $\mathcal{B}' = \{v' \in V(\mathtt{W} \otimes \mathtt{D}) | v \in \mathcal{B}^0, \ell' \in G_j, \textsc{TransTo}(v \xrightarrow{\ell'} v')^{\mathtt{W} \otimes \mathtt{D}}\}$. For each belief vertex in the tree, if the set of vertices in $\mathcal{B}'$ have been visited before, then we do not expand it since the label map choice has already been made for this belief, which makes this structure a tree. In addition, if the belief state $\mathcal{B}'$ is reached by plan $\mathtt{P}$ and violates the stipulations, then we will mark it as violating the stipulation. Otherwise, we will mark it as satisfying the stipulation.

Now, a label map choice will give a subtree of the observer's belief tree, where:

- there is only a single outgoing edge for each OR node;
- all outgoing edges at a chosen AND node must be included;
- all belief vertices in the subtree must satisfy the stipulation (if reached, then the belief vertex satisfies stipulation $\mathbf{\Phi}$).

There is only a single outgoing edge for each OR node, since we choose a single partial partition for its outgoing events. To be ready to receive all possible observations, all outgoing edges should be included in the subtree for a chosen AND node. Stipulations should hold at the belief vertex that can be generated by some execution from the plan.

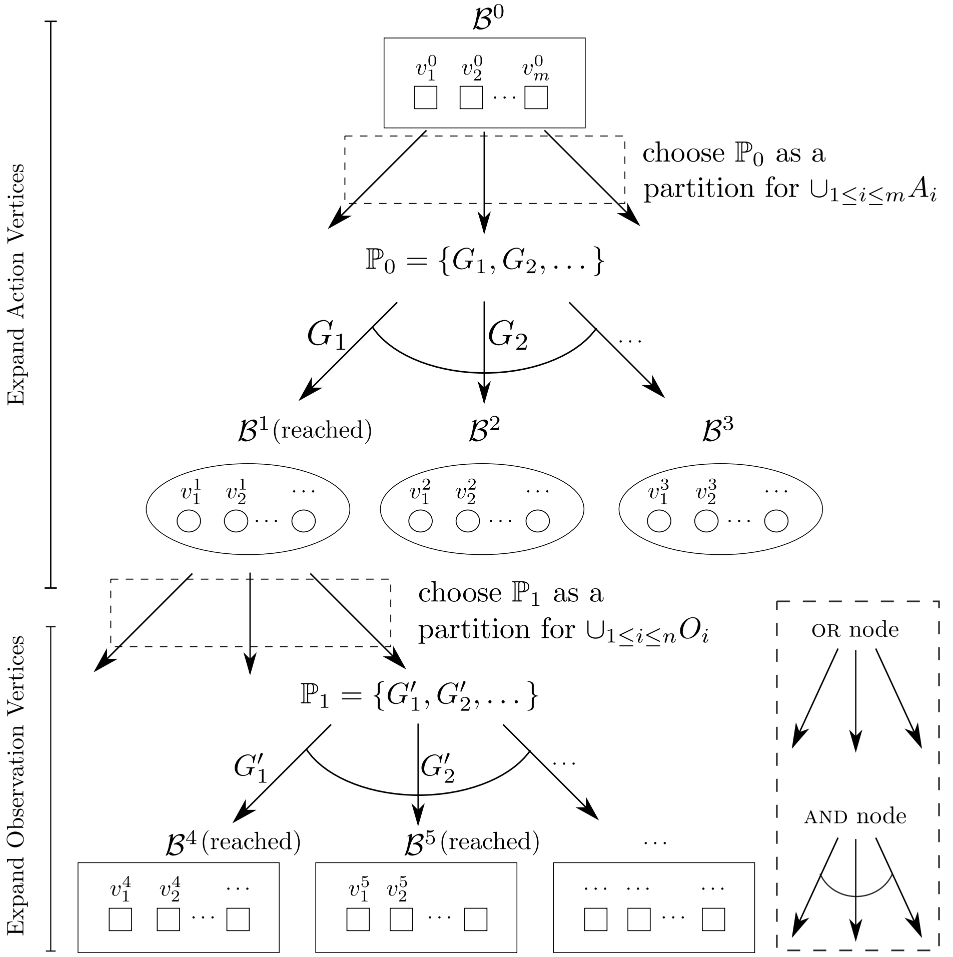

$\mathcal{B}^0$
v_1^0 v_2^0 v_m^0
choose $\mathbb{P}_0$ as a partition for $\cup_{1\leq i\leq m} A_i$
$\mathbb{P}_0 = \{G_1, G_2, \dots\}$
G_1
G_2
$\mathcal{B}^1$ (reached)
$\mathcal{B}^2$
$\mathcal{B}^3$
Expand Action Vertices
choose $\mathbb{P}_1$ as a partition for $\cup_{1\leq i\leq n} O_i$
$\mathbb{P}_1 = \{G_1', G_2', \dots\}$
G_1'
G_2'
$\mathcal{B}^4$ (reached)
$\mathcal{B}^5$ (reached)
Expand Observation Vertices
OR node
AND node

Figure 4.10: The observer's belief tree for SEEK$_{\lambda}$. For a set of action vertices comprising a vertex $\mathcal{B}^0$ a tier of OR nodes and another tier of AND nodes are generated. The first encodes each partition choice $\mathbb{P}_i$ of values $\{\mathtt{G}_1, \mathtt{G}_2, \dots\}$, (i.e., partial label maps). A given partition is expanded as an AND node with each outgoing edge bearing a group of events sharing the same image under the partial label map. Observation vertex $\mathcal{B}^1$ are expanded in the same way.

For example, a valid subtree for Fig. 4.11 can be found by choosing ①②⑤, as long as these chosen partial partitions do not conflict with each other.

Obtaining a label map involves choices across of multiple partial partitions in the AND–OR tree. These partial partitions should not conflict with each other, otherwise, they do not form a valid partition. There are two possible conflicts: conflicts between a partial partition and its ancestor choices (e.g., $\mathbb{P}_0$ conflicts with $\mathbb{P}_1$ in Fig. 4.11); conflicts between partitions in different subtrees (e.g., $\mathbb{P}_1$ conflicts with $\mathbb{P}_2$). The first conflict can be solved by passing the choices we

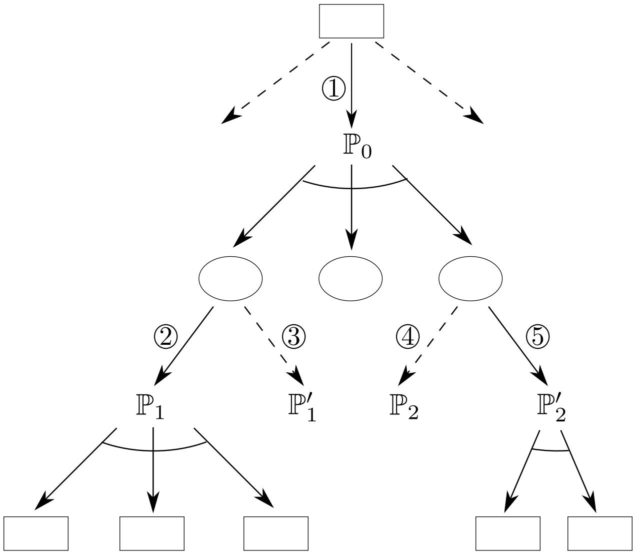


Figure 4.11: A valid subtree is found from the observer's belief tree by choosing the set of non-conflict partial partitions ①②⑤.

have committed to downwards and only choosing non-conflicting options below. The second type of conflict prevents us from dividing the problem into strictly independent sub-problems on the basis to the AND–OR tree structure. Thus, we need to be able to check conflicts across choices in different subtrees and retain sufficient state to backtrack on those choices when needed.

4.4.3 Search for a label map and a plan, with the same plan to be disclosed

It is not merely the joint search that makes this, the third problem, more interesting. It actually sidesteps a subtle issue not yet discussed. Consider a particularly cunning observer observing a robot but who also knows that the robot does not wish to violate some stipulation—this observer may take advantage of this awareness. The observer could reason that any execution in the disclosed plan that could violate the stipulations will never be executed by the robot. Hence, the observer might eliminate this execution from consideration, thereby pruning elements from the disclosed set. By repeating this process, the observer can refine its prior knowledge about robot's plan and thereby tighten its estimate. The third problem aims to deal with this, by searching for the solution containing a plan and assuming exactly same plan is disclosed to the observer. This is the most specific prior knowledge any observer can have, making the sort of bootstrapping just described futile. This matter, and the chicken-and-egg nature of the fact that plan being sought is actually used in evaluating the stipulations, makes this third problem substantially more difficult.

At a high level, it is not hard to see why: the definitions in the previous two sections show that both $\mathtt{P}$ and $\mathtt{D}$ play a role in determining whether a plan satisfies a stipulation. Where $\mathtt{D}$ is known and fixed beforehand (for example, in Case IV, $\mathtt{D} = \mathtt{W}$, or Case III, $\mathtt{D} = \mathtt{P}^*$), a solution can proceed by building a correspondence in the triple graph $\mathtt{W} \otimes \mathtt{D} \otimes h^{-1}\langle \text{S}\textsc{de}(\mathtt{I})\rangle$ and searching in this graph for a plan. In $\text{S}\textsc{eek}_{\boldsymbol{x},\boldsymbol{\lambda}}$, however, one is interested in the case where $\mathtt{D} = \mathtt{P}$, where the divulged plan is tight, being the robot's plan exactly. We cannot search in the same product graph, because we can't make the correspondence since $\mathtt{D}$ has yet to be discovered, being determined only after $\mathtt{P}$ has been found. Crucially, the feasibility of $\mathtt{P}$ depends on $\mathtt{D}$, that is, on itself! Finding such a solution requires an approach capable of building incremental correspondences from partial plans. A key result of this paper is that $\text{S}\textsc{eek}_{\boldsymbol{x},\boldsymbol{\lambda}}$ is actually solvable without resorting to mere generate-and-C<span>HECK</span>.

Lemma 4.10. Let $\mathcal{W}$ be estimated world states for the finest observer $h\langle \mathtt{W}\rangle$, and let w be the world state which is observable to the robot. If there exists a solution for $\text{S}\textsc{eek}_{\boldsymbol{x},\boldsymbol{\lambda}}$, then there exists a solution that only visits each pair $(w, \mathcal{W})$ at most once.

Proof. Let $(\mathtt{P}, V_{\text{term}})$ and h be a solution for $\text{S}\textsc{eek}_{\boldsymbol{x},\boldsymbol{\lambda}}$. Suppose $\mathtt{P}$ visited $(w, \mathcal{W})$ n times. Let the set of actions taken at i-th visit be A_i. Then we can construct a new plan $(\mathtt{P}', V_{\text{term}})$ which always takes A_n at $(w, \mathcal{W})$. If $\mathtt{P}$ does not violate the stipulations, then $\mathtt{P}'$ will never do since $\mathtt{P}'$ is a shortcut of $\mathtt{P}$ and never visits more I-states than $\mathtt{P}$ does. In addition, $\mathtt{P}'$ will also terminate at the goal region if $\mathtt{P}$ does. □

Theorem 4.4. If there exists a solution for $\text{S}\textsc{eek}_{\boldsymbol{x},\boldsymbol{\lambda}}\big((\mathtt{W}, V_{\text{goal}}), \boldsymbol{x}, (I_f, \boldsymbol{x}), \boldsymbol{\lambda}, \Phi\big)$, then there exists a plan $\mathtt{P}$ that takes $(w, \mathcal{W}_B^{h\langle \mathtt{W}\rangle, \mathtt{P}})$ as its plan state, where w is the world state and the set $\mathcal{W}_B^{h\langle \mathtt{W}\rangle, \mathtt{P}}$ consists of the estimated world states for I-states B. Furthermore, if $(w, \mathcal{W}_B^{h\langle \mathtt{W}\rangle, \mathtt{P}}) \in V(\mathtt{P})$, then $\forall w' \in \mathcal{W}_B^{h\langle \mathtt{W}\rangle, \mathtt{P}}$, $(w', \mathcal{W}_B^{h\langle \mathtt{W}\rangle, \mathtt{P}}) \in V(\mathtt{P})$.

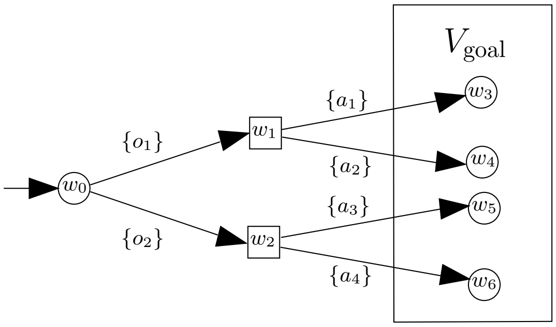


Figure 4.12: An example to show that actions should be obfuscated in $\text{SEEK}_{\boldsymbol{x},\boldsymbol{\lambda}}$.

Proof. Lemma 4.10 shows that we can treat $(w, \mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{P}})$ as the plan state for the plan to be searched for.

Since $w' \in \mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{P}}$, we have $\exists s \in \mathcal{S}_{w'}^{\mathtt{W}} \cap \mathcal{L}^I(\mathtt{P}) \cap h^{-1}[\mathbb{S}_B^{h\langle \mathtt{W}\rangle}]$. Since $s \in \mathcal{L}^I(\mathtt{P})$, s reaches w and $h(s)$ reaches B, we have s reaches the tuple $(w', \mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{P}})$. Hence, $(w', \mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{P}}) \in V(\mathtt{P})$. □

In searching for $(\mathtt{P}, V_{\text{term}})$, for any action state $v_p = (w, \mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{P}})$, we determine:

$w \in V_{\text{goal}}$: We must decide whether $v_p \in V_{\text{term}}$ holds or not;

$w \notin V_{\text{goal}}$: We must choose the set of nonempty actions to be taken at v_p. It has to be a set of actions, since these chosen actions are not only aiming for the goal but also obfuscating each other under the label map. We will show this with an example: in the world graph shown in Fig. 4.12, we may simply pick either a_1 or a_2 at w_1 in the planning problem. But to solve problem $\text{SEEK}_{\boldsymbol{x},\boldsymbol{\lambda}}$with stipulation $\boldsymbol{\Phi = (\neg w_3 \vee w_4) \wedge (w_3 \vee \neg w_4)}$, we have to choose 'both' action a_1 and a_2 when reaching w_1 by mapping them to the same image in the label map. Thence, the observer will never be able to distinguish the transitions to w_3 from w_4. Therefore it is necessary to choose a set of actions at a particular world state in $\text{SEEK}_{\boldsymbol{x},\boldsymbol{\lambda}}$, when the plan is also disclosed.

A state $v_p = (w, \mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{P}})$ is a terminating state in the plan when $\mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{P}} \subseteq V_{\text{goal}}$.

With action choices for each plan state $(w, \mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{P}})$ and label map h, we are able to maintain transitions of the estimated world states for B' after observing the image x. Now, if $(w, \mathcal{W}_B^{h\langle \mathtt{W}\rangle,\mathtt{P}})$ is an action state, let the set of actions taken at w be A_w. Then the label map h partitions the actions in $\cup_{w \in \mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}}} A_w$ into groups, each of which shares the same image. The estimated worlds states for B' transition in terms of groups

$$\begin{aligned}\mathcal{W}_{B'}^{h\langle\mathtt{W}\rangle,\mathtt{P}} = \Big\{ w' \in V(\mathtt{W}) \Big| (w, \mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}}) \notin V_{\text{term}}, w \in \mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}},& \\ \exists a \in A_w, h(a) = x, \textsc{TransTo}(w \xrightarrow{a} w')^{\mathtt{W}} \Big\}&.\end{aligned}$$

Conversely, if $(w, \mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}})$ is an observation state, let the observations available at w be O_w. Then h also partitions the observations in $\cup_{(w, \mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}}) \notin V_{\text{term}}} O_w$ and estimated world states for B' transition as

$$\begin{aligned}\mathcal{W}_{B'}^{h\langle\mathtt{W}\rangle,\mathtt{P}} = \Big\{ w' \in V(\mathtt{W}) \Big| (w, \mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}}) \notin V_{\text{term}}, w \in \mathcal{W}_B^{h\langle\mathtt{W}\rangle,\mathtt{P}},& \\ \exists o' \in O_w, h(o') = x, \textsc{TransTo}(w \xrightarrow{o} w')^{\mathtt{W}} \Big\}&.\end{aligned}$$

Similarly to SEEK$_\lambda$, we employ an observer's belief tree. A generalized example shown in Fig. 4.13, where there is an additional OR layer used to choose a set of actions for each action world state in the belief. A solution consisting of a plan and a label map, gives a subtree, where:

- a subset of actions is chosen for each action world state, and all observations available at the observation world state should appear in the subtree;
- a single partition is selected for all chosen actions or observations;
- all edges at each AND node are included;
- all belief vertices must satisfy the stipulations;
- each belief vertex in the subtree is visited at most once;
- all belief vertices can eventually lead to goal states.

This problem inherits all the properties of the subtree from SEEK$_\lambda$. In addition, it allows us to choose a subset of actions and map them to the same symbol, so as to satisfy the stipulation.

According to Theorem 4.10, we are only interested in the plans that visit each vertex in the subtree at most once and can eventually reach the goal.

4.5 Algorithms, implementations and experimental results

Previously, the CHECK problem boiled down to vertex checks on a constructed triple p-graph; the preceding section showed that the SEEK problems can be treated as subtree searches on the triple p-graph or observer's belief tree. Given recent advances in practical formal methods, instead of implementing these algorithms from scratch, we wish to utilize carefully optimized, off-the-shelf model checking tools to check information stipulations and to solve these planning problem for us. To do so, we encode the system dynamics as a Kripke structure, the information stipulation and goal attainment into a property specified by Computation Tree Logic (CTL), and then use nuXmv to verify this CTL property for us. The solution for SEEK$_{\boldsymbol{x}}$ may be constructed from the counterexamples of the negated CTL property. However, the requirement of non-conflicting partitions (to give a valid label map) is unfortunately too complicated to be encoded into a CTL property. Hence, we had to construct our own modifications of AND–OR search to solve both SEEK$_{\boldsymbol{\lambda}}$ and SEEK$_{\boldsymbol{x},\boldsymbol{\lambda}}$. Experimental results will also be presented in this section.

4.5.1 A computation tree logic based implementation toward CHECK and SEEK$_x$

In this section, we will present solutions for CHECK and SEEK$_{\boldsymbol{x}}$ through the use of CTL. The triple p-graph will be encoded as a Kripke structure, and the stipulations and goal conditions will be specified as properties, described by CTL, that should be satisfied in the computation tree of the Kripke structure. With both Kripke structure and CTL specifications, the software nuXmv [69] is able to evaluate whether these properties are satisfied or not, and give a counterexample in the latter case. We will use this mechanism to solve CHECK problem and SEEK plan.

For CHECK, we will firstly transform $\mathtt{W} \otimes \mathtt{P} \otimes \mathrm{SDE}(\mathtt{I})^{-1}$ into a Kripke structure, where each vertex $(v^{\mathtt{W}}, v^{\mathtt{P}}, v^{\mathrm{SDE}(\mathtt{I})^{-1}})$ is marked as stipulation satisfied if $v^{\mathrm{SDE}(\mathtt{I})^{-1}}$ satisfies the stipulations. Similarly in the problem of SEEK plan, we will also encode $\mathtt{W} \otimes \mathtt{D} \otimes \mathrm{SDE}(\mathtt{I})^{-1}$ into a Kripke structure, where each vertex $(v^{\mathtt{W}}, v^{\mathtt{P}}, v^{\mathrm{SDE}(\mathtt{I})^{-1}})$ is marked as a goal state if all world states associated with

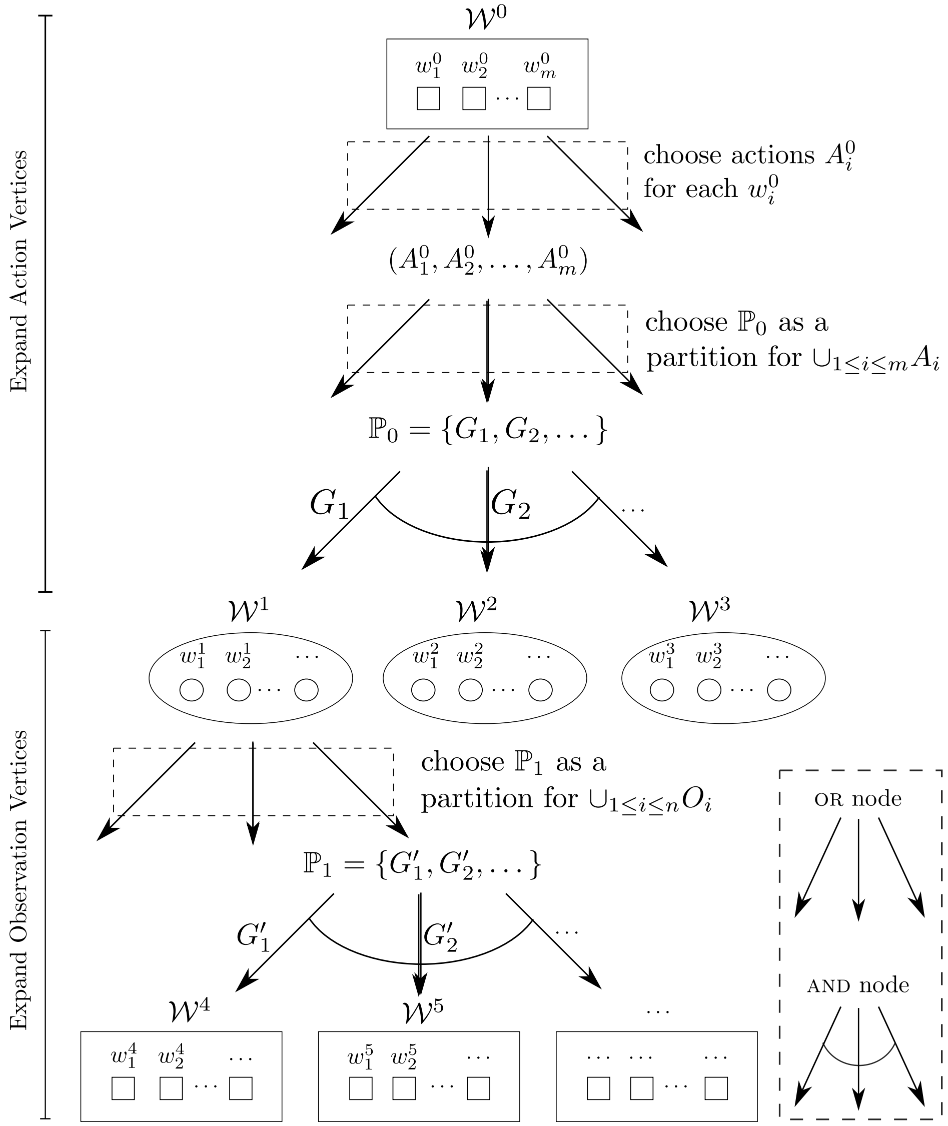

$\mathcal{W}^0$
w_1^0 w_2^0 w_m^0
choose actions A_i^0 for each w_i^0
$(A_1^0, A_2^0, \dots, A_m^0)$
choose $\mathbb{P}_0$ as a partition for $\cup_{1\leq i\leq m} A_i$
$\mathbb{P}_0 = \{G_1, G_2, \dots\}$
G_1
G_2
$\dots$
$\mathcal{W}^1$
$\mathcal{W}^2$
$\mathcal{W}^3$
w_1^1 w_2^1 $\cdots$
w_1^2 w_2^2 $\cdots$
w_1^3 w_2^3 $\cdots$
choose $\mathbb{P}_1$ as a partition for $\cup_{1\leq i\leq n} O_i$
$\mathbb{P}_1 = \{G'_1, G'_2, \dots\}$
G'_1
G'_2
$\dots$
$\mathcal{W}^4$
$\mathcal{W}^5$
$\dots$
w_1^4 w_2^4 $\cdots$
w_1^5 w_2^5 $\cdots$
$\cdots$ $\cdots$ $\cdots$
OR node
AND node
Expand Action Vertices
Expand Observation Vertices

Figure 4.13: The observer's belief tree for $\textsc{Seek}_{\boldsymbol{x},\boldsymbol{\lambda}}$.

$v^{\mathtt{P}}$ are in the goal region, and stipulation satisfied if $v^{\textsc{Sde}(\mathtt{I})^{-1}}$ satisfies the stipulations. Then the computation tree of these Kripke structure is shown in the lower half of Fig. 4.14. It has two phases: the initialization phase and execution phase. In the initialization phase, every variable, including *action, observation, label map*, is initialized from its domain. Given a plan and label map, these *variables* will be initialized a value by the plan and label map. Otherwise, the variable will be assigned a value from its domain. In the execution phase, the system assigns a value for *observation*, then assigns a value for *action* at the next time step, according to the Kripke structure. At each time step, only one variable is updated and, as there are several choices for assignments of variables, this gives branches in the computation tree.

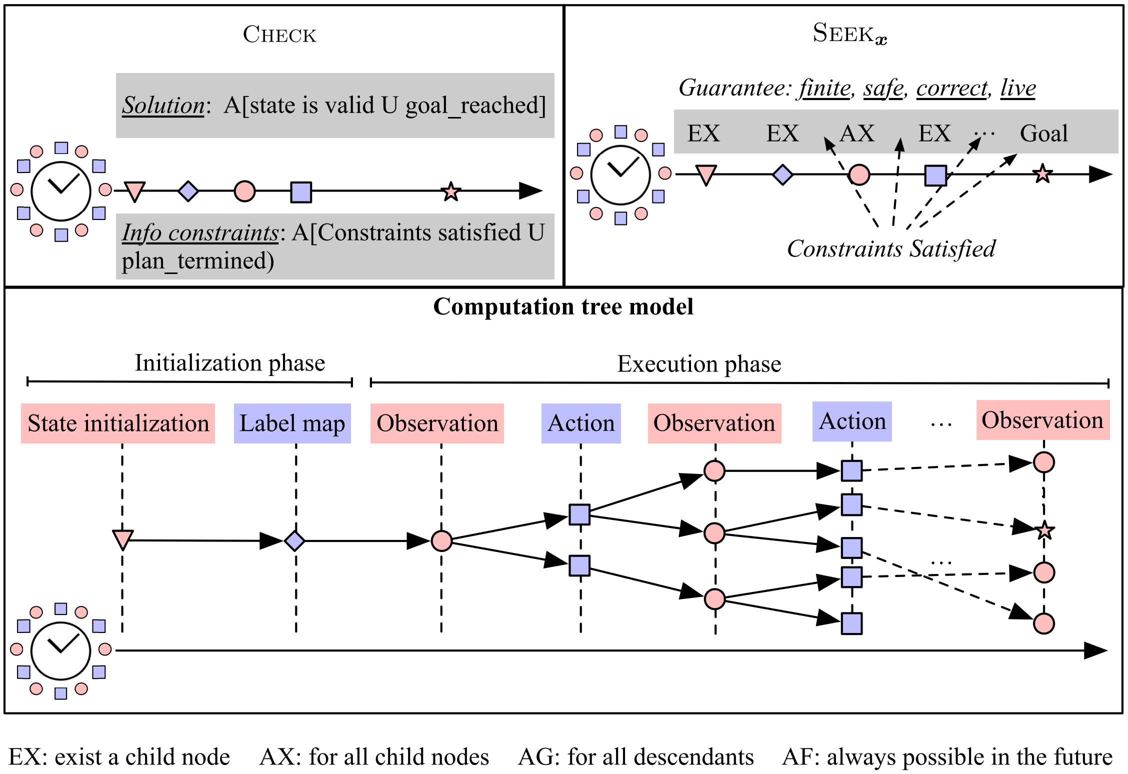


Figure 4.14: The computation tree and model specifications for both CHECK and SEEK problems.

CTL introduces two kinds of temporal operators, in addition to the logical operators. The first group is quantifiers over paths:

- A p requires that p should be true on all paths starting from the current state.
- E p requires that p should be true on at least one path from the current state where p also holds.

The second group is the path-specific quantifiers. Only the ones that will be used in this paper are listed:

- X p requires that p should be true at the next state in the path.
- p U q requires that p has to be true util q holds. Note that q is also required to hold some time in the future.

To CHECK whether the goal is reached, we will use "A [state is valid U goal is reached]", which guarantees: (i.) the plan is safe and correct since all states reached by the plan are valid states; (ii.) the plan is finite and live since the goal will eventually be reached in finite number of steps. Similarly, to specify that the stipulations are satisfied all the way toward goal states, we use CTL "A [stip_satisfied U (goal is reached $\wedge$ stip_satisfied)]".

To SEEK the plan that reaches the goal and satisfies the stipulations, we will use the combination of (AX EX). AX p requires that p holds at all the children states, which is same as the search in AND nodes. EX p requires that p holds at one of the children states, which shares the same spirit as the search for OR nodes. With k combinations of AX EX, we can search for the plan with depth k in terms of the action nodes. Hence, the we can use CTL "EX stip_satisfied $\big($$\wedge$ (goal $\vee$ AX stip_satisfied $\wedge$ (goal $\vee$ EX stip_satisfied)))$^{\mathrm{k}}$ (Goal $\wedge$ stip_satisfied)$\big)$" to search for the plan with maximum depth k. We will increase k from 0 to $|V(\mathtt{W})| \times |V(\mathtt{D})| \times V(\mathtt{I})$, which is the upper bound given in Theorem 4.2.

Once it has been determined that a plan exists, we can extract a plan by using the counterexamples from the negated CTL. The counterexample of the negated CTL only serves as one execution of a plan, which only gives us the action choices for a part of the plan. But we are still able to get the other actions choices in two ways: (i.) By changing the Kripke structure, we can force the system to transition to the other executions and get additional counterexamples. Combing all these counterexamples, we can build the plan. (ii.) By changing the CTL and initial state of the Kripke structure, we can find an action for each action state appeared in the plan. Eventually, we are able to build the whole plan. In this paper, we use the second method and the procedure is shown in Algorithm 4.

Algorithm 4: BUILDPLAN(W, CTL, initState)

```
NCLT←“!("+CTL+“)"
(result, counterexample)← nuXmv.evaluate(W, NCLT, initState)
if result is False then
    (s, event, t)← counterexample.next()
    initialize plan as empty
    if event is an action and s$\notin V_{\text{goal}}$ then
        CTL.removeFirstEX()
        subplan←BuildPlan(W, CTL, t)
        plan.addTransition(s, event ,subplan.initstate)
    else if event is an observation and s$\notin V_{\text{goal}}$ then
        CTL.removeFirstAX()
        for obs $\in$ W.outEvents(s) do
            tgt ← W.transTo(s, obs)
            subplan← BuildPlan(W, CTL, tgt)
            plan.addTrans(s, obs, subplan.initState)
        end for
    else
        plan.addTermVertex(s)
    end if
    plan.initState←initState
    return plan
else
    return empty
end if
```

4.5.2 Modified AND–OR search for SEEK$_\lambda$ and SEEK$_{x,\lambda}$

In both SEEK$_{\boldsymbol{\lambda}}$ and SEEK$_{\boldsymbol{x},\boldsymbol{\lambda}}$, we need to search for a subtree from the observer's belief tree. In terms of the subtree search, SEEK$_{\boldsymbol{\lambda}}$ can be treated as a special case of SEEK$_{\boldsymbol{x},\boldsymbol{\lambda}}$. In SEEK$_{\boldsymbol{x},\boldsymbol{\lambda}}$, we first decide to include a set of actions in the plan, and then a partition is chosen for these specific actions. While in SEEK$_{\boldsymbol{\lambda}}$, the first step is skipped and we choose a partition for all available actions. Here, avoiding needless repetition, we give a unified subroutine to expand the action and observation vertices for both $\textsc{Seek}_{\boldsymbol{\lambda}}$ and $\textsc{Seek}_{\boldsymbol{x},\boldsymbol{\lambda}}$.

To search for a subtree, we need to perform a modified AND–OR search on the observer's belief tree, which respects the AND–OR structure in the tree and should check whether there is any conflict for the selected partial partitions. Here, we write subroutines to expand action (Algorithm 5) and observation (Algorithm 6) vertices respectively. In Algorithm 5, we will firstly find all actions to be partitioned in $\textsc{Seek}_{\boldsymbol{\lambda}}$ and $\textsc{Seek}_{\boldsymbol{x},\boldsymbol{\lambda}}$. For $\textsc{Seek}_{\boldsymbol{\lambda}}$, we will consider finding label maps for all actions available at states in $\mathcal{W}$ (line 1–2). However, for problem $\textsc{Seek}_{\boldsymbol{x},\boldsymbol{\lambda}}$, we will firstly check whether we already reach the goal region. If all states in the belief are in the goal region, then we find the plan and return the result (line 4–5). Otherwise, we will choose a set of actions for each of the world states represented by this action vertex (line 9–14), and consider partitions for these chosen actions (line 15). For all possible actions in $\textsc{Seek}_{\boldsymbol{\lambda}}$ and the chosen actions in $\textsc{Seek}_{\boldsymbol{x},\boldsymbol{\lambda}}$, we will pick a partition which does not conflict with its ancestors (line 16–19). Next, we will integrate this choice with its ancestors and pass down to its subtree (line 20), and wait for the committed partial partitions from the subtree (line 21–24). If there is no valid partial partition (stipulations are satisfied on the reached vertices) from its children, then we will try another choice (line 25–30). If there are valid choices from its children, we will integrate and return the partial partitions to its parent node (line 31–32). If there is no choice that could have valid partial partitions for its children, then we will return empty, which will cause a backtrack (line 33). The subroutine to expand an observation vertex (Algorithm 6) shares most of the procedure with Algorithm 5, without searching for the actions.

If there exists a solution for $\textsc{Seek}_{\boldsymbol{x},\boldsymbol{\lambda}}$, then there exists a subtree where the chosen partitions do not conflict with each other and the belief vertices eventually terminate at the goal states under the action choices. The proposed algorithm will be able to find it.

Let the number of actions and observations in `W` be $|Y|$ and $|U|$, and the number of vertices be $|V|$. There are $2^{|U||V|}$ action choices to consider, in the worst case, for all the world states in $\mathcal{W}$. The total number of partitions is a Bell number $B_{|U|}$, where $B_{n+1} = \sum_{k=0}^{n} C_n^k B_k$ and $B_0 = 1$. For

Algorithm 5: ActionExpand($\mathcal{W}, \mathtt{P}, h, \mathbf{\Phi}$)

```
if the problem is $\textsc{Seek}_{\boldsymbol{\lambda}}$ then
    AllPartitions $\leftarrow$ All partitions for all available actions at states in $\mathcal{W}$
else if the problem is $\textsc{Seek}_{\boldsymbol{x},\boldsymbol{\lambda}}$ then
    if $\mathcal{W} \subseteq V_{\text{goal}}$ then
        return $(\mathtt{P}, h)$
    end if
    AllActChoices $\leftarrow \{\}$
    for $w \in \mathcal{W}$ do
        AllActChoices $\leftarrow$ AllActChoices $\times$ $w$.avblActs // Encode action choices for states in $\mathcal{W}$ cartesian product
    end for
    for actChoice $\in$ AllActChoices do
        AllChosenActs $\leftarrow \{\}$
        for $w \in \mathcal{W}$ do
            $A_w \leftarrow$ actChoice$[w]$ // Obtain the set of action choices for each state
            AllChosenActs $\leftarrow$ AllChosenActs $\cup A_w$
        end for
        $P[\mathcal{W}] \leftarrow$ actChoice // Put action choices in the plan
        AllPartitions $\leftarrow$ All partitions of AllChosenActs
    end for
end if
for partition $\in$ AllPartitions do
    $(\mathtt{P}', h')$ as a copy of $(\mathtt{P}, h)$
    NoSoln$\leftarrow$ $False$
    if $h'$ does not conflict with partition then
        $h' \leftarrow h$.integrate(partition)
        for *group* $\in$ partition do
            $\mathcal{W}' \leftarrow$ vertices $\mathcal{W}$ transitions to under *group*
            if $\mathcal{W}'$ satisfies stipulation $\mathbf{\Phi} \wedge \mathtt{P}$.contains($\mathcal{W}'$)=False then
                $(\mathtt{P}', h') \leftarrow$ ObservationExpand($\mathcal{W}', \mathtt{P}', h', \mathbf{\Phi}$)
                if $(\mathtt{P}', h')$ is empty then
                    NoSoln $\leftarrow$ $True$
                    break
                end if
            else
                NoSoln $\leftarrow$ $True$
                break
            end if
        end for
        if NoSoln is False then
            return $(\mathtt{P}', h')$
        end if
    end if
end for
return empty
```

Algorithm 6: ObservationExpand($\mathcal{W}, \mathtt{P}, h, \boldsymbol{\Phi}$)

```
if the problem is $\textsc{Seek}_{\boldsymbol{x},\boldsymbol{\lambda}}$ and $\mathcal{W} \subseteq V_{\text{goal}}$ then
    return $(\mathtt{P}, h)$
end if
$\text{AllObs} \leftarrow \{\}$
for $w \in \mathcal{W}$ do
    $O_w \leftarrow w$.avblObs
    $\text{AllObs} \leftarrow \text{AllObs} \cup O_w$
end for
$P[\mathcal{W}] \leftarrow \text{AllObs}$
$\text{AllPartitions} \leftarrow$ All partitions of $\text{AllObs}$
for $\text{partition} \in \text{AllPartitions}$ do
    $(\mathtt{P}', h')$ as a copy of $(\mathtt{P}, h)$
    $\text{NoSoln} \leftarrow False$
    if $h'$ does not conflict with $\text{partition}$ then
        $h' \leftarrow h$.integrate(partition)
        for *group* $\in$ $\text{partition}$ do
            $\mathcal{W}' \leftarrow$ vertices $\mathcal{W}$ transitions to under *group*
            if $\mathcal{W}'$ satisfies stipulation $\boldsymbol{\Phi} \wedge \mathtt{P}$.contains($\mathcal{W}'$)=False then
                $(\mathtt{P}', h') \leftarrow$ActionExpand($\mathcal{W}', \mathtt{P}', h'$, $\boldsymbol{\Phi}$)
                if $(\mathtt{P}', h')$ is empty then
                    $\text{NoSoln} \leftarrow True$
                    break
                end if
            else
                $\text{NoSoln} \leftarrow True$
                break
            end if
        end for
        if $\text{NoSoln}$ is False then
            return $(\mathtt{P}', h')$
        end if
    end if
end for
return empty
```

each partition, the number of groups we must consider is $|U|$. To expand an action vertex in the search tree, the computation complexity is $2^{|U||V|}|U|B_{|U|}$. Similarly, the complexity to expand an observation vertex is $|Y|B_{|Y|}$. If the depth of the tree is d, then the computational complexity is $O(2^{d|U||V|})$.

The search algorithm for $\text{SEEK}_{\boldsymbol{\lambda}}$ will be treated as a special case of Algorithm 5 and 6. It does not need to check goal conditions when expanding the vertices, or enumerate all actions for the vertices since the actions are given. In addition, the vertices will be marked as stipulation satisfied if (i). it is not reached by the plan P or, (ii). it satisfies stipulation $\boldsymbol{\Phi}$ if reached by P. The algorithm for $\text{SEEK}_{\boldsymbol{\lambda}}$ can be constructed by removing the highlighted part of Algorithm 5 and 6.

4.5.3 Experimental results

We implemented all the algorithms in this paper, the mainly using Python. All executions in this section used a OSX laptop with a 2.4 GHz Intel Core i5 processor.

To experiment we constructed a 3×4 grid for the nuclear inspection scenario of Fig. 4.1. Including the differing facility types and radioactivity status, the world graph is a p-graph with 96 vertices before state-determined expansion (154 vertices for the state-determined form). The robot can move left, right, up, down one block at a time. After the robot's movement, it receives 5 possible observations: pebble bed facility or not (only when located at the blue star), radioactivity high or low at one of the '**?**' cells, and cell is an exit. But the observer only knows the image of the actions and observations under a label map. The stipulation requires that the observer should learn the radioactivity strength, but should never know the facility type.

Firstly, we CHECK whether the plan and label map pair given in Fig. 4.15a solve the problem. The plan reaches the goal but the observer cannot distinguish the radioactivity status when the full world is divulged (i.e., $\mathtt{D} = \mathtt{W}$). Also, it violates the stipulations because the facility type is leaked when the exact plan is disclosed (i.e., $\mathtt{D} = \mathtt{P}$). Evaluation takes less than 1 second in both tree-based and CTL methods.

For this setting with the same label map and disclosed plan $\mathtt{D} = \mathtt{W}$, no satisfying solution exists, and hence $\text{SEEK}_{\boldsymbol{x}}$ returns False.

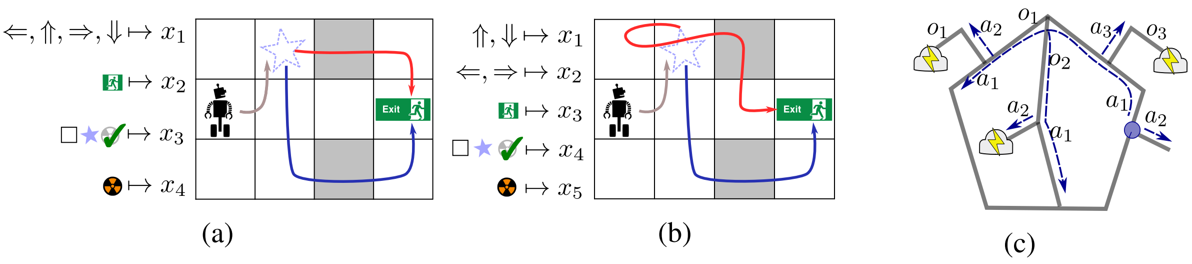


Figure 4.15: The scenario and results for CHECK and SEEK problem: (a) shows the plan and label map to be checked in the nuclear inspection scenario, when the observer knows nothing about robot's plan or the exact plan. (b) gives the plan found by SEEK$_{\boldsymbol{x}}$ with the given label map. The same plan will also be used as an input for SEEK$_{\boldsymbol{\lambda}}$. (For (a) & (b) plans can be understood as follows: the robot traces the gray arrow, then the blue one if blue light is seen, the red one otherwise.) (c) shows the pentagonal world in SEEK$_{\boldsymbol{x},\boldsymbol{\lambda}}$, where the robot moves along the gray lines.

Now altering the label map so that $h(\Uparrow) = h(\Downarrow)$ and $h(\Leftarrow) = h(\Rightarrow)$, a plan can be found (with the world graph disclosed, $\mathtt{D} = \mathtt{W}$). It takes 11 seconds for the AND–OR search and 24 seconds for the CTL-based implementation to find their solutions. The CTL solver takes longer, but it prioritizes finding the plan of shortest length first. The plan found by CTL is shown in Fig. 4.15b. As the plan found by AND–OR search is lengthy, we omit it.

If we disclose the same plan to the observer (i.e., $\mathtt{D} = \mathtt{P}$), then the label map in Fig. 4.15b tells the observer the robot's exact trajectory (blue or red trajectory). When the observer knows that the robot moves along the blue trajectory, the facility type has to be a pebble bed reactor; the red trajectory indicates a breed reactor. This clearly violates the stipulations. We ran the algorithm for SEEK$_{\boldsymbol{\lambda}}$ to search for an appropriate label map for the plan shown in Fig. 4.15b when $\mathtt{D} = \mathtt{P}$. It turns out that the stipulations can be satisfied by additionally making all four actions ambiguous, i.e., $h(\Uparrow) = h(\Downarrow) = h(\Leftarrow) = h(\Rightarrow)$.

Since, for the nuclear inspection scenario, SEEK$_{\boldsymbol{x},\boldsymbol{\lambda}}$ doesn't return any result within reasonable time we opted to examine a smaller problem. Here a robot moves in the pentagonal world shown in Fig. 4.15c. The robot can either decide to loop in the world (a_1) or exit the loop at some point (a_2 or a_3). We wish to find a plan and label map pair so that the robot can reach some charging station. The observer should not be able to distinguish the robot's position when at either of the top two charging locations. $\textsc{Seek}_{\boldsymbol{x},\boldsymbol{\lambda}}$ gives a plan which moves forward 6 times and then exits at the next time step. Additionally, to disguise the actions and observations after the exit, it maps $h(a_2) = h(a_3)$ and $h(o_1) = h(o_3)$. Note that in this problem, the robot reaches a goal, without considering the stipulations, by taking the exit at the next time step. The stipulations force the robot to navigate at least one loop in the world to conflate state for the sake of the observer.

4.6 A playback observer

In the above sections, we examined what information could be learned from observation of plan execution by a filtering adversary, namely, one who uses its current estimate and the latest observation to construct a new estimate for the current time. Starting from this section, we consider a different kind of adversary—a playback observer, one who may construct estimates for any past time by playing back all its received observations. We will present algorithms to check a plan or search for a plan, in the presence of a playback observer.

With the planning problem, the disclosed plan and the disclosed symbols through the label map, a playback observer will (1) infer all potential interactions that could happen in the world according to its prior knowledge, (2) play back these interactions to identify all consistent state trajectories in the world and, (3) obtain the estimated world states at each time step. Formally, we have:

Definition 4.19 (playback observer's estimate)**.** Given planning problem $(\mathtt{W}, V_{\text{goal}})$, robot's disclosed plan $\mathtt{D}$, and information disclosure policy h, when the observer receives a string $\boldsymbol{x} = x_1 x_2 \dots x_k$,

1) the set of potential strings inferred by the observer is $\mathcal{S}_{\boldsymbol{x}} = h^{-1}(\boldsymbol{x}) \cap \mathcal{L}^{I}(\mathtt{W}) \cap \mathcal{L}^{I}(\mathtt{D})$,

2) the set of trajectories consistent with those potential strings is $\mathcal{T}_{\boldsymbol{x}} = \cup_{s' \in \mathcal{S}_{\boldsymbol{x}}} \textsc{Traj}^{\mathtt{W}}_{s'}$.

3) the set of estimated world states $\mathcal{W}^{m}_{\boldsymbol{x}}$ at time step m is extracted from these state trajectories, i.e., $\mathcal{W}^{m}_{\boldsymbol{x}} = \cup_{\tau \in \mathcal{T}_{\boldsymbol{x}}} \tau[m]$, where $\tau[m]$ is the m^{th} world state on trajectory τ.

Now, we have the following satisfaction problem for the playback observer:

Problem: $\textsc{CheckPlayback}\left((\mathtt{W}, V_{\text{goal}}), (\mathtt{P}, V_{\text{term}}), h, \mathtt{D}, \mathbf{\Phi}\right)$

Input: A planning problem $(\mathtt{W}, V_{\text{goal}})$, a plan $(\mathtt{P}, V_{\text{term}})$, an information disclosure h, a disclosed plan $\mathtt{D}$, and a stipulation $\mathbf{\Phi}$.

Output: True if $(\mathtt{P}, V_{\text{term}})$ solves the planning problem $(\mathtt{W}, V_{\text{goal}})$ and, $\forall s \in \mathcal{L}^{I}(\mathtt{W}) \cap \mathcal{L}^{I}(\mathtt{P})$, $\mathbf{\Phi}$ is always evaluated as True on $\mathcal{W}^{m}_{h^{-1}\circ h(s)}$ for all integer $0 \leq m \leq k$; False otherwise.

4.7 An algorithm to check a plan with hindsight

To solve CHECKPLAYBACK, the key is to trace all trajectories that are consistent for the images of the strings in the plan. Unhappily there can be many strings such strings. Instead of computing the beliefs for each string from scratch, we propose an graph-based algorithm to produce the set of all beliefs for any string and its extensions. The stipulations are violated once it is violated on some string in the plan.

4.7.1 A p-graph representing observer's prior knowledge

First, we construct a p-graph to integrate observer's prior knowledge about the planning problem and the disclosed plan, and then compute the estimated trajectories in this new graph.

To combine the observer's prior knowledge about the world and robot's plan, we construct a product graph $\mathtt{J} = \mathtt{W} \otimes \mathtt{D}$ as the tensor product graph of world $\mathtt{W}$ and disclosed plan $\mathtt{D}$ with initial states $V_0(\mathtt{W}) \times V_0(\mathtt{D})$. The language of this joint graph is the set of executions that could happen in the world and could potentially be taken by the robot's plan, i.e., $\mathcal{L}^{I}(\mathtt{J}) = \mathcal{L}^{I}(\mathtt{W}) \cap \mathcal{L}^{I}(\mathtt{D})$. In addition, if we trace any string $s \in \mathcal{L}^{I}(\mathtt{J})$ in $\mathtt{J}$, take the first part (world state) of each joint state in the trajectory $\textsc{Traj}^{\mathtt{J}}_{s}$, and denote this new sequence as $\textsc{Traj}^{\mathtt{J},\mathtt{W}}_{s}$, then we will obtain exactly the same trajectory as tracing s in the world, i.e., $\textsc{Traj}^{\mathtt{J},\mathtt{W}}_{s} = \textsc{Traj}^{\mathtt{W}}_{s}$.

Now, instead of computing the trajectories in the world graph, we can compute it in the joint graph $\mathtt{J}$, essentially pretending that it is new 'world'. The observer's belief is a set of states in the joint graph.

4.7.2 A graph-based algorithm for CHECKPLAYBACK

To solve CHECKPLAYBACK, we need to check both the solutions for the planning problem and the stipulations on the disclosed information. A tensor product graph is constructed to examine whether the plan always terminates at a goal state. To examine the disclosed information, we give an algorithm that incrementally constructs all beliefs which are learned by the playback observer. In estimation with hindsight, the observer is able to playback the observations, refining previous beliefs by eliminating the states from which subsequent events crash. Instead of playing back each string in the plan, we conduct a breadth-first search (BFS) on the belief graph to efficiently simulate the observer's estimation process as shown in Algorithm 7.

Firstly, we construct a product graph $\mathtt{W} \otimes \mathtt{P}$ to check whether every termination state in $\mathtt{P}$ is paired with some goal state in $\mathtt{W}$ in the product graph (line 1–4). If not, then the plan does not solve the planning problem. Otherwise, it does.

The observer is only able to see the strings in $h\langle \mathtt{J} \rangle$, which is obtained by replacing the labels on the edges in $\mathtt{J}$ with their images after label map h. In $h\langle \mathtt{J} \rangle$, one string may reach two states non-deterministically. We construct a deterministic form $\mathtt{J}'$ for $h\langle \mathtt{J} \rangle$, following Algorithm 2 in [12] (line 5–6). During this state-determined transformation, states in $h\langle \mathtt{J} \rangle$ are merged into a single state v' in $\mathtt{J}'$ if they are non-deterministically reached by some string. We say that these states are the corresponding states for v'. Each state in $\mathtt{J}'$ is a belief state, and its corresponding states are included in the belief. Not all belief states in $\mathtt{J}'$ will be active and perceived by the observer, since the plan may not produce those beliefs. We construct a product graph $\mathtt{J}' \otimes h\langle \mathtt{P} \rangle$ to mark each belief state in $\mathtt{J}'$ as active, if it is paired with some plan state in the product graph (line 7–8). Stipulations will be evaluated on these active belief states once they are generated. The plan fails to satisfy the stipulations, if the stipulations are violated on any active belief.

The active belief states in $\mathtt{J}'$ only contribute to part of the beliefs generated by the observer. They will be refined when the observer can play back its observations. Some states in the beliefs of past times will be eliminated when there is no string as an extension of these states to the states in the frontier. We say that these states are not alive. A BFS search on $\mathtt{J}'$ is conducted to simulate this playback estimation (line 9–33). Starting from each active belief state V_k in the frontier of the search, we mark each corresponding state in belief V_k as alive. Then we propagate liveness backward to find the states in the past beliefs that are not alive. For each transition $V_{k-1} \xrightarrow{x} V_k$, we mark each state in the belief V_{k-1} as alive if the state transitions to some state in V_k under x in $h\langle \mathtt{J} \rangle$. Otherwise, we mark that it is not alive. We refine belief V_{k-1} by removing all states that are not alive, and construct a new belief V'_{k-1} (line 23–25). When none of the states in belief V_k are eliminated, i.e., $V'_{k-1} = V_{k-1}$, then we may stop propagating the liveness, since no new beliefs will be generated. If V'_{k-1} is finer than V_{k-1}, then one must keep propagating the liveness in V'_{k-1} backward (line 28–30). Stipulations must be evaluated on the refined belief states when they are generated (line 26–27). The evaluation can stop early when one of these beliefs violates the stipulations.

4.8 An incremental algorithm to search for a plan

We are interested in seeking plans that never disclose information to playback observers that violate given stipulations:

Problem: SEARCHPLAYBACK $\big((\mathtt{W}, V_{\text{goal}}), \mathtt{D}, h, ?, \mathbf{\Phi}\big)$

Input: A planning problem $(\mathtt{W}, V_{\text{goal}})$, a disclosed plan $\mathtt{D}$, an information disclosure h, a stipulation $\mathbf{\Phi}$.

Output: A plan $(\mathtt{P}, V_{\text{term}})$, such that CHECKPLAYBACK$\big((\mathtt{W}, V_{\text{goal}}), \mathtt{D}, h, \mathtt{P}, \mathbf{\Phi}\big) = True$.

As mentioned in the satisfaction problem, $\mathtt{J}'$ captures the observer's beliefs. We are interested in searching for a plan which reaches the goal in $\mathtt{W}$ and always generates beliefs in $\mathtt{J}'$ that satisfy the stipulations. To do this, we construct a product graph of $h^{-1}\langle \mathtt{J}' \rangle$ and $\mathtt{W}$, denoted as $\mathtt{T} = h^{-1}\langle \mathtt{J}' \rangle \otimes \mathtt{W}$. The joint state in $\mathtt{T}$ consists of two parts: states in $h^{-1}\langle \mathtt{J}' \rangle$ to examine stipulations and states in $\mathtt{W}$ to examine the goal condition. Next, we conduct a AND-OR search on $\mathtt{T}$ to find a subgraph such that (i) each action state has a single outgoing edge bearing one action, (ii) each observation state has outgoing edges bearing all observations in the world, (iii) all beliefs generated from $\mathtt{J}'$ in the subgraph must satisfy stipulations, and (iv) the states eventually terminating in the subgraph must give world states all of which are in the goal region. When constructing the AND-OR search, we are able to incrementally search for an action for each action state in the subgraph, and obtain a partial plan. By calling the CHECKPLAYBACK procedure, we are able to examine whether stipulations are satisfied on all generated beliefs of the partial plan. If the partial plan fails to satisfy the stipulations, then one must backtrack the action choice just made and choose a different action. This process is repeated until a solution is found.

4.9 Summary

In this chapter, we study planning with stipulations on the information divulged from the robot's plan to an observer. We formalize the observer as a filter or a smoother, which allows it to reason about what is happening in the world and what the robot knows about some fact. To make such inferences, the observer uses its prior knowledge about robots plan and observations about robots plan execution. The observer we formulate is rich enough to capture its different prior knowledge and capabilities to compute and store estimation results. We also introduce an information disclosure policy to determine how the information is disclosed to the observer.

With a filtering observer, the appropriate solution concept for privacy-preserving planning consists of a pair: a plan and an information disclosure policy. We proposed algorithms to search for such solutions jointly, so that the information learned by the observer is constrained, with any given prior knowledge and the most complete tracking capability (i.e., the finest internal structure). We are able to search for a solution, even the observer somehow knows the plan yet to be sought. We also develop algorithms to check a plan and search for a plan as a solution for a playback observer, which runs a smoother to examine the strings with the power of hindsight and to estimate the robot–world interaction.

Algorithm 7: CheckPlayback$((\mathtt{W}, V_{\text{goal}}), (\mathtt{P}, V_{\text{term}}), h, \mathtt{D}, \mathbf{\Phi})$

```
$K = \mathtt{W} \otimes \mathtt{P}$
for $(w, r) \in K.vertices$ do
    if $r \in V_{\text{term}}$ and $w \notin V_{\text{goal}}$ then
        return False
    end if
end for
$\mathtt{J} \leftarrow \mathtt{W} \otimes \mathtt{D}$
$\mathtt{J}' \leftarrow \textsc{Sde}(\mathtt{h}\langle \mathtt{J} \rangle)$
$Q \leftarrow \mathtt{J}' \otimes h\langle \mathtt{P} \rangle$
$active\_v = \pi_{\mathtt{J}'}(Q.vertices)$
$q \leftarrow [\mathtt{J}'.initVertex]$
$visited = []$
while $q$ is not empty do
    $m \leftarrow q.pop()$
    add $m$ to $visited$
    if $m \notin active\_v$ then
        continue
    end if
    if $m$.correspState violates stipulation $\mathbf{\Phi}$ then
        return False
    end if
    $p \leftarrow []$
    for $(n, x) \in \mathtt{J}'.incoming(m)$ do
        add $(n, x, m)$ to $p$
    end for
    $p \leftarrow [(n, x, m)]$
    while $p$ is not empty do
        $(v', x, v) \leftarrow p.pop()$
        $b_v \leftarrow v.correspStates$
        $b_{v'} \leftarrow refine(\mathtt{J}, v'.correspStates, x, b_v)$
        if $b_{v'}$ violates stipulation $\mathbf{\Phi}$ then
            return False
        end if
        if $b_v! = b_{v'}$ and $v' \notin \mathtt{J}.initStates$ then
            for $(u, x) \in \mathtt{J}'.incoming(v')$ do
                add $(u, x, v')$ to $p$
            end for
        end if
    end while
    for $w \in m.children()$ do
        if $w \notin visited$ then
            add $w$ to $q$
        end if
    end for
end while
return True
```

5. SENSOR DESIGN IN PRIVACY-PRESERVING PLANNING PROBLEMS*

This chapter focuses on sensor design in privacy-preserving planning problems. It introduce a mathematical description for sensors based on a set cover structure, and proposes an algorithm to enumerate all abstract sensors that provide sufficient information for a robot to reach its goal, as well as constraining the robot's belief. It contributes data structures that enable whole sets of sensors to be summarized via a single special representative. It also gives a means by which other aspects (either task domain knowledge, sensor technology or fabrication constraints) can be incorporated to reduce the sets to be enumerated.

To illustrate multiple aspects of the sensor design problem, we introduce the following simple scenario as shown in Fig. 5.1: A robot, uncertain about its initial position and incapable of navigating stairs, needs to reach a charging station. We give four exemplar sensors that, under different plans, ensure goal attainment:

(i) a camera to distinguish red and gray helps to eliminate uncertainty in the initial pose when following the top plan;

(ii) a robot with a distance sensor can disambiguate initial position 2 from $\{1, 3\}$, since it observes that it is near the wall after two forward moves only when it starts at 2, while observing medium or far from the wall for $\{1, 3\}$;

(iii) with a lidar sensor the robot can distinguish 3 from $\{1, 2\}$ since, after three forward moves from 3, it senses a different polygon from those of 2 and 3.

(iv) the vacuous sensor also suffices, albeit only under the assumption of benign collisions, and with many steps.

The sensors do not all quash the uncertainty completely, but they eliminate enough to reach the goal under different plans. For example, the robot with a distance sensor never resolves whether it came from 1 or 3 in executing the corresponding plan. The robot with a lidar sensor does not

[*] Reprinted with permission from "Abstractions for computing all robotic sensors that suffice to solve a planning problem" by Yulin Zhang and Dylan Shell, 2020. *2020 IEEE International Conference on Robotics and Automation*. 8469-8475. Copyright 2020 by *IEEE*.

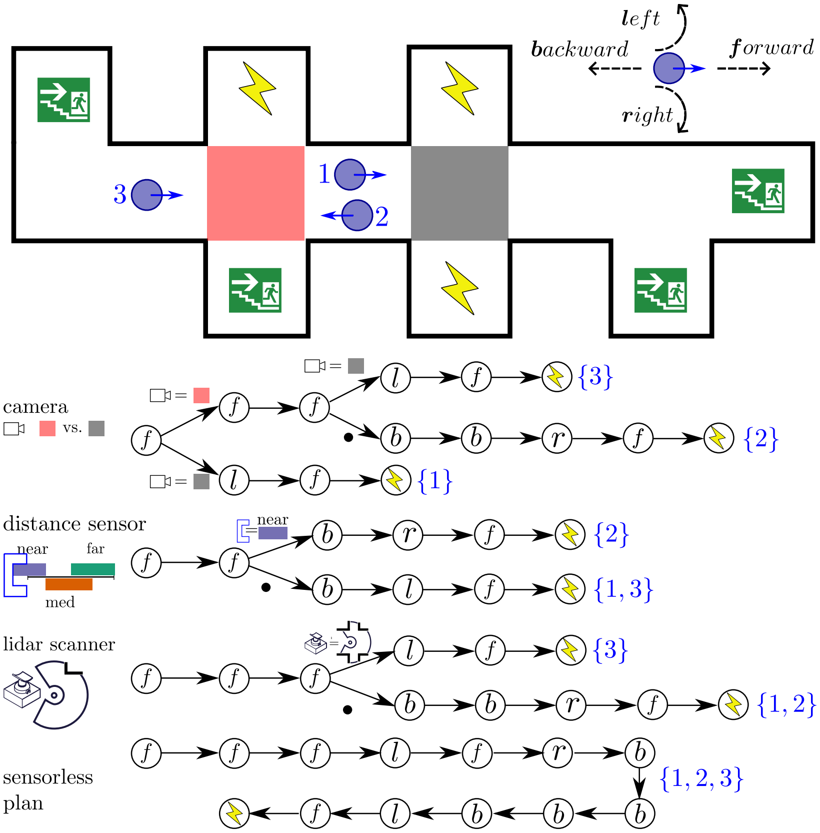


Figure 5.1: A wheeled robot (as a blue disk) needs a charging station (the lightning bolts), but is slightly lost (the uncertainty in its initial pose is shown visually, as three possibilities). Unable to navigate stairs, it must avoid those locations lest it topple down a stairwell. The robot is able to recharge its battery despite the presence of uncertainty, with the help of either a camera, a simple linear distance sensor, or a short-range scanning lidar. (If bumping into walls is permitted, a sensorless plan is possible as well.)

distinguish 1 from 2. But, in both cases, the robot reaches a charger. There are also important differences in the sensors' fidelity. The camera divides all the locations into three equivalent classes: a red location, a gray one, and the white ones. In contrast, the distance sensor's specification tells us that middle range distance readings are noisy, failing to separate medium and far distances from the wall crisply (when the robot observes 'med', then it is either at a medium or a far range from the wall; when obtaining 'near', it is close to the wall).

Sensors can be modeled as the information they provide for the plan. While previous works [58, 70] regard sensors as partitions over all events to be perceived, this paper is more general, considering sensors as covers. Doing so requires some care, including new representations and means to lessen the combinatorial explosion that a naïve treatment entails.

5.1 Model

We study a setting depicted in Fig. 5.2. The robot is equipped with a *sensor*, through which it receives observations from the world. Actions are chosen to alter states according to the robot's *plan* to, ultimately, reach some goal states in the world. The sensor may have limited fidelity and fail to distinguish different observations from the world. The uncertainty in sensing is modeled via a type of function, termed a *sensor map*. These elements are formalized in terms of p-graphs and sensor maps that we outline below.

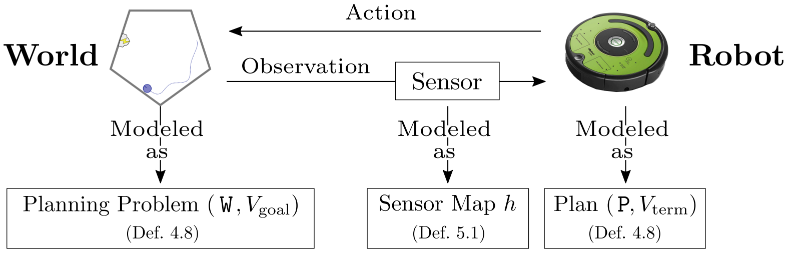


Figure 5.2: An overview of the setting: the robot is modeled abstractly as realizing a plan to achieve some goal in the world. The sensor is modeled as a sensor map. Both the world and the plan have concrete representations as p-graphs.

5.1.1 Sensor maps

As we see in Definition 4.8, the world and the robot are modeled are two p-graphs that are coupled, resulting in a planning problem. Sensors influence this coupling relationship by influencing the distinguishability of observations made by the robot. Conflations and corruptions of events are treated next.

Definition 5.1 (observation/sensor maps [59])**.** A *sensor map* on p-graph $\mathtt{G}$ is a function $h : Y \to \mathcal{P}(X)\setminus\{\varnothing\}$ mapping from an observation in Y to a non-empty set of observations X, where $\mathcal{P}(X)$ is the powerset of X.

If h maps y_1 to $\{x_1, x_2, x_3, x_4\}$ then, when event y_1 happens in the world, the robot may receive any of those four values as a sensor reading; further, we assume the choice happens nondeterministically.

Given any subset of sensor readings $X' \subseteq X$ as input to (that is, observed or perceived by) the robot, the associated observations within the world $\mathtt{W}$ are related via the preimages of X' under h, denoted by $h^{-1}(X') \coloneqq \{\ell \in Y(\mathtt{W}) \mid h(\ell) \cap X' \neq \varnothing\}$. Below, the notation for a sensor map h and its preimage h^{-1} is extended in the usual manner to p-graphs by applying the function to labels on each observation edge, i.e., in the obvious way.

Definition 5.2 (solves under sensor map)**.** A plan $(\mathtt{P}, V_{\text{term}})$ *solves a planning problem* $(\mathtt{W}, V_{\text{goal}})$ *under sensor map* h if $(h^{-1}\langle\mathtt{P}\rangle, V_{\text{term}})$ solves $(\mathtt{W}, V_{\text{goal}})$.

5.1.2 Sensor design in a planning problem

Now we can define the central problem of the paper:

Problem: Joint-Plan-Sensor-Design (JPSD)

Input: A planning problem $(\mathtt{W}, V_{\text{goal}})$

Output: All the sensor maps $\mathcal{H}$, such that there exists a plan $(\mathtt{P}, V_{\text{term}})$ to solve the planning problem $(\mathtt{W}, V_{\text{goal}})$ under each sensor map $h \in \mathcal{H}$.

5.2 Computational abstractions for sensor maps

Sensor maps map observations to their images, while the planning problem is defined in the preimage space. To solve this problem, we will begin by considering an alternate form in the preimage space for the sensor maps.

5.2.1 Equivalent representation for sensor maps

Any sensor map has an equivalent cover representation.

Theorem 5.1. For planning problem $(\mathtt{W}, V_{\text{goal}})$, any sensor map h is equivalent to a cover up to plan solvability.

Proof. $\Rightarrow$: Given any sensor map h, to see whether a plan is a solution (cf. Def. 5.2), we must determine the preimage $h^{-1}(x) = \{\ell \in Y(\mathtt{W}) \mid h(\ell) = x\}$ for single readings x. Collect all the data associated with h, on the X, via

$$M = \{h^{-1}(x_1), h^{-1}(x_2), \ldots, h^{-1}(x_n)\},$$

where $X = \{x_1, x_2, \ldots, x_n\}$. This is a multiset. But now observe that where for any x_i and x_j we have $h^{-1}(x_i) = h^{-1}(x_j)$, we can construct a new sensor map by replacing x_i and x_j with a new symbol x'. This new sensor map is also a solution if and only if h is a solution for JPSD. Under this new sensor map, no two readings in the sensor map share the same preimage, and h^{-1} can be thus represented as set

$$C = \{h^{-1}(x_1), h^{-1}(x_2), \ldots, h^{-1}(x_n)\},$$

where $\cup_{x_i \in X} h^{-1}(x_i) = Y(\mathtt{W})$. The set above is called a *cover* for set $Y(\mathtt{W})$. Henceforth, we call the cover for sensor map h an *observation cover*, denoting it C_h. (It is a subset of the powerset of $Y(\mathtt{W})$, i.e., $C_h \subseteq \mathcal{P}(Y(\mathtt{W})) \setminus \{\varnothing\}$.)

$\Leftarrow$: Having just showed that there exists a cover interpretation for any sensor map h, we now construct a sensor map for any observation cover. Suppose cover $\{S_1, S_2, \ldots, S_k\} \subseteq \mathcal{P}(Y(\mathtt{W}))$ for set $Y(\mathtt{W})$ is given. Taking the first k natural numbers for X, consider a label map h defined so that $y \overset{h}{\mapsto} \big\{i \in \{1, 2, \ldots, k\} \mid y \in S_i\big\}$.

Together, the cover C_h is an equivalent representation for any sensor map h, up to plan solvability. □

5.2.2 Operations on observation covers

Next, we give two operations on covers (projection and intersection) that are useful for sensor maps.

The sensor map is a cover for all observations in the planning problem. Only some small number of observations may be applicable while at particular world states. We are interested in how the observations in such a reduced set conflate with each other. This is realized via an operation that reduces the domain:

Definition 5.3 (cover projection)**.** For cover $C = \{G_1, G_2, \dots, G_n\}$, denote its domain by $d(C) = \cup_{1 \leq i \leq n} G_i$. Then the *projection of* C on any domain D is $\pi_C(D) = \{G_i \cap D | G_i \in C\}$.

We call sensor map $\pi_C(D)$ with reduced domain $d(C) \cap D$ a *partial sensor map*. The word 'partial' is apt as the sensor map need not cover every observation in the planning problem.

On the other hand, we are also interested in finding all sensor maps with certain behavior on their restrictions. Specifically, we desire to find all label maps which, when given two partial label maps, agree with those label maps on their projections. This comes from an intersection between two partial sensor maps.

Definition 5.4 (cover intersection)**.** For any two partial sensor maps, expressed as cover C_1 and C_2, with the union of their domains $D = d(C_1) \cup d(C_2)$, then let $\mathbb{D}$ be all covers[*] whose domain is D. Then the *intersection* of C_1 and C_2, denoted $C_1 \sqcap C_2$, is defined so that $\forall C' \in \mathbb{D}$, we have $C' \in C_1 \sqcap C_2$, if and only if

(a) $d(C') = d(C_1) \cup d(C_2)$, and

(b) $\pi_{C'}(d(C_1)) \subseteq C_1$ and $\pi_{C'}(d(C_2)) \subseteq C_2$.

Note that $\sqcap$ is associative and that $C_1 \sqcap \varnothing = \varnothing$ for any cover C_1. When no cover that satisfies *(a)* and *(b)* above, then $C_1 \sqcap C_2 = \varnothing$. We say that C_1 is *compatible* with C_2 if $C_1 \sqcap C_2 \neq \varnothing$. We will also lift this notation to the intersection of lists of covers. In writing $\mathbb{L}_1 \sqcap \mathbb{L}_2$ for two lists of covers $\mathbb{L}_1$ and $\mathbb{L}_2$, we mean $\mathbb{L}_1 \sqcap \mathbb{L}_2 = \cup_{C_1 \in \mathbb{L}_1, C_2 \in \mathbb{L}_2} C_1 \sqcap C_2$.

5.3 Jointly searching for sensor designs & plans

First, we construct a robot's belief tree and then give approaches to search for all sensor designs and plans in it.

[*] Throughout, variables in blackboard bold represent a list of covers.

5.3.1 The belief tree under different sensor maps and actions

The robot's plan must manage uncertainties owing to initial ignorance, action non-determinism, and sensor imperfection. The robot's belief expresses this uncertainty, which we represent as a set of states. Without this, the robot may violate plan safety by trying to execute some action that is not possible in its actual state. The dynamics of the belief will be captured by a finite tree structure, where each vertex lists a set of world states, the robots' belief. Plans need only visit each belief vertex at most once.

Theorem 5.2. Let $\mathcal{W}$ be the set of estimated world states for the robot's belief. For any sensor design $h \in \mathcal{H}$, where $\mathcal{H}$ is a set of sensor maps for JPSD, if there exists a plan that solves the planning problem under h, then there exists another plan, also a solution, that visits $\mathcal{W}$ at most once under h.

Proof sketch. This theorem can be proved by constructing a new plan, which always takes the action chosen at the last visit at $\mathcal{W}$ under the same sensor map. Then the new plan is a shortcut of the original one. Inherited from the original plan, the new one will always terminate at the goal region. □

Let $A(w)$ be the set of outgoing events for vertex $w \in V(\mathtt{W})$. Then the belief tree, a sketch of which appears in Fig. 5.3, can be constructed as follows:

- ▷ *Initialization*: An initial vertex $\mathcal{W}^0$ of the same vertex type is created for the set of initial world states $V_0(\mathtt{W})$.
- ▷ *Expanding action vertex* $\mathcal{W}$: Collect the common actions as $U(\mathcal{W}) = \cap_{w \in \mathcal{W}} A(w)$, i.e., the set of actions each of which is available at every state in $\mathcal{W}$. Now, for any action $a \in U(\mathcal{W})$, consider the transition $\mathcal{W} \xrightarrow{\{a\}} \mathcal{W}'$. If the set of world states $\mathcal{W}'$ has not appeared earlier in the path from $\mathcal{W}^0$ to $\mathcal{W}$, add new belief vertex $\mathcal{W}'$ connected via an edge bearing $\{a\}$. Otherwise, add transition from $\mathcal{W}$ to a vertex $\mathcal{W}_{\text{dummy}}$ to avoid expanding the same belief vertex multiple times.

▷ *Expanding observation vertex* $\mathcal{W}$: Let all possible observations at the states in $\mathcal{W}$ be $Y(\mathcal{W})$, i.e., $Y(\mathcal{W}) = \cup_{w\in\mathcal{W}} A(w)$. As before, construct a transition from $\mathcal{W}$ to $\mathcal{W}'$ if $\mathcal{W}'$ is new, or to $\mathcal{W}_{\text{dummy}}$ otherwise. But now do this, not just the singletons, but for every $G \subseteq Y(\mathcal{W})$.

▷ *Goals in the tree*: Mark $\mathcal{W}$ a goal state, if $\mathcal{W} \subseteq V_{\text{goal}}$.

The belief tree is finite. Any sensor map and goal-achieving plan are a subtree that satisfies the following:

(i) *Goals are achieved:* the leaf vertices in the subtree are all in the goal region;

(ii) *Readiness to receive all observations:* the outgoing labels at a particular observation vertex in the subtree cover all outgoing events in the original belief tree;

(iii) *Discernment is consistent:* the subset of observations in the tree is universal, i.e., if $\{o_1, o_2\}$ appears on any edge of the subtree, then it will appear at every belief vertex whose outgoing events contains both o_1 and o_2.

5.3.2 Searching for sensor designs and plans jointly

Next, we search this structure for sensor designs and plans jointly, returning all appropriate sensor maps. While the tree is constructed from the root down, this search bubbles from the leaves back upwards.

For each belief vertex $\mathcal{W}$, we will maintain a list of covers, denoted by $\mathbb{L}(\mathcal{W})$, to record all the appropriate observation covers in the subtree. When $\mathcal{W}$ is in the goal region, there are no constraints on sensor maps from its subtree. Hence, we create a new symbol $\epsilon \notin Y$, and initialize its cover list to $\mathbb{L}(\mathcal{W}) = [\![\{\epsilon\}]\!]$. This will make it compatible with any cover when integrating with the goal-achieving sensor covers in a bottom-up manner. For any non-goal belief vertex $\mathcal{W}^p$ ('p' stands for parent), we will construct its cover list from its children. Let the outgoing events be $\{G_1, G_2, \ldots, G_m\}$ and the corresponding child vertices be $\{\mathcal{W}^c_1, \mathcal{W}^c_2, \ldots, \mathcal{W}^c_m\}$ ('c' for child). Then we have:

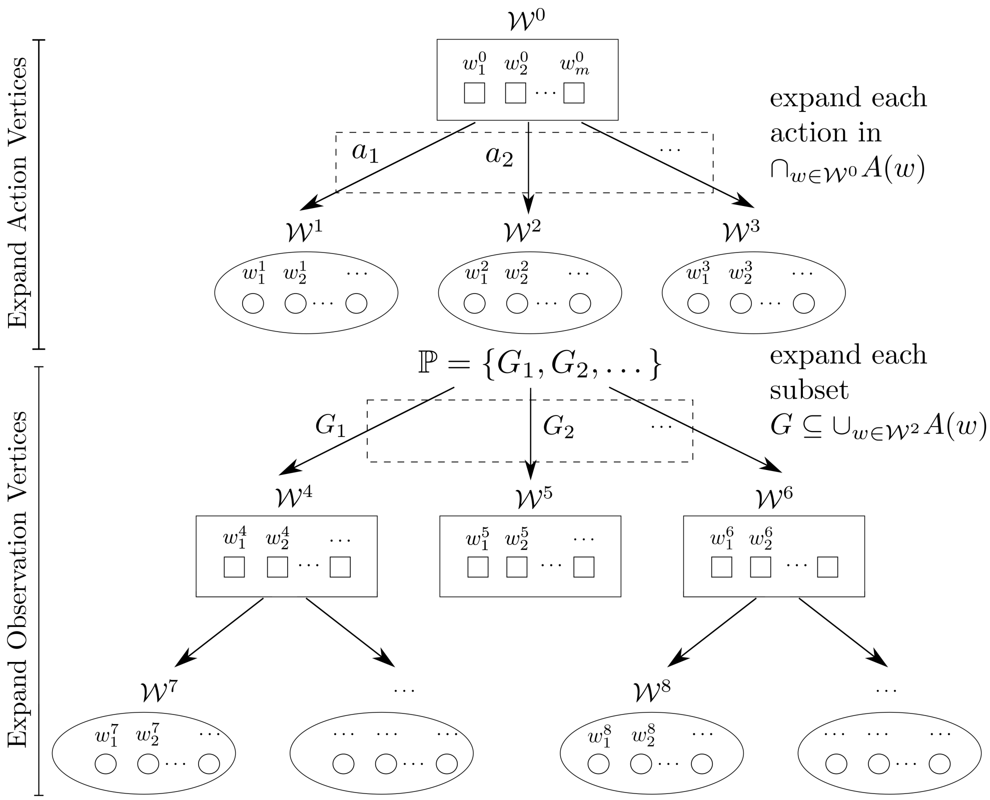


Figure 5.3: The robot's belief tree. Action and observation vertices, visualized as boxes and circles respectively, have different expansions.

- If $\mathcal{W}^p$ is an action vertex, then each cover in any of its children's lists $\mathbb{L}(\mathcal{W}^c_i)$ is a valid one for $\mathcal{W}^p$ (under a particular action choice), i.e., $\mathbb{L}(\mathcal{W}^p) = \cup_{1\le i\le m}\mathbb{L}(\mathcal{W}^c_i)$.
- If $\mathcal{W}^p$ is an observation vertex, we must consider the combinations from $\{G_1, G_2, \dots, G_m\}$ that nevertheless cover $Y(\mathcal{W}^p)$. Let K denote one such combination, then $K = \{G_{k_1}, G_{k_2}, \dots, G_{k_\ell}\}$, where $k_j \in \{1, 2, \dots, m\}$ and $\cup_{1\le j\le \ell} G_{k_j} = Y(\mathcal{W}^p)$. Each edge labeled with G_{k_j} gives a child vertex $\mathcal{W}^c_{k_j}$, where that child has a cover list $\mathbb{L}(\mathcal{W}^c_{k_j})$ modeling the sensors that can reach the goal from $\mathcal{W}^c_{k_j}$. For a given combination K, representing a set of sensor readings, we want to find all sensor maps, denoted as $\mathbb{L}_K$, that can generate K when projected to $d(K)$, and is goal-achieving for the subtrees starting from each child vertex $\mathcal{W}^c_{k_j}$. This is realized via intersection operations:

$$\mathbb{L}_K = \cup_{C_1\in\mathbb{L}(\mathcal{W}^c_{k_1}),\dots,C_n\in\mathbb{L}(\mathcal{W}^c_{k_n})} K \sqcap C_1 \sqcap \dots C_{m-1} \sqcap C_m.$$

The $\sqcap$ operation guarantees the universality of the subsets in the resulting cover. Let $\mathbb{C}$ be the set of all such combinations, such that their labels cover $Y(\mathcal{W}^p)$. Each combination $K \in \mathbb{C}$ gives a list of covers for the parent vertex. So we update $\mathbb{L}(\mathcal{W}^p)$ to value $\cup_{K\in\mathbb{C}}\mathbb{L}_K$.

By propagating the list of covers from the goal vertices back upward until the initial belief vertex, we are able to obtain all the covers from $\mathbb{L}(\mathcal{W}^0)$ where there exists some plan for each cover in $\mathbb{L}(\mathcal{W}^0)$ toward the goal.

5.3.3 Compact representation with upper covers

In the data structure above, we need to maintain a list of covers $\mathbb{L}(\mathcal{W})$ for each belief vertex $\mathcal{W}$. The list can grow very large. Luckily, we only need to maintain the largest covers among the ones with the same domain. Every subset of such covers is also a valid solution, so long as it is a proper cover.

Theorem 5.3. If C is an observation cover in the solution of JPSD, then for any $C' \subseteq C$, such that $d(C') = d(C)$, there exists a plan achieving the goal.

This theorem can be proved by showing that the subtree without edges bearing subsets of events in $C' \setminus C$, still has all leaf vertices as goals and sensor map as a valid cover.

Definition 5.5 (upper cover)**.** Let $\mathbb{C}$ be a list of observation covers, C is an *upper cover* in $\mathbb{C}$ if there does not exist any cover $C' \in \mathbb{C}$, such that $C' \supsetneq C$ and $d(C') = d(C)$.

According to Theorem 5.3, we only need to maintain a set of upper covers in each $\mathbb{L}(\mathcal{W})$.

5.3.4 Empirical explorations of sensor maps

We implemented the algorithms in Python to search for all sensor map solutions for the problem displayed in Fig. 5.4, a modified version of the (Fig. 5.1) motivating example: a robot, initially located at 1 or 2, moves to the charging station. The robot can only move forward one or two steps, turn left or right at the location 5 or the corner 6. The robot must avoid bumping into the walls of the four offices A–D and also the stairs. It must, thus, obtain information from its sensors to

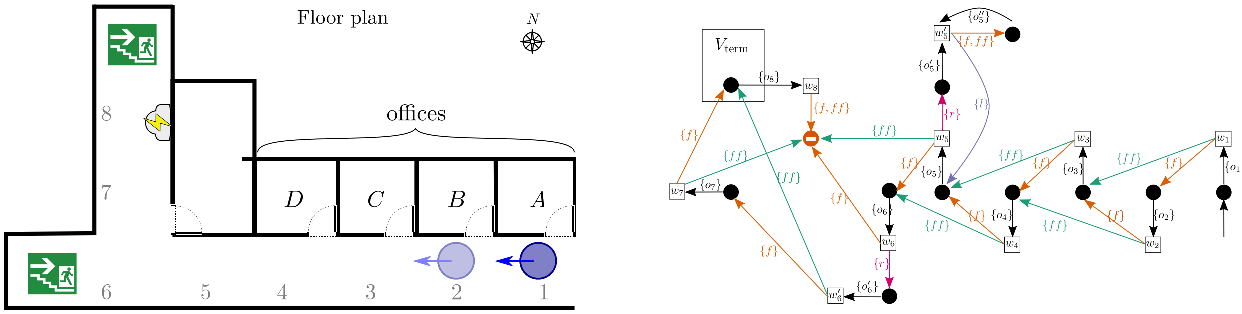


Figure 5.4: A robot moves toward the charging station, while avoiding stairs. The figure below shows the p-graph of this planning problem.

reduce its uncertainty. To realize this scenario, we construct a world p-graph with 22 states and 11 observations.

The algorithm outputs an upper cover with 767 entries. By enumerating all subsets of the upper cover that covers all the observations (Theorem 5.3), an enormous number of sensor maps are produced. Among these, several are directly recognizable sensors. For example, they include a sensor map distinguishing every pair of positions, describing a GPS device. The sensor partitioning the situations into those before and after bumping into walls, could be realized as a contact sensor.

Naturally, some of the sensor maps are inscrutable and there are others for which no known hardware implementation could be discerned. For instance, the sensor isolating cell 5 when facing west from cell 7 when facing north (e.g., a distance sensor won't work). This motivates the next section.

5.4 Structure and fabrication constraints for realizable sensors

The covers found via the preceding approach might be thought of as a sort of 'free object', on which we may now impose additional constraints. Specifically we're interested in including constraints that help model aspects pertinent to realizable sensors.

5.4.1 Sensor map properties

We will start with the following properties:

Property 5.1 (Partition)**.** Cover $C = \{G_1, G_2, \ldots, G_n\}$ is a *partition*, if $G_i \cap G_j = \varnothing$ for any

$i, j \in \{1, \ldots, n\}, i \neq j$.

The label map for the camera in Fig. 5.1 is a partition, as it divides the space into red, gray and white locations.

The next concept of interest is a notion of contiguousness, but we need a more basic structure first.

Definition 5.6 (Neighbor)**.** Relation $N \subseteq Y \times Y$, written $y_1 N y_2$, is a *neighbor relation* if it is reflexive and commutative.

Property 5.2 (Contiguous)**.** With neighbor relation N, then $\mathcal{C}$ is the largest contiguous cover if (1) $\forall y \in Y$, $\{y\} \in \mathcal{C}$; (2) $\forall G_1, G_2 \in \mathcal{C}$, $G_1 \cup G_2 \in \mathcal{C} \iff \exists y_1 \in G_1, \exists y_2 \in G_2$, such that $y_1 N y_2$. A given cover C is *contiguous*, if $C \subseteq \mathcal{C}$.

The distance sensor in Fig. 5.1 has a contiguous sensor map for the obvious neighbor notion, since its noise distribution is contiguous.

Property 5.3 (output)**.** Cover C is k*-outputting*, if $|C| = k$.

The cardinality of the sensor keeps track of total number of output readings, a sort of notion of dynamic range.

Property 5.4 (overlap)**.** A cover $C = \{G_1, G_2, \ldots, G_n\}$ is k*-overlapping*, if $\forall i, j \in \{1, \ldots, n\}$ and $i \neq j$, $|G_i \cap G_j| \leq k$.

This is a generalization of the partition property, quantifying how much readings bleed into one another.

Property 5.5 (width)**.** Cover $C = \{G_1, G_2, \ldots, G_n\}$ is k*-wide* (or, has width k), if $\forall 1 \leq i \leq n$, $|G_i| = k$.

The width of a cover gives a notion of precision, a sense of the volume of noise, describing the number of events that could account for a single sensor reading.

The properties above may also be combined in specifying constraints on sensor maps.

All of the properties can be used either in (1) reducing the sets generated, or in (2) filtering to discard those which violate the constraints, as operators are applied. For instance, in the first case, if searching for partitions only, then partitions exclusively need be computed—a process easier to write and faster to execute than the full cover case.

5.4.2 Empirical search for sensors under fabrication constraints

We included the properties described above in our implementation and examined in the following scenario. A robot moves along a cyclic track toward some goal, marked by a star. The robot can move forward or backward at different speeds at different parts of the track, which discretizes the track into 6 segments $\{s_1, s_2, \ldots, s_6\}$ as shown in Fig. 5.5. The angular range of segment s_i is denoted as $\{o_i\}$ for $i \in \{1, \ldots, 4\}$, and $\{o_i, o\}$ for $i \in \{5, 6\}$. The overlap o is the common angular range for both s_5 and s_6, arising from the kink. Now, the set of all observations is $Y = \{o, o_1, o_2, \ldots, o_6\}$, where each observation represents a range of angles.[†] The neighbor relationship of these observations inherits from the circular neighbor relationship of their angles as shown in the figure. Considering only forward or backward actions, the robot, initially located at s_1 or s_3, must move to reach the goal s_5. To achieve this, the robot has to reduce its uncertainty, and it does this via a VHF omnidirectional range (VOR) sensor. As shown on the right-hand side of Fig. 5.5, the sensor measures the angular information via a timer. The specification of the timer determines the properties of the sensor map. Suppose we have a timer with no noise, then it gives 1-overlapping contiguous sensor maps, such as $[\{o_2, o_3, o_4, o_5\}, \{o_5, o\}, \{o, o_6, o_1\}]$. There are only 2183 such sensor maps. But with a noisy timer, it generates contiguous sensor maps, which leads to 235 807 observation covers.

Consideration of the scenario above leads to the following:

Proposition 5.1. A noiseless sensor taking measurements on a continuous or non-continuous space always gives a 1-overlapping sensor map under discretization.

Proof. When there is no noise for the sensor, the sensor map partitions the original continuous

[†] Previously we pointed out that Y was finite; this is still true, though the elements it contains are themselves infinite sets.

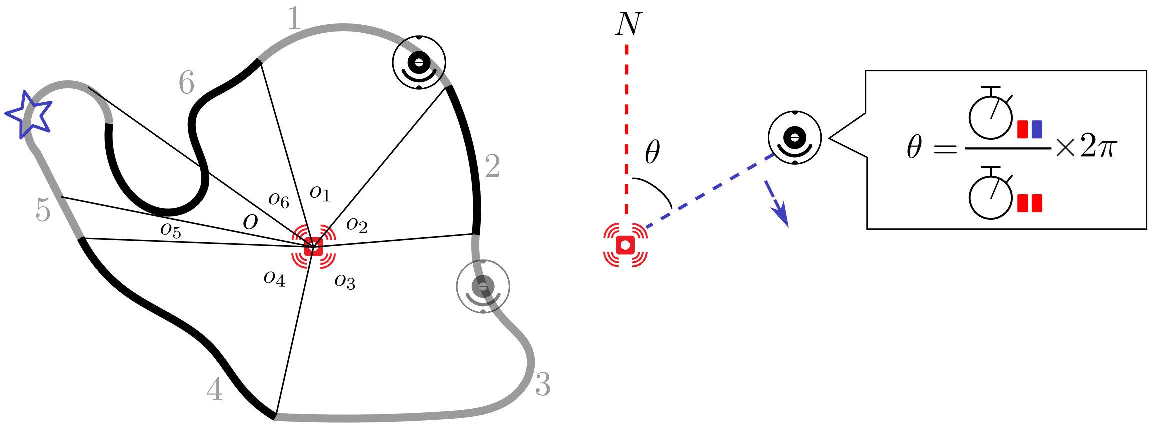


Figure 5.5: A robot with a sensor equipped to determine angles moves from its initial position toward the goal along a cyclic track. The sensor is realized by a VOR-like beacon at the center, a photo-electric sensor and a timer on the robot. The beacon has a unidirectional blue light rotating at a fast constant angular velocity, which is so fast that can be neglected with respect to robot's movement. It also emits an omnidirectional red light when the blue light points North. The robot can determine angular information by timing the difference between seeing red-red and red-blue flashes.

or non-continuous measurement space. The task may only need a coarser discretization of the measurement space. If every boundary of the sensor map is a discretization boundary, then the sensor map is still a partition on the discretized space. If it is not, then there exists a sensor map boundary that falls into one of the discretized observations. That observation is shared by the preimage of only the readings separated by the corresponding sensor map boundary. Hence, the maximum overlap between subsets in the observation cover is 1. □

5.5 Generalization to belief stipulations

Some prior work has examined instances wherein a robot should be stopped from knowing too much due to privacy considerations [1]. In these cases, one may pose constraints on robot's belief; in our previous chapter this was achieved via logical expressions. To search for sensor maps and plans that also satisfy these richer stipulations, the algorithm above needs the following modifications:

- The belief tree should only contain belief vertices satisfying the stipulations, and the dummy vertex. We transition to the dummy if the target belief violates the stipulations.

- We must expand the action vertex in the belief tree over all subsets of actions in the plan instead of just the singleton ones, since none of the singleton actions may transition to the subtree that satisfies the stipulations in each belief vertex.

5.6 Summary

This chapter views robot sensors as generalized information disclosure policies to conflate the information between the world and the robot, and it abstracts sensors as covers, which are generalizations of prior models. It explores the space of all feasible abstract sensors by jointly searching for plans and sensors in the planning problem. A notion of upper cover is proposed to compress the representation and speed the search process. Properties are introduced to express domain-knowledge regarding fabrication constraints for sensors.

6. COMBINATORIAL FILTER MINIMIZATION[*]

In the previous chapters, the world, the plan, and the observer's estimator are represented as graphs. An important aspect of robot design in these problems is to minimize the storage resource used by these structures. This chapter focuses on the problem of minimizing the number of states in estimators, which is termed combinatorial filter minimization. It generalizes the existing notion of combinatorial filter minimization, by defining and examining the vertex-multi-outputting filters and nondeterministic filters. It shows that multiple concepts previously believed to be true about combinatorial filter minimization (and actually conjectured, claimed, or assumed to be) are in fact false, and contributes the first known complete algorithm to solve the deterministic filter minimization problems. In addition, it gives complexity results for a family of combinatorial filter minimization problems, and identifies and discusses the degrees of freedom that separate filter minimization from automata minimization.

6.1 Combinatorial filter minimization

Combinatorial filters or filters are discrete structures used for estimation and inference tasks. In this section, we first give an intuitive example to illustrate combinatorial filters and their minimization problem, then formalize the problem, and correct some prior ideas for filter minimization.

6.1.1 Motivating example

Here is a motivating example: Consider the safari park with vehicle rental service shown in Figure 6.1a. The cars for hire are each equipped with a compass and an intelligent gear shifting system. The compass measures the heading of the vehicle before and after its movement, e.g., 'nw' means that the vehicle was heading north and then turned to face west. The intelligent gear shifting system takes the readings from the compass as input, and automatically shifts gears to abide with

[*] Part of this chapter is reprinted with permission from "Cover Combinatorial Filters and Their Minimization Problem" by Yulin Zhang and Dylan Shell, 2020. *Algorithmic Foundations of Robotics XIV*. 17:90–106. Copyright 2020 by *Springer Nature*.

the speed limit. There are three types of speed limits on roads: (a) between 15 and 30 (gray), (b) slower than 15 (brown and green). Every vehicle is capable of moving with a low gear to drive with a maximum speed 15, and with a high gear to drive between speed 15 and 30.

A naïve gear shifting system satisfying the speed limits is realized by a filter shown in Figure 6.1b: each vertex represents a system state, each edge represents the state transition, with the label on the edges representing the readings from the compass. A vertex is colored gray if the system outputs high gear, or colored green if it outputs low gear. Filer minimization is to find a minimal filter, like the one shown in Figure 6.1c, that realizes appropriate behavior but with fewest states.

A natural way to proceed is by first constructing a discrete state-transition system (such as in Figure 6.1b) using the problem description as a basis. Then, the next step is to apply some algorithm capable of compressing it. One commonly used approach for filter reduction is to merge the states in the input filter. But are merging operations sufficient to find a minimizer for any input filter? We will answer this question in this section.

6.1.2 Definition of combinatorial filter minimization

We firstly introduce the notion of p-filter:

Definition 6.1 (procrustean filter [59])**.** A *procrustean filter*, *p-filter* or *filter* for short, is a tuple (V, V_0, Y, τ, C, c) with:

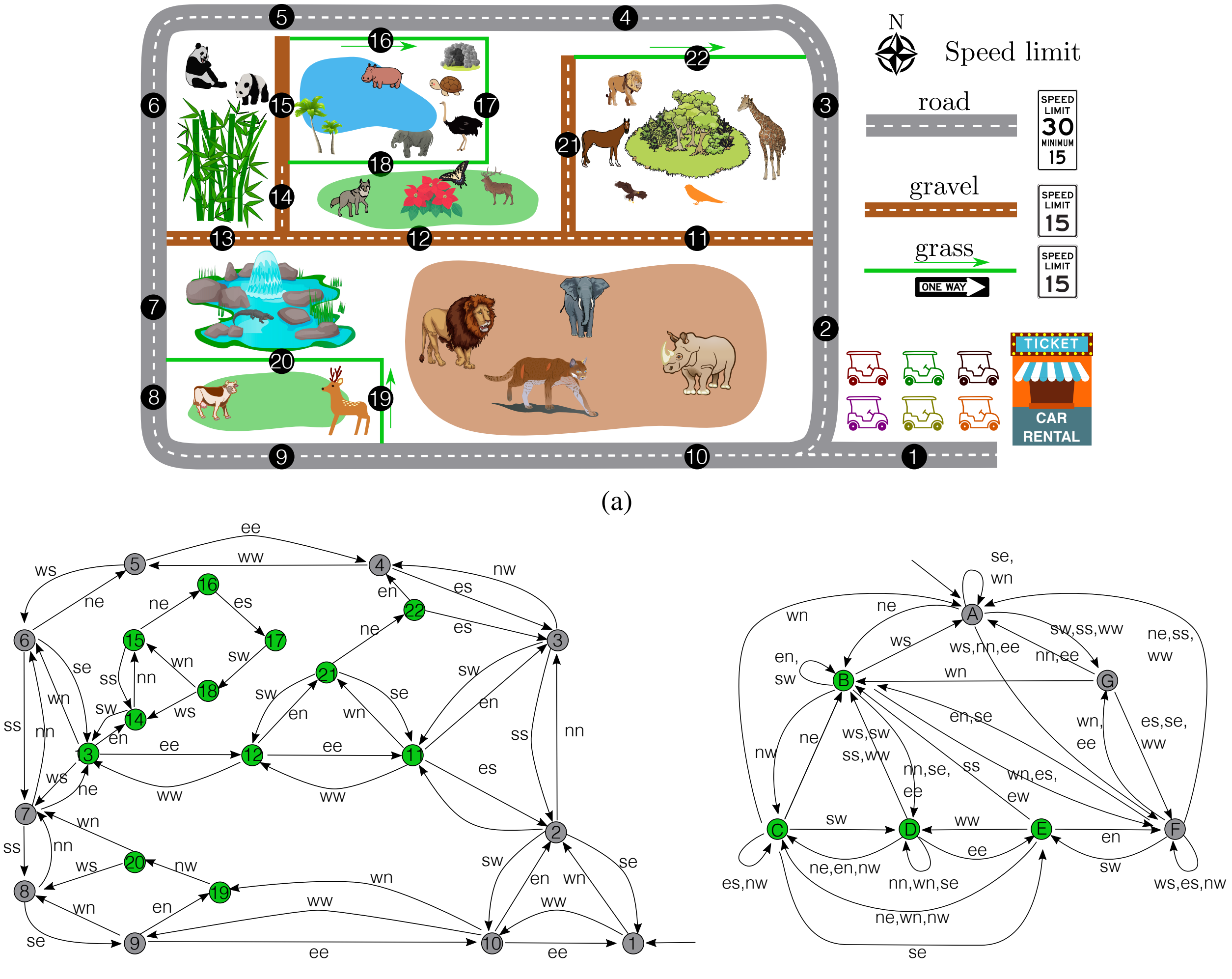
1) a finite set of states V, a non-empty initial set of states $V_0 \subseteq V$, and a set of possible observations Y,

2) a transition function $\tau : V \times V \to 2^Y$,

3) a set C, which we call the output space, and

4) an output function $c : V \to 2^C \setminus \{\varnothing\}$.

The states, initial states and observations for p-filter $\texttt{F}$ will be denoted $V(\texttt{F})$, $V_0(\texttt{F})$ and $Y(\texttt{F})$. Without loss of generality, we will also treat a p-filter as a graph with states as its vertices and transitions as directed edges.



(a)

(b)

(c)

Figure 6.1: (a) A safari park with vehicles for hire. The vehicles are equipped with an intelligent gear shifting system automatically shifts gears to satisfy the speed limit according to its compass readings. (b) A naïve filter to implement the intelligent gear shifting system. (c) A minimal filter for the intelligent gear shifting system.

A sequence of observations can be traced on the p-filter:

Definition 6.2 (reached)**.** Given any p-filter $\mathtt{F} = (V, V_0, Y, \tau, C, c)$, a sequence of observations $s = y_1 \dots y_n \in Y^*$, and states $w_0, w_n \in V$, we say that w_n is a state *reached by* some sequence s from w_0 in $\mathtt{F}$ (or s reaches w_n from w_0), if there exists a sequence of states $w_0, \dots, w_n$ in $\mathtt{F}$, such that $\forall i \in \{1, \dots, n\}, y_i \in \tau(w_{i-1}, w_i)$. We denote the set of all states reached by s from state w_0 in $\mathtt{F}$ as $\mathcal{V}_{\mathtt{F}}(w_0, s)$. For simplicity, we use $\mathcal{V}_{\mathtt{F}}(s)$, without the subscript, to denote the set of all states reached when starting from any state in V_0, i.e., $\mathcal{V}_{\texttt{F}}(s) = \cup_{v_0 \in V_0} \mathcal{V}_{\texttt{F}}(v_0, s)$. Note that $\mathcal{V}_{\texttt{F}}(s) = \varnothing$ holds only when sequence s *crashes* in $\texttt{F}$ starting from V_0.

For convenience, we will denote the set of sequences reaching state $v \in V$ from some initial state by $\mathcal{S}_v^{\texttt{F}}$.

Definition 6.3 (extensions, executions and interaction language)**.** An *extension* of a state v on a p-filter $\texttt{F}$ is a finite sequence of observations s that does not crash when traced from v, i.e., $\mathcal{V}_{\texttt{F}}(v, s) \neq \varnothing$. An *extension* of any initial state $v_0 \in V_0(\texttt{F})$ is also called an *execution* or a *string* on $\texttt{F}$. The set of all extensions of a state v on $\texttt{F}$ is called the *extensions* of v, written as $\mathcal{L}_{\texttt{F}}^{e}(v)$. The extensions of all initial vertices on $\texttt{F}$ is also called the *interaction language* (or, briefly, just *language*) of $\texttt{F}$, and is written $\mathcal{L}^{I}(\texttt{F}) = \cup_{v_0 \in V_0(\texttt{F})} \mathcal{L}_{\texttt{F}}^{e}(v_0)$.

Note in particular that the empty string ϵ belongs to the extensions of any state on the filter, and belongs to the language of the filter as well.

Definition 6.4 (vertex single-outputting)**.** A filter $\texttt{F} = (V, V_0, Y, \tau, C, c)$ is *vertex single-outputting* if $|C| = 1$. Otherwise, it is *vertex multi-outputting*.

When minimizing some filter, we are interested in reduced filters that simulate the given filter in terms of outputs on its strings:

Definition 6.5 (output simulating)**.** Let $\texttt{F}$ and $\texttt{F}'$ be two filters, then $\texttt{F}'$ *output simulates* $\texttt{F}$ if the following properties hold: (i) language inclusion: $\mathcal{L}^{I}(\texttt{F}) \subseteq \mathcal{L}^{I}(\texttt{F}')$; (ii) output consistency: $\forall s \in \mathcal{L}^{I}(\texttt{F}), \mathcal{C}(\texttt{F}', s) \subseteq \mathcal{C}(\texttt{F}, s)$.

Intuitively, this requires that $\texttt{F}'$ be capable of processing all the inputs which $\texttt{F}$ can, and produce outputs that $\texttt{F}$ could. The input set is no smaller; the set of outputs no larger.

We want to search for a minimal filter that output simulates the original filter:

Problem: Vertex Single-outputting Filter Minimization (VSO-FDM)

Input: A deterministic vertex single-outputting filter $\texttt{F}$.

Output: A deterministic filter $\texttt{F}^{\dagger}$ with fewest states, such that $\texttt{F}^{\dagger}$ output simulates $\texttt{F}$.

We use ‘VSO’ for vertex single output p-filters, ‘D’ for deterministic input and output, ‘M’ for minimization.

Theorem 6.1. VSO-FDM is NP-Complete (Theorem 2 in [2]).

6.1.3 Revisiting prior ideas for filter minimization

In this section, we revisit some prior ideas for filter minimization, and correct them with the new ones shown in Figure 6.2.

The original question of minimizing state in filtering is first alluded to by LaValle [67] as an open problem, who suggested that it is ‘similar to Nerode equivalence classes’. The problem of filter reduction, i.e., VSO-FDM in our terms, was formalized and shown to differ in complexity class from the automata problem in [2]. That paper also proposed a heuristic algorithm, which served as a starting point for subsequent work. The heuristic algorithm uses conflict graphs to designate which vertices cannot be merged (are conflicting). It starts with a conflict relation where two vertices are in conflict when they have different outputs, then iteratively refines the conflict relation. Refinement has two steps: (i) *introducing edges*: two vertices are determined to be conflicting or not via a graph coloring subroutine, and edges are added between conflicting vertices; (ii) *propagating conflicts upstream*: filter states are marked as conflicted when they transition to conflicted states under the same observation. An example input filter, shown in Figure 6.3a, is reduced by following this procedure, which is depicted step-by-step in Figures 6.3b–6.3e.

A conjecture in [2] was that this algorithm is guaranteed to find a minimal filter if the graph coloring subroutine gives a minimal coloring. (Put another way: the inexactness in arriving at a minimal filter can be traced to the graph coloring giving a suboptimal result.) But this conjecture was later proved to be false by Saberifar et al. [63]. They show an instance where there exist multiple distinct optimal solutions to the graph coloring subproblem, only a strict subset of which lead to the minimal filter. One might naturally ask, and indeed they do ask, the question of whether some optimal coloring is sufficient to arrive at the optimal filter. Following along these lines (see §7.3 in [63]), one might sharpen the original conjecture of [2] to give the following statement:

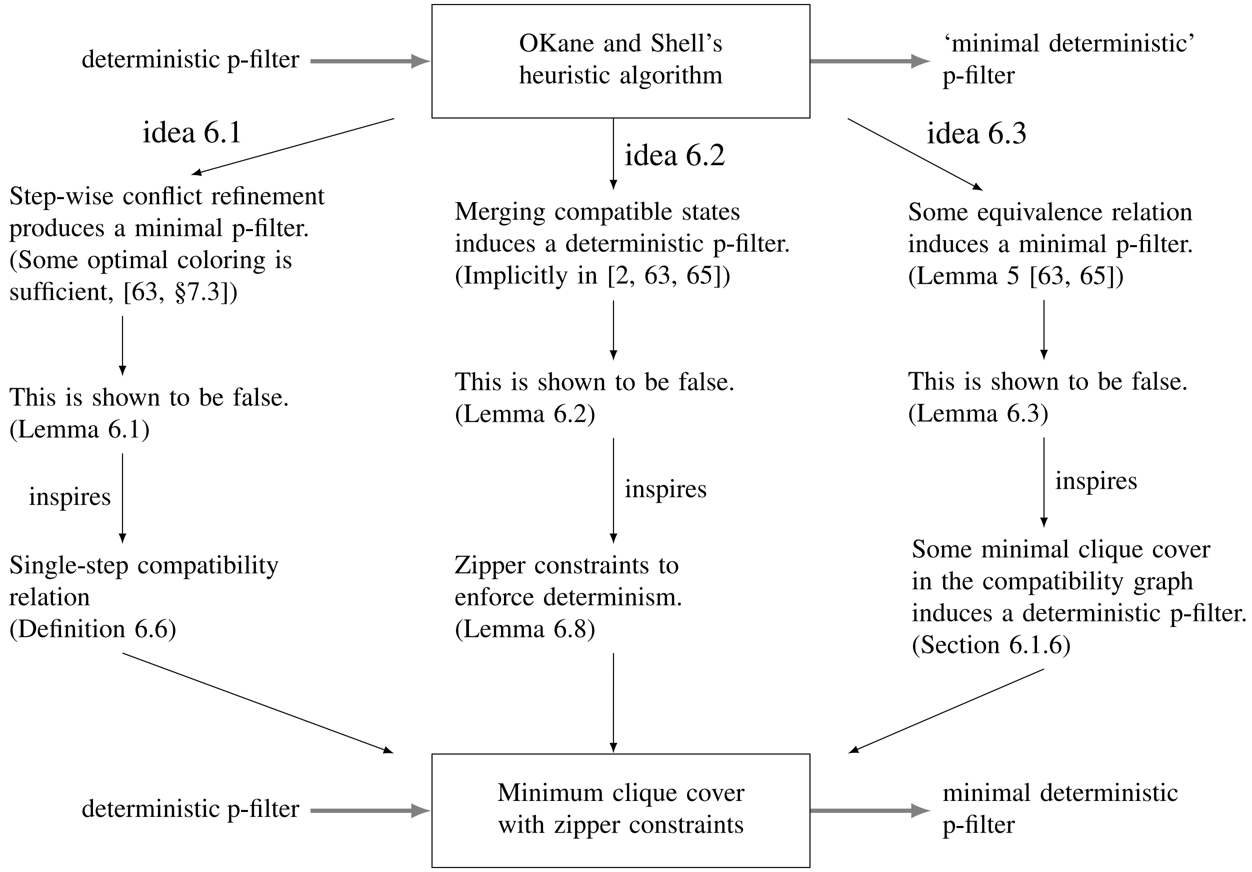

deterministic p-filter
OKane and Shell's heuristic algorithm
'minimal deterministic' p-filter
idea 6.1
idea 6.2
idea 6.3
Step-wise conflict refinement produces a minimal p-filter. (Some optimal coloring is sufficient, [63, §7.3])
Merging compatible states induces a deterministic p-filter. (Implicitly in [2, 63, 65])
Some equivalence relation induces a minimal p-filter. (Lemma 5 [63, 65])
This is shown to be false. (Lemma 6.1)
This is shown to be false. (Lemma 6.2)
This is shown to be false. (Lemma 6.3)
inspires
inspires
inspires
Single-step compatibility relation (Definition 6.6)
Zipper constraints to enforce determinism. (Lemma 6.8)
Some minimal clique cover in the compatibility graph induces a deterministic p-filter. (Section 6.1.6)
deterministic p-filter
Minimum clique cover with zipper constraints
minimal deterministic p-filter

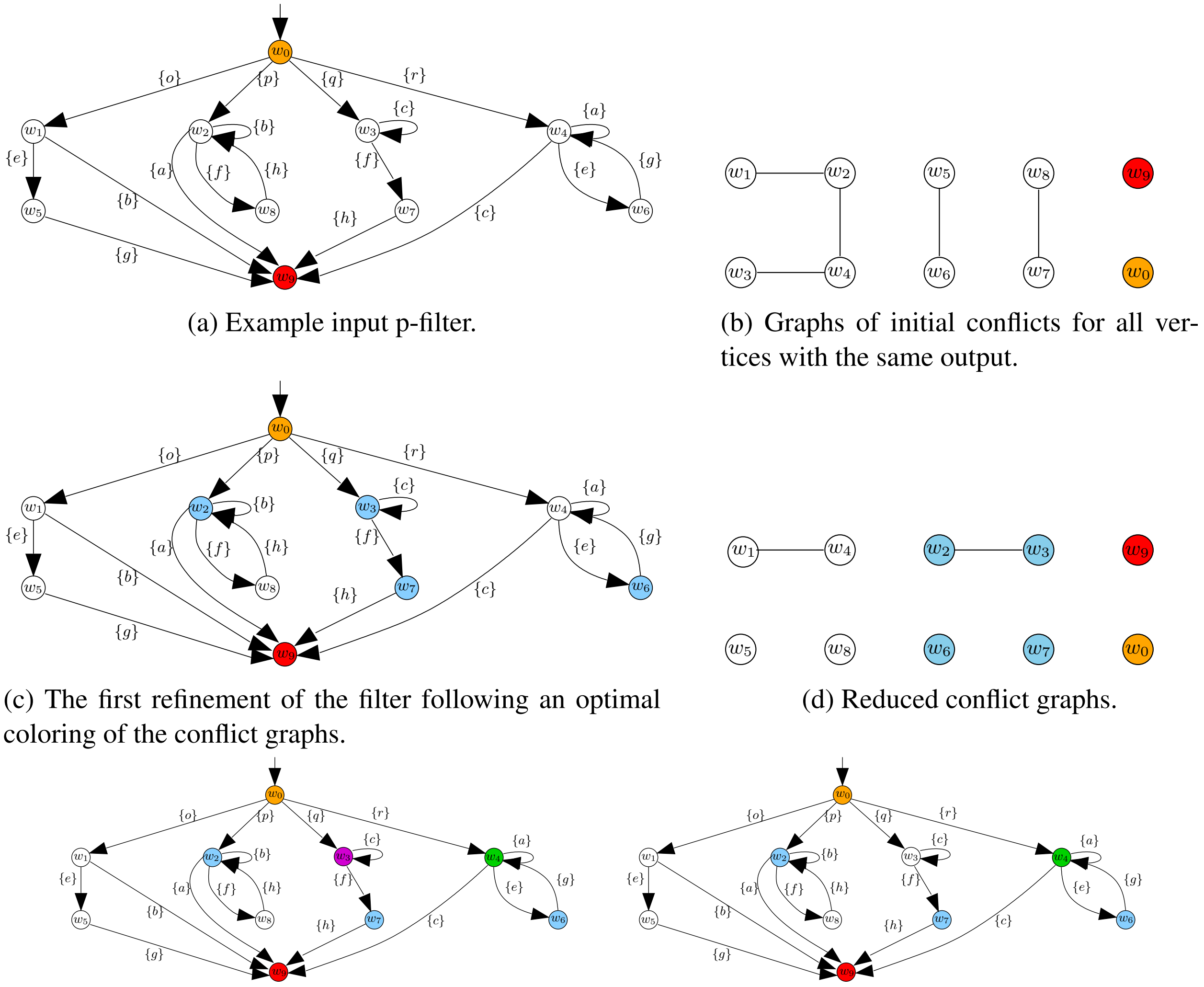
Figure 6.2: This roadmap shows the provenance of those insights in terms of previous ideas in VSO-FDM, which we examine carefully.

Idea 6.1. In the step-wise conflict refinement procedure of O'Kane and Shell's heuristic algorithm [2], some optimal coloring is sufficient to guarantee a minimal filter for VSO-FDM.

Lemma 6.1. Idea 6.1 is false.

Proof. This is simply shown with a counterexample. Consider the problem of minimizing the input filter shown in Figure 6.3a, the heuristic algorithm will first initialize the colors of the vertices with their output. Next, it identifies the vertices that disagree on the outputs of extensions with length 1 as shown in Figure 6.3b, and then refines the colors of the vertices as shown in Figure 6.3c following a minimal graph coloring solution on the conflict graph. Then it further identifies the conflicts on extensions with length 2, via the conflict graph shown in Figure 6.3d, and the vertex colors are further refined as shown in Figure 6.3e. Now, no further conflicts can be found. A filter, with 6 states, is then obtained by merging the states with the same color. However, there exists a minimal filter, with 5 states, shown in Figure 6.3f, that can be found by choosing coloring solution for the conflict graph shown in Figure 6.3b. That coloring is suboptimal. □

This appears to indicate a sort of *local optimum* arising via sub-problems associated with incremental (or stepwise) reduction. Since optimal colorings for individual steps are seen to be



(a) Example input p-filter.

(b) Graphs of initial conflicts for all vertices with the same output.

(c) The first refinement of the filter following an optimal coloring of the conflict graphs.

(d) Reduced conflict graphs.

(e) Second refinement of filter following an optimal coloring of the conflict graphs.

(f) The coloring that gives the minimal filter.

Figure 6.3: An example run of the heuristic minimization algorithm in [2] (a)–(e). This particular input also shows that optimal step-wise conflict refinement may fail to yield a minimal filter (Lemma 6.1).

insufficient to guarantee a minimal filter, to find a minimal filter, we would have to enumerate all colorings (suboptimal or otherwise) at each iteration. That is, however, essentially a brute force algorithm. A more informed approach is to compute implications of conflicts more *globally*, in a way that doesn't depend on earlier merger decisions. In our algorithm, rather than tracking vertices which are in conflict, we introduce a new notion of compatibility between vertices that may be merged. This notion differs from the one recursively defined in [65], as our compatibility relation is computed in one fell swoop, before making any decisions to reduce the filter:

Definition 6.6 (compatibility)**.** Let $\mathtt{F}$ be a deterministic p-filter. We say a pair of vertices $v, w \in V(\mathtt{F})$ are *compatible*, denoted $v \sim_c w$, if they agree on the outputs of all their extensions, i.e., $\forall s \in \mathcal{L}^e_{\mathtt{F}}(v) \cap \mathcal{L}^e_{\mathtt{F}}(w), \forall v' \in \mathcal{V}_{\mathtt{F}}(v, s), \forall w' \in \mathcal{V}_{\mathtt{F}}(w, s), c(v') = c(w')$. A *mutually compatible* set consists of vertices where all pairs are compatible.

Via this notion of compatibility, we get an undirected compatibility graph:

Definition 6.7 (compatibility graph)**.** Given a deterministic filter $\mathtt{F}$, its compatibility graph $\mathcal{K}(\mathtt{F})$ is an unlabeled undirected graph constructed by creating a vertex associated with each state in $\mathtt{F}$, and building an edge between the pair of vertices associated with two compatible states.

This compatibility graph can be constructed in polynomial time. As every filter state and associated compatibility graph state are one-to-one, to simplify notation we'll use the same symbol for both and context to resolve any ambiguity.

The second idea relates to the type of the output one obtains after merging states that are compatible or not in conflict. Importantly, the filter minimization problem VSO-FDM requires one to give a minimal filter which is deterministic.

Idea 6.2. By merging the states that are compatible, the heuristic algorithm always produces a deterministic p-filter.

The definition of the reduction problems within [2, 63, 65] are specified so as to require that the output obtained be deterministic. But this postcondition is never shown or formally established. In fact, it does not always hold.

Lemma 6.2. Idea 6.2 is false.

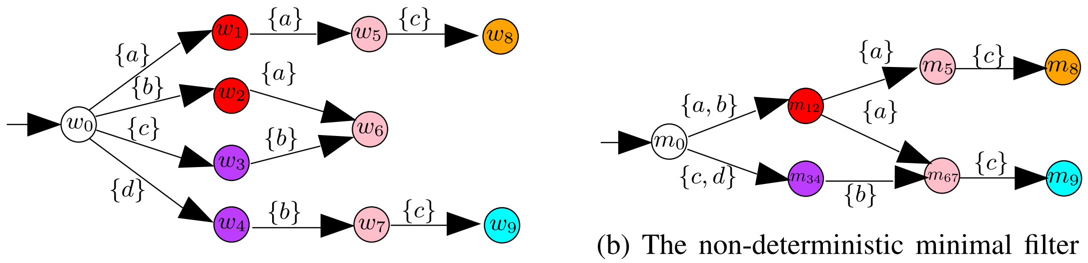
Proof. We show that the existing algorithm may produce a non-deterministic filter, which does not output simulate the input filter, and is thus not a valid solution. Consider the filter shown in Figure 6.4a as an input. The vertices with the same color are compatible with each other, with the following exception for w_5, w_6 and w_7. Vertex w_5 is compatible with w_6, vertex w_6 is compatible with w_7, but w_5 is not compatible with w_7. The minimal filter found by the existing algorithm is shown in Figure 6.4b. The string aac suffices to shows the non-determinism, reaching both orange and cyan vertices. It fails to output simulate the input because cyan should never be produced. □

If determinism can't be taken for granted, we might constrain the output to ensure the result will be a deterministic filter. To do this, we introduce a zipper constraint when merging compatible states:

Definition 6.8 (zipper constraint)**.** In the compatibility graph $\mathtt{G} = \mathcal{K}(\mathtt{F})$ of filter $\mathtt{F}$, if there exists a set of mutually compatible states $U = \{u_1, u_2, \ldots, u_n\}$, then they can only be selected to be merged if they always transition to a set of states that are also selected to be merged. For any sets of mutually compatible states $U, W \subseteq V(\mathtt{G})$ and some observation y, we create a *zipper constraint* expressed as a pair $(U, W)_y$ if $W = \{w \in V(\mathtt{G}) \mid y \in \tau(u, w) \text{ for some } u \in U\}$. We denote the set of all zipper constraints on compatibility graph $\mathtt{G} = \mathcal{K}(\mathtt{F})$ by $\mathscr{Z}(\mathtt{F})$.

The zipper constraints for the input filter shown in Figure 6.4a consist of $(\{w_1, w_2\}, \{w_5, w_6\})_a$ and $(\{w_3, w_4\}, \{w_6, w_7\})_b$. Constraint $(\{w_1, w_2\}, \{w_5, w_6\})_a$ is interpreted as: if w_1 and w_2 are selected for merger, then w_5 and w_6 (reached under a) should also be merged. We call it a zipper constraint owing to the resemblance to a zipper fastener: merger of two earlier states, merges (i.e., pulls together) later states. In the worst case, the number of zipper constraints can be exponential in the size of the input filter.

A third idea is used by O'Kane and Shell's heuristic algorithm and is also stated, rather more explicitly, by Saberifar et al. (see Lemma 5 in [63] and Lemma 5 in [65]). It indicates that we can obtain a minimal filter via merging operations on the compatible states, which yields a special



(a) An input filter.

(b) The non-deterministic minimal filter found by the existing algorithm.

Figure 6.4: A counterexample showing how compatible merges may introduce non-determinism (Lemma 6.2). The input filter also illustrates a violation of the presumption that an equivalence relation can yield a minimum filter (Lemma 6.3).

class of filter minimization problems. For this class, recent work has exploited integer linear programming techniques to compute exact and feasible solutions efficiently [64].

Idea 6.3. Some equivalence relation induces a minimal filter in VSO-FDM.

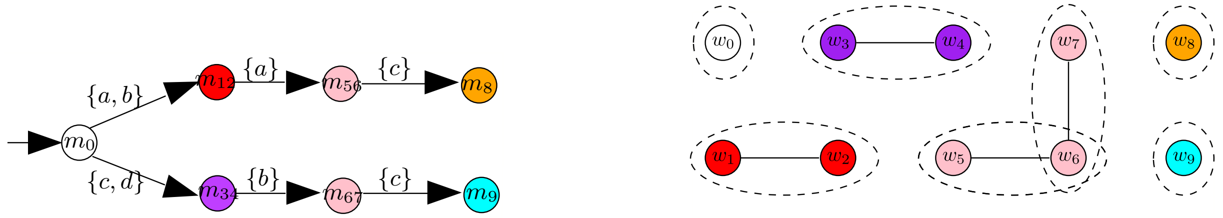
Before examining this, we rigorously define the notion of an induced relation:

Definition 6.9 (induced relation)**.** Given a filter $\mathtt{F}$ and another filter $\mathtt{F}'$, if $\mathtt{F}'$ output simulates $\mathtt{F}$, then $\mathtt{F}'$ induces a relation $R \subseteq V(\mathtt{F}) \times V(\mathtt{F})$, where $(v, w) \in R$ if and only if there exists a vertex $v' \in V(\mathtt{F}')$ such that $\mathcal{S}_v^{\mathtt{F}} \cap \mathcal{S}_{v'}^{\mathtt{F}'} \neq \varnothing$ and $\mathcal{S}_w^{\mathtt{F}} \cap \mathcal{S}_{v'}^{\mathtt{F}'} \neq \varnothing$. We also say that v and w corresponds to state v'.

Lemma 6.3. Idea 6.3 is false.

Proof. It is enough to scrutinize the previous counterexample closely. The minimization problem VSO-FDM for the input filter shown in Figure 6.4a, is shown in Figure 6.5a. It is obtained by (i) splitting vertex w_6 into an upper part reached by a and a lower part reached by b, (ii) merging the upper part of w_6 with w_5, the lower part of w_6 with w_7, and other vertices with those of the same color. This does not induce an equivalence relation, since w_6 corresponds to two different vertices in the minimal filter. □

In light of this, for some filter minimization problems, there may be no quotient operation that produces a minimal filter and an exact algorithm for minimizing filters requires that we look



(a) A minimal filter for Figure 6.4a.

(b) Cliques from the minimal filter.

Figure 6.5: A minimal filter for Figure 6.4a and its induced cliques.

beyond equivalence relations.

Some strings that reach a single state in an input filter may reach multiple states in a minimal p-filter (e.g., ba and cb on Figure 6.4a and 6.5a). On the other hand, strings that reach different states in the input p-filter may reach the same state in the minimal filter (e.g., a and b on those same filters). We say that a state from the input filter corresponds to a state in the minimal filter if there exists some string reaching both of them and, hence, this correspondence is many-to-many. An important observation is this: for each state s in some hypothetical minimal filter, suppose we collect all those states in the input filter that correspond with s. When we examine the associated states in the compatibility graph for that collection, they must all form a clique. Were it not so, the minimal filter could have more than one output associated for some strings owing to non-determinism. But this causes it to fail to output simulate the input p-filter.

By building the correspondence between the input p-filter in Figure 6.4a and the minimal result in Figure 6.5a, one obtains the set of cliques in the compatibility graph shown visually in Figure 6.5b. Like previous approaches that make state merges by analyzing the compatibility graph, we interpret each clique as a set of states to be merged into one state in the minimal filter. The clique containing w_3 and w_4 in Figure 6.5b gives rise to m_{34} in the minimal filter in Figure 6.5a (and w_1 and w_2 yields m_{12}, and so on). However, states may further be shared across multiple cliques. We observe that w_6 was merged with w_5 in the minimal filter to give m_{56}, and w_6 also merged with w_7 to give m_{67}. The former has an incoming edge labeled with an a, while the latter has an incoming edge labeled b. The vertex w_6, being shared by multiple cliques, is split into different copies and each copy merged separately.

After firming up and developing these intuitions, the next subsection introduces the concept of a clique cover which enables representation of a search space that includes relations more general than equivalence relations. Based on this new representation, we propose a graph problem use of zipper constraints, and prove it to be equivalent to filter minimization.

6.1.4 A new minimum clique cover problem

To begin, we extend the preceding argument from the compatibility clique associated to single state s, over to all the states in the minimal filter. This leads one to observe that the collection of all cliques for each state in the minimal p-filter forms a clique cover:

Definition 6.10 (induced clique cover)**.** Given a p-filter $\mathtt{F}$ and another p-filter $\mathtt{F}'$, we say that a vertex v in $\mathtt{F}$ *corresponds to* a vertex v'_i in $\mathtt{F}'$ if $\mathcal{S}^{\mathtt{F}}_v \cap \mathcal{S}^{\mathtt{F}'}_{v'_i} \neq \varnothing$. Then, denoting the subset of vertices of $\mathtt{F}$ corresponding to v'_i in $\mathtt{F}'$ with $K_{v'_i} = \{v \in V(\mathtt{F}) \mid v \text{ corresponds to } v'_i\}$, we form the collection of all such sets, $\mathbf{Q}(\mathtt{F}, \mathtt{F}') = \{K_{v'_1}, K_{v'_2}, \ldots, K_{v'_n}\}$, for $i \in \{1, \ldots, n\}$ where $n = |V(\mathtt{F}')|$. When $\mathtt{F}'$ output simulates $\mathtt{F}$, then the $K_{v'_i}$ form cliques in the compatibility graph $\mathcal{K}(\mathtt{F})$. Further, when this collection of sets $\mathbf{Q}(\mathtt{F}, \mathtt{F}')$ covers all vertices in $\mathtt{F}$, i.e., $\cup_{K_i \in \mathbf{Q}(\mathtt{F},\mathtt{F}')} = V(\mathtt{F})$, we say that $\mathbf{Q}(\mathtt{F}, \mathtt{F}')$ is an induced clique cover.

It is worth repeating: the size of filter $\mathtt{F}'$ (in terms of number of vertices) and the size of the induced clique cover (number of sets) are equal.

Without loss of generality, here and henceforth we only consider the p-filter with all vertices reachable from the initial state, since the ones that can never be reached will be deleted during filter minimization anyway.

Each clique of the clique cover represents the states that can be potentially merged. But the zipper constraint, to enforce determinism, requires that the set of vertices to be merged should always transition under the same observation to the ones that can also be merged. Hence, the zipper constraints (of Definition 6.8) can be evaluated across whole covers:

Definition 6.11. A clique cover $\mathbf{K} = \{K_0, K_1, \ldots, K_m\}$ satisfies the set of zipper constraints $\textsc{Zip} = \{(U_1, W_1)_{y_1}, (U_2, W_2)_{y_2}, \ldots\}$, when for every zipper constraint $(U_i, W_i)_{y_i}$, if there exist a clique $K_s \in \mathbf{K}$, such that $U_i \subseteq K_s$, then there exists another clique $K_t \in \mathbf{K}$ such that $W_i \subseteq K_t$.

Now, we have our new graph problem, MCCZC.

Problem: Minimum clique cover with zipper constraints (MCCZC)

Input: A compatibility graph G, a set of zipper constraints $\textsc{Zip}$.

Output: A minimal cardinality clique cover of G satisfying $\textsc{Zip}$.

6.1.5 From minimal clique covers to filters

Given a minimal cover that solves MCCZC, we construct a filter by merging the states in the same clique and choosing edges between these cliques appropriately:

Definition 6.12 (induced filter from a clique cover)**.** Given a clique cover $\mathbf{K}$ on the compatibility graph of deterministic p-filter $\mathtt{F}$, if $\mathbf{K}$ satisfies all the zipper constraints in $\mathscr{Z}(\mathtt{F})$, then it *induces a filter* $\mathtt{F}' = M(\mathtt{F}, \mathbf{K})$ by treating cliques as vertices:

1. Create a new filter $\mathtt{F}' = (V', V_0', Y, \tau', C, c')$ with $|\mathbf{K}|$ vertices, where each vertex v' is associated with a clique $K_{v'}$ in $\mathbf{K}$;

2. Add each vertex v' in $\mathtt{F}'$ to V_0' iff the associated clique contains an initial state in $\mathtt{F}$;

3. The output of every v' in $\mathtt{F}'$, with associated clique $K_{v'}$, is the set of common outputs for all states in $K_{v'}$, i.e., $c'(v') = \cap_{v \in K_{v'}} c(v)$.

4. For any pair of v' and w' in $\mathtt{F}'$, inherit all transitions between states in the cliques of v' and w', i.e., $\tau(v', w') = \cup_{v \in K_{v'}, w \in K_{w'}} \tau(v, w)$.

5. For each vertex v' in $\mathtt{F}'$ with multiple outgoing edges labeled y, keep only the single edge to the vertex w', such that all vertices $K_{v'}$ transition to under y are included in $K_{w'}$. This edge must exist since $\mathbf{K}$ satisfies all $\mathscr{Z}(\mathtt{F})$.

The size of the cover (in terms of number of sets) and size of the induced filter (number of vertices) are equal.

Notice that the earlier intuition is mirrored by this formal construction: states belonging to the same clique are merged when constructing the induced filter; states in multiple cliques are split when we make the edge choice in step 5. Next, we establish that the induced filter indeed supplies the goods:

Lemma 6.4. Given any clique cover $\mathbf{K}$ on the compatibility graph $\mathcal{K}(\mathtt{F})$ of a deterministic p-filter $\mathtt{F}$, if $\mathbf{K}$ satisfies the zipper constraints $\mathscr{Z}(\mathtt{F})$ and covers all vertices of $\mathcal{K}(\mathtt{F})$, then the induced filter $\mathtt{F}' = M(\mathtt{F}, \mathbf{K})$ is deterministic and output simulates $\mathtt{F}$.

Proof. For any string $s \in \mathcal{L}^I(\mathtt{F})$, let the vertex reached by string s in $\mathtt{F}$ be v. Then v must belong at least one clique in $\mathbf{K}$, where all vertices in this clique can be viewed as merged into a new vertex in $\mathtt{F}'$. Hence, s should reach at least one vertex in $\mathtt{F}'$ and this vertex yield the same output $\mathcal{C}(\mathtt{F}, s)$. Since $\mathbf{K}$ satisfies the zipper constraints $\mathscr{Z}(\mathtt{F})$, the induced filter $\mathtt{F}'$ must be deterministic since there is no vertex that has any non-deterministic outgoing edges bearing the same label. Because $\mathtt{F}'$ is deterministic, $\forall s \in \mathcal{L}^I(\mathtt{F})$, s reaches a single vertex in $\mathtt{F}'$. In addition, this vertex in $\mathtt{F}'$ shares the same output $\mathcal{C}(\mathtt{F}, s)$. Therefore, $\mathtt{F}'$ also output simulates $\mathtt{F}$. □

A surprising aspect of the preceding is how the zipper constraints —which are imposed to ensure that a deterministic filter is produced— enforce output-simulating behavior, albeit indirectly, too. One might have expected that this separate property would demand a second type of constraint, but this is not so.

On the other hand, needing to satisfy the zipper constraints of the input filter does not entail the imposition of any gratuitous requirements:

Lemma 6.5. Given any deterministic p-filters $\mathtt{F}$ and $\mathtt{F}'$, if $\mathtt{F}'$ output simulates $\mathtt{F}$, then the induced clique cover $\mathbf{Q}(\mathtt{F}, \mathtt{F}^{'})$ on the compatibility graph of $\mathtt{F}$ satisfies all zipper constraints in $\mathscr{Z}(\mathtt{F})$.

Proof. Suppose that $\mathbf{Q}(\mathtt{F}, \mathtt{F}^{'})$ does not satisfy all zipper constraints in $\mathscr{Z}(\mathtt{F})$. Specifically, let $(U, V)_y \in \mathscr{Z}(\mathtt{F})$ be the zipper constraint that is violated, where each vertex in V transitions from some vertex in U under observation y. Then there exists a clique $K_s \in \mathbf{Q}(\mathtt{F}, \mathtt{F}^{'})$, such that $U \subseteq K_s$, but there is no clique $K_j \in \mathbf{Q}(\mathtt{F}, \mathtt{F}^{'})$ that $V \subseteq K_j$. According to the construction of the induced cover, there exists a vertex $v_s' \in \mathtt{F}^{'}$, such that K_s corresponds to v_s'. For any vertex $u_1 \in K_s$, let $s_1 \in \mathcal{S}^{\mathtt{F}}_{u_1} \cap \mathcal{S}^{\mathtt{F}'}_{v_s'}$. Then $s_1 y$ is also a string in both $\mathtt{F}$ and $\mathtt{F}'$ since u_1 transitions to some vertex v_1 in V under observation y in $\mathtt{F}$ and $\mathtt{F}'$ is output simulating $\mathtt{F}'$. Let $\mathcal{V}_{F'}(s_1 y) = \{v_t'\}$. (It is a singleton set as $\mathtt{F}'$ is deterministic.) Hence v_1 corresponds to v_t' on common string $s_1 y$. Similarly, each vertex $v \in V$ corresponds to v_t' on some string ending with y. Let the clique corresponding to v_t' be K_t, and we have $K_t \supseteq V$. But that is a contradiction. □

6.1.6 Correspondence of MCCZC and VSO-FDM solutions

To establish the equivalence between MCCZC and VSO-FDM, we will show that the induced filter from the solution of MCCZC is a minimal filter for VSO-FDM, and the induced clique cover from a minimal filter is a solution for MCCZC.

Lemma 6.6. Minimal clique covers for MCCZC induce minimal filters for VSO-FDM.

Proof. Given any minimal clique cover $\mathbf{K} = \{K_1, K_2, \ldots, K_m\}$ as a solution for problem MCCZC with input p-filter $\mathtt{F}$, construct p-filter $\mathtt{F}' = M(\mathtt{F}, \mathbf{K})$. Since $\mathbf{K}$ satisfies the zipper constraints $\mathscr{Z}(\mathtt{F})$, $\mathtt{F}'$ is deterministic and output simulates $\mathtt{F}$ according to Lemma 6.4. To show that $\mathtt{F}'$ is a minimal deterministic filter for VSO-FDM, suppose the contrary. Then there exists a minimal deterministic filter $\mathtt{F}^{\star}$ with fewer states, i.e., $|V(\mathtt{F}^{\star})| < |V(\mathtt{F}')|$. Hence, $\mathtt{F}^{\star}$ induces a clique cover $\mathbf{K}^{\star}$ with fewer cliques than $\mathbf{K}$. Since $\mathtt{F}^{\star}$ is deterministic, $\mathbf{K}^{\star}$ satisfies all $\mathscr{Z}(\mathtt{F})$ via Lemma 6.5. But then $\mathbf{K}^{\star}$ satisfies all the requirements to be a solution for MCCZC, and has fewer cliques than $\mathbf{K}$, contradicting the assumption. □

Lemma 6.7. A minimal filter for VSO-FDM with input $\mathtt{F}$ induces a clique cover that solves MCCZC with compatibility graph and zipper constraints of $\mathtt{F}$.

Proof. Given minimal filter $\mathtt{F}^{\star}$ as a solution for VSO-FDM with input filter $\mathtt{F}$, we can construct a clique cover $\mathbf{K} = \mathbf{Q}(\mathtt{F}, \mathtt{F}^{\star})$ from the minimal filter. For this cover to be a solution for MCCZC with compatibility graph $\mathtt{G} = \mathcal{K}(\mathtt{F})$ and zipper constraints $\mathscr{Z}(\mathtt{F})$, first, it must satisfy all constraints in $\mathscr{Z}(\mathtt{F})$. Lemma 6.5 affirms this fact. Second, we must show it to be minimal among all the covers satisfying those constraints. Supposing $\mathbf{K}$ is not a minimal, there must exist a clique cover $\mathbf{K}'$ with $|\mathbf{K}'| < |\mathbf{K}|$ satisfying $\mathscr{Z}(\mathtt{F})$. Then, consider the induced filter $\mathtt{F}' = M(\mathtt{F}, \mathbf{K}')$. Since $\mathbf{K}'$ satisfies all the zipper constraints $\mathscr{Z}(\mathtt{F})$, $\mathtt{F}'$ is deterministic and will output simulate $\mathtt{F}$ (Lemma 6.4). But $|V(\mathtt{F}')| = |\mathbf{K}'| < |\mathbf{K}| = |V(\mathtt{F}^{\star})|$, contravening the fact that $\mathtt{F}^{\star}$ is a minimal filter. Hence $\mathbf{K}$ is minimal. □

Together, they establish the theorem.

Theorem 6.2. The solution for MCCZC with compatibility graph and zipper constraints of a filter $\mathtt{F}$ induces a solution for VSO-FDM with input filter $\mathtt{F}$, and vice versa.

Proof. Lemma 6.6 and Lemma 6.7 comprise the complete result. □

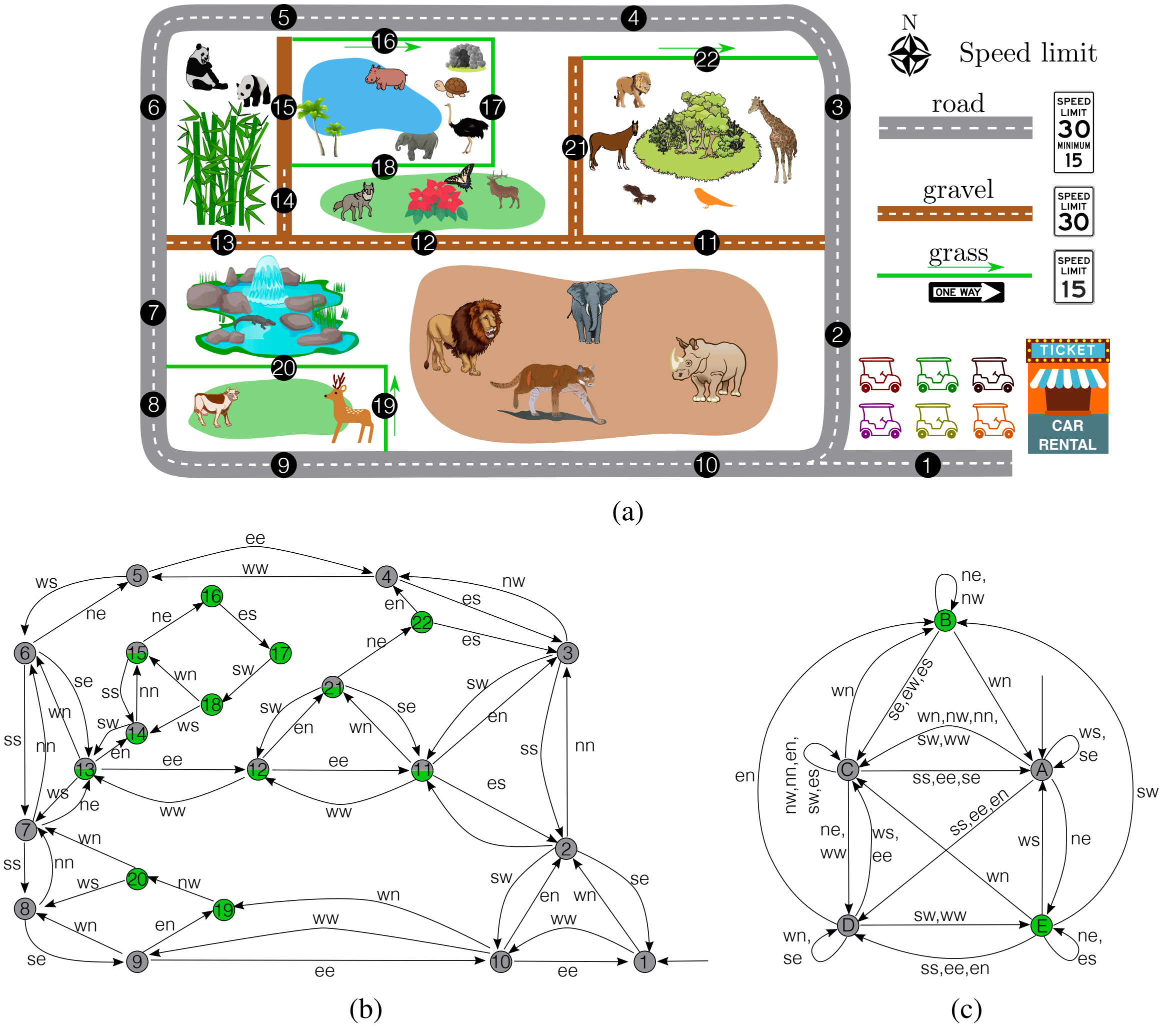
Having established this correspondence, merging and split operations, modeled by a cover, are sufficient to find a minimizer for the VSO-FDM problems.

6.2 Cover combinatorial filter minimization

In this section, we consider the problem of minimizing a slightly general filter, which does not need to be vertex single-outputting. We call this problem FDM.

6.2.1 Motivating example

To motivate the generalized filters and their minimization problems, we relax the speed limit of the gravel roads in the previous motivating example as shown in Figure 6.6a, such that the vehicle can drive at a high gear or low gear at these roads. The naïve filter shown in Figure 6.6b, which creates a state for each road, is no longer vertex single-outputting, and we call them *cover filters*. In this new naïve filter, the states representing the gravel roads are colored both green and gray. The states with multiple valid outputs introduce a new degree of freedom which influences the size of the minimal filter. These arise, for instance, whenever there are ‘dont-care’ options. The flexibility of such states must be retained to truly minimize the number of states, as they give a smaller minimizer as shown in Figure 6.6c.



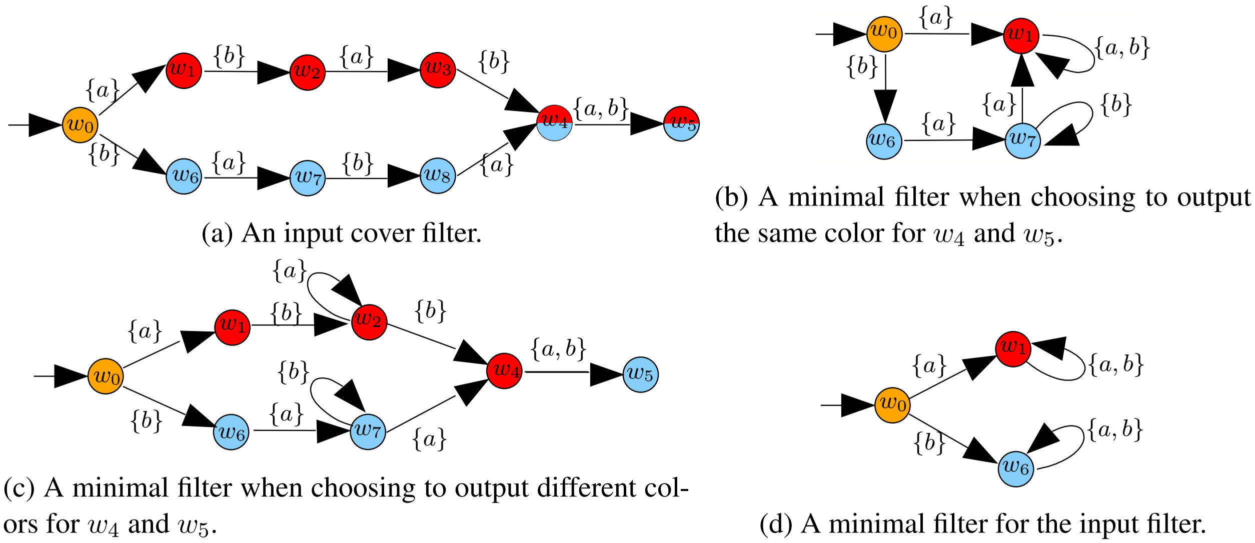
(a)

(b)

(c)

Figure 6.6: (a) A safari park with vehicles for hire. The speed limit of the gravel road is relaxed and both high gears are applicable. (b) A naïve filter. (c) A minimal filter for the intelligent gear shifting system.

To minimize cover filters, one straightforward approach is to enumerate all filters under different output choices for the states with multiple outputs, and then solve every one the resulting deterministic single-outputting filters as instances of VSO-FDM. The filter with the fewest states among all the minimizers could then be treated as a minimal one for the FDM problem. Unfortunately, this is too simplistic. Prematurely committing to an output choice is detrimental. Consider the input filter shown in Figure 6.7a, it has two multi-outputting states (w_4 and w_5). If we choose to have both w_4 and w_5 give the same output, the VSO-FDM minimal filter, shown in Figure 6.7b, has 4 states. If we choose distinct outputs for w_4 and w_5, the VSO-FDM minimal filter, shown in Figure 6.7c, now has 7 states. But neither is the minimal filter for FDM. The true minimizer appears in Figure 6.7d, with only 3 states. It is obtained by splitting both w_4 and w_5 into two copies, each copy giving a different output. Therefore, it may fail to find a minimizer by picking an output for each vertex first and then conducting mergers and splits, since the construction of the minimizer may require to choose different outputs for the splits. The multi-outputting states give extra degrees of freedom to minimize the cover filters.



(a) An input cover filter.

(b) A minimal filter when choosing to output the same color for w_4 and w_5.

(c) A minimal filter when choosing to output different colors for w_4 and w_5.

(d) A minimal filter for the input filter.

Figure 6.7: A multi-outputting filter minimization problem.

To exploit this extra degree of freedom, is it NP-hard? In this section, we are going to examine the hardness results and minimize the cover filters by generalizing the notion of compatibility relationships, zipper constraints, and minimum clique cover problems.

6.2.2 Cover combinatorial filter minimization problems and their hardness results

Problem: Filter Minimization (FDM)

Input: A deterministic filter F.

Output: A deterministic filter F† with fewest states, such that F† output simulates F.

We use 'F' for deterministic filters, 'D' for deterministic input and output, 'M' for minimization.

Theorem 6.3. FDM is NP-Complete.

Proof. Firstly, VSO-FDM is a special case of FDM problems. These FDM problems are at least as hard as VSO-FDM. Hence, FDM are in NP-hard. On the other hand, a solution for FDM can be verified in polynomial time. (Change the equality check on line 7 of Algorithm 1 in [2] to a subset check.) Therefore, FDM is NP-Complete. □

6.2.3 Modeling FDM as a generalized minimum cover problem

The idea underlying a correct approach is that output choices should be made together with the splitting and merging operations during filter minimization. Multi-outputting vertices may introduce additional split operations, but these split operations can still be treated via clique covers on the compatibility graph. This requires that we define a new compatibility relationship—it is only slightly more general than Definition 6.6:

Definition 6.13 (group compatibility)**.** Let F be a deterministic p-filter. We say that the set of states $U = \{u_1, u_2, \ldots, u_n\}$ are *group compatible*, if there is a common output on all their extensions, i.e.,

$$\forall s \in \bigcup_{u\in U} \mathcal{L}^{e}_{\mathtt{F}}(u),\ \bigcap_{w'\in W'} c(w') \neq \varnothing,\ \text{where } W' = \mathcal{V}_{\mathtt{F}}(u_1, s) \cup \mathcal{V}_{\mathtt{F}}(u_2, s) \cup \cdots \mathcal{V}_{\mathtt{F}}(u_n, s).$$

(The preceding exploits the subtle fact that $\mathcal{V}_{\mathtt{F}}(v, s) = \varnothing$ when tracing s from v crashes in F.) With this definition, the compatibility graph must be generalized suitably:

Definition 6.14 (compatibility simplicial complex)**.** Given a deterministic multi-output filter F, its *compatibility simplicial complex* is a collection of simplices, where each simplex is a set of group compatible vertices in F.

The zipper constraints are generalized too, replacing mutual compatibility with group compatible states:

Definition 6.15 (generalized zipper constraint)**.** In the compatibility simplicial complex of filter F, if there exists a set of group compatible states $U = \{u_1, u_2, \ldots, u_n\}$, then they can only be selected to be merged if they always transition to a set of states that are also selected to be merged. For any sets of group compatible states $U, W \subseteq V(\mathtt{G})$ and some observation y, we create a *generalized zipper constraint* expressed as a pair $(U, W)_y$ if $W = \{w \in V(\mathtt{G}) \mid y \in \tau(u, w) \text{ for some } u \in U\}$.

The information formerly encoded in cliques of edges is now within simplicies; the minimum clique cover on the compatibility graph, thus, becomes a minimum simplex cover on the compatibility simplicial complex. Hence, the MCCZC problem is generalized as follows:

Problem: Generalized Minimum Cover with zipper constraints (GMCZC)

Input: A compatibility simplicial complex M, a set of generalized zipper constraints ZIP.

Output: A minimal cardinality simplex cover of M satisfying ZIP.

6.3 An algorithm toward filter minimization

In the previous section, we have shown that the operations that are sufficient for a filter minimization problem consist of both mergers and splits, and these two operations on both vertex single-outputting and cover filters can be modeled by a clique cover, which leads to a minimum clique cover problem. But formalizing a clique cover problem is inefficient since it involves zipper constraints that are exponential in size. In this section, instead of formalizing the clique cover problem with zipper constraints, we encode the solutions of the filter minimization problem as a vertex cover, then induce a filter from the cover, and introduce output simulating constraints on the induced filter. This gives us a formalism with only polynomial number of constraints and can be solved by a variety of solvers.

To begin, define the basic combinatorial object involved:

Definition 6.16 (vertex cover)**.** A *vertex cover* $\boldsymbol{K} = \{K_1, K_2, \ldots, K_m\}$ on a filter $\texttt{F}$ is a collection of subsets of vertices which cover all $\texttt{F}$'s vertices, i.e., $K_i \subseteq V(\texttt{F})$ for each i, and $\bigcup_{i=1}^{m} K_i = V(\texttt{F})$. The size of $\boldsymbol{K}$ is number of the subsets, i.e., $|\boldsymbol{K}| = m$.

A vertex cover $\boldsymbol{K} = \{K_1, K_2, \ldots, K_m\}$ on filter $\texttt{F}$ is *zipped* if for every subset $K_i \in \boldsymbol{K}$ and for each observation $y \in Y(F)$, there exists at least one subset $K_j \in \boldsymbol{K}$ that contains all y-children of the states within K_i. Next, we show how a zipped vertex cover begets a filter.

Definition 6.17 (induced filter from a vertex cover)**.** Given zipped vertex cover $\boldsymbol{K} = \{K_1, \ldots, K_m\}$ on $\texttt{F} = (V, \{v_0\}, Y, \tau, C, c)$, its induced filter $\texttt{F}^\dagger = (V^\dagger, \{v_0^\dagger\}, Y, \tau^\dagger, C, c^\dagger)$ is constructed as follows:

1. Create a state $v_i^\dagger$ in $\texttt{F}^\dagger$ for each non-empty subset K_i.

2. Select an arbitrary vertex $v_i^\dagger$ as $v_0^\dagger$, such that the corresponding K_i contains the initial state v_0 in $\texttt{F}$.

3. For each vertex $v_i^\dagger$ and $y \in Y$, if y-children of vertices in K_i is not empty, then add one transition from $v_i^\dagger$ to $v_j^\dagger$ under y such that K_j contains all y-children of K_i; if there are multiple such $v_j^\dagger$s, pick an arbitrary one.

4. Assign the output for $v_i^\dagger$ to be $c^\dagger(v_i^\dagger) \in \bigcap_{v \in K_i} c(v)$, i.e., an output common to all vertices in K_i.

Note that each vertex in filter $\texttt{F}$ may be contained in multiple subsets in the vertex cover, and that each subset in the cover is mapped to a *unique* vertex in the induced filter. Hence, we say that each vertex in $\texttt{F}$ may be *mapped to* multiple vertices in the induced filter.

Being zipped ensures that, for any y with children, there is always an outgoing transition in 3) of Definition 6.17. Thus, the construction never shrinks the filter's language.

Lemma 6.8. Let $\texttt{F}^\dagger$ be the induced filter for a zipped vertex cover $\boldsymbol{K}$ on a filter $\texttt{F}$. It holds that $\mathcal{L}^I(\texttt{F}) \subseteq \mathcal{L}^I(\texttt{F}^\dagger)$.

Proof sketch. This lemma can be proved by induction on the length of strings $s \in \mathcal{L}^I(\mathtt{F})$ that shows if s reaches a state v in F, then s reaches a state $v_i^\dagger$ in $\mathtt{F}^\dagger$ such that the corresponding K_i contains v. This will show that if $s \in \mathcal{L}^I(\mathtt{F})$, then $s \in \mathcal{L}^I(\mathtt{F}^\dagger)$, meaning that $\mathcal{L}^I(\mathtt{F}) \subseteq \mathcal{L}^I(\mathtt{F}^\dagger)$. □

The next throws light on why vertex covers interest us.

Lemma 6.9. Given an input filter $\mathtt{F} = (V, \{v_0\}, Y, \tau, C, c)$, if there exists a solution to k-FM, then there is always a filter $\mathtt{F}^\dagger$ as a solution to k-FM such that $\mathtt{F}^\dagger$ is induced from a zipped vertex cover on F.

Proof. Let $\mathtt{F}^\star = (V^\star, \{v_0^\star\}, Y, \tau^\star, C, c^\star)$ be any solution to k-FM with input F. From $\mathtt{F}^\star$ and F, we construct a zipped vertex cover $\boldsymbol{K}$ on F, then construct an induced filter $\mathtt{F}^\dagger$ from $\boldsymbol{K}$ and show that $\mathtt{F}^\dagger$ is also a solution for k-FM.

We construct $\boldsymbol{K}$ as follows: For every state $v_i^\star \in V^\star$, construct set $K_i = \{v \in V \mid \mathcal{S}_v^{\mathtt{F}} \cap \mathcal{S}_{v_i^\star}^{\mathtt{F}^\star} \neq \varnothing\}$, and form $\boldsymbol{K} = \{K_1, K_2, \dots, K_{|V^\star|}\}$. Collection $\boldsymbol{K}$ is a vertex cover on F because by assumption $\mathcal{L}^I(\mathtt{F}) \subseteq \mathcal{L}^I(\mathtt{F}^\star)$, which implies that for each $v \in V$ there is at least one vertex $v^\star \in V^\star$ with $\mathcal{S}_v^{\mathtt{F}} \cap \mathcal{S}_{v^\star}^{\mathtt{F}^\star} \neq \varnothing$, and this means that each vertex v of F is contained in at least one subset $K \in \boldsymbol{K}$. In addition, $\boldsymbol{K}$ is zipped, otherwise some string is in F but not in $\mathtt{F}^\star$, contradicting the fact that $\mathtt{F}^\star$ output simulates F.

Now, we show that $\mathtt{F}^\dagger$, constructed from $\boldsymbol{K}$ following Definition 6.17, is also a solution to k-FM. Trivially, $|\boldsymbol{K}| = |V^\star| \leq k$, and $|V(\mathtt{F}^\dagger)| \leq |\boldsymbol{K}|$. Hence, $|V(\mathtt{F}^\dagger)| \leq k$.

We will prove by contradiction that $\mathtt{F}^\dagger$ output simulates F. Suppose $\mathtt{F}^\dagger$ does not output simulate F. Then there must be a string $s \in \mathcal{L}^I(\mathtt{F})$, such that either (i) $s \notin \mathtt{F}^\dagger$, or (ii) at least two states are reached by s in $\mathtt{F}^\dagger$ ($\mathtt{F}^\dagger$ is non-deterministic), or (iii) $\mathcal{C}(\mathtt{F}^\dagger, s) \not\subseteq \mathcal{C}(\mathtt{F}, s)$. Regarding case (i), since $s \in \mathcal{L}^I(\mathtt{F})$ and $\mathcal{L}^I(\mathtt{F}) \subseteq \mathcal{L}^I(\mathtt{F}^\star)$, we have $s \in \mathcal{L}^I(\mathtt{F}^\star)$. Let $v_j^\star$ be a vertex reached by s in $\mathtt{F}^\star$. Then s must also reach $v_j^\dagger$ in $\mathtt{F}^\dagger$, which indicates that $s \in \mathtt{F}^\dagger$. Regarding case (ii), let $v_j^\dagger$ and $v_l^\dagger$ ($j \neq l$) be two states in $\mathtt{F}^\dagger$ that are reached by s. Notice that there is an injective function from the vertices and edges in $\mathtt{F}^\dagger$ to those in $\mathtt{F}^\star$. Thus, s must reach two different vertices $v_j^\star$ and $v_l^\star$ in $\mathtt{F}^\star$, which contradicts the fact that $\mathtt{F}^\star$ is deterministic. Regarding case (iii), there must exist a vertex

$v_i^\dagger$ in $\mathtt{F}^\dagger$ and a vertex v in $\mathtt{F}$ that are both reached by s and that $v_i^\dagger$ and v have different outputs. But according to the construction of $\mathtt{F}^\dagger$, $v_i^\dagger$ must share the same output as v. Hence, $\mathtt{F}^\dagger$ must output simulate $\mathtt{F}$. □

Hence, to solve FM, we can always look for vertex covers.

6.3.1 Searching over vertex covers via variables

Now we represent a vertex cover with binary variables.

To encode a vertex cover $\boldsymbol{K} = \{K_1, K_2, \dots K_m\}$ on an input filter $\mathtt{F} = (V, \{v_0\}, Y, \tau, C, c)$, we introduce the following binary variables:

- Create a binary variable R_v^i for each $v \in V$ and each $i \in \{1, 2, \dots, |V|\}$, and assign $R_v^i = 1$ if and only if v is contained in K_i. If $i > |\boldsymbol{K}|$, then we view K_i as an empty set and set $R_v^i = 0$ for all $v \in V$.
- Create a binary variable q^i for each $i \in \{1, 2, \dots, |V|\}$, and assign $q^i = 1$ if and only if K_i is not empty.

We also define additional variables with constant values assigned from the structure of the input filter:

- Introduce a binary variable t_v^y for each $v \in V$ and $y \in Y$, to which we assign value 1 if and only if v has non-empty y-children.
- Introduce a binary variable p_v^o for each $v \in V$ and $o \in O$, to which we assign value 1 if and only if v has o in its outputs, i.e., $o \in c(v)$.

With these variables, we can encode an output filter and the constraints for it to be a valid solution in FM.

6.3.2 FM as an integer nonlinear program (INP)

Now, we formalize FM as an integer nonlinear program. In what follows, we denote the input filter as $\mathtt{F} = (V, \{v_0\}, Y, \tau, C, c)$, the vertex cover to be searched for as $\boldsymbol{K}$, and the induced output filter from $\boldsymbol{K}$ as $\mathtt{F}^\dagger = (V^\dagger, \{v_0^\dagger\}, Y, \tau^\dagger, C, c^\dagger)$. For brevity, we will simply write $\forall i$ for $\forall i \in \{1, 2, \ldots, |V|\}$, $\forall v$ for $\forall v \in V$, and $\forall y$ for $\forall y \in Y$.

Minimize

$$\sum_{1 \le j \le |V|} q^j \tag{INP-Obj}$$

Subject to:

$$q^i, R_v^i \in \{0, 1\} : \forall i, \forall v \tag{INP-Vars}$$

$$R_v^i \le q^i : \forall i, \forall v \tag{INP-NESubset}$$

$$q^i \le q^{i-1} : \forall i \tag{INP-Sym}$$

$$\sum_{1 \le j \le |V|} R_{v_0}^j \ge 1 \tag{INP-ValidCover}$$

$$\sum_{1 \le j \le |V|} \prod_{v \in V} (2 - R_v^i - t_v^y + R_{v_y}^j) \ge 1 : \forall i, \forall y \tag{INP-Zip}$$

$$\sum_{o \in C} \prod_{v \in V} (1 - R_v^i + p_v^o) \ge 1 : \forall i \tag{INP-Out}$$

The objective (INP-Obj) is to minimize the number of non-empty subsets in $\boldsymbol{K}$. For each j, variable q^j receives value 1 if at least one vertex of $\mathtt{F}$ is assigned to K_j. This is expressed by constraints (INP-NESubset). We use the idea of Méndez-Díaz and Paula [71] to reduce symmetry by pushing the non-empty subsets to smaller indices. This is imposed by constraints (INP-Sym).

Constraint (INP-ValidCover) requires that the initial state of $\mathtt{F}$ be contained in at least one subset of the vertex cover. Together with constraints (INP-Zip), this ensures that all vertices of $\mathtt{F}$ that are reachable from the initial state will be covered by $\boldsymbol{K}$. For the output filter to be deterministic, for each observation y and each state $v_i^\dagger$ in $\mathtt{F}^\dagger$, $v_i^\dagger$ must have at most a single y-child. Accordingly, for each subset K_i, there must exist a subset K_j that contains all the y-children of K_i. More exactly, $\forall i \in \{1, 2, \ldots, |V|\}, \forall y \in Y$:

$$\exists j \in \{1, 2, \ldots, |V|\}, \text{s.t.,} \underbrace{\forall v \in V, \Big((R_v^i t_v^y = 1) \implies (R_{v_y}^j = 1) \Big)}_{\text{all } y\text{-children of } K_i \text{ are contained in } K_j}.$$

In algebraically simplified form, this is the zipped constraints (INP-Zip). Note that we allow multiple such K_j's to exist for any K_i and y, but in the output filter, only one will be (arbitrarily) picked for the transition.

In addition, the output for vertex $v_i^\dagger$ should be the common output of all vertices in the subset K_i. This means that for each subset K_i, all states within that subset must share a common output. Formally, $\forall i \in \{1, 2, \ldots, |V|\}$,

$$\exists o \in C, s.t., \underbrace{\forall v \in V, \big((R_v^i = 1) \implies (p_v^o = 1)\big)}_{\text{all vertices in } K_i \text{ must share output } o},$$

which is expressed by constraints (INP-Out).

With a solution to the integer nonlinear program in hand, we first form the vertex cover $\boldsymbol{K}$ by constructing the subsets according to the values assigned to variables R_v^i's. Then we make the output filter $\mathtt{F}^\dagger$ by following Definition 6.17.

The next we prove the correctness of INP.

Lemma 6.10 (correctness)**.** Let $\boldsymbol{K}$ be the vertex cover formed by an optimal solution to the integer nonlinear program for an input filter $\mathtt{F}$ and let $\mathtt{F}^\dagger$ be an induced filter from $\boldsymbol{K}$. Filter $\mathtt{F}^\dagger$ is an optimal solution to FM with input $\mathtt{F}$.

Proof sketch. The nonlinear programming constraints formalize the requirement for the induced $\mathtt{F}^\dagger$ to be deterministic and output simulate filter $\mathtt{F}$. The objective function ensures the minimum number of states. Both do this *exactly*, which we show by establishing the equivalence between the nonlinear constraints and properties of determinism and output simulating in FM. This holds in both directions.

$\impliedby$: If constraints (INP-ValidCover) and (INP-Zip) hold, then $\boldsymbol{K}$ is a zipped vertex cover, $\mathtt{F}^\dagger$ constructed following Definition 6.17 is deterministic, and $\mathcal{L}^I(\mathtt{F}) \subseteq \mathcal{L}^I(\mathtt{F}^\dagger)$ as per Lemma 6.8. If constraint (INP-Out) is satisfied, then $\forall s \in \mathcal{L}^I(\mathtt{F})$, $\mathcal{C}(\mathtt{F}, s) \supseteq \mathcal{C}(\mathtt{F}^\dagger, s)$. Hence, $\mathtt{F}^\dagger$ is deterministic and output simulates $\mathtt{F}$.

$\implies$: Given an $\mathtt{F}^\dagger$ that is an optimal solution for FM, construct an induced zipped cover $\boldsymbol{K}$ following Lemma 6.9. The values of the variables encoding this cover must satisfy constraints (INP-NESubset) and (INP-ValidCover). If $\mathtt{F}^\dagger$ is deterministic, then constraints (INP-Zip) must also be satisfied. The fact that $\texttt{F}^\dagger$ output simulates $\texttt{F}$ implies that constraints (INP-Out) are satisfied.

Proof that if $\texttt{F}^\dagger$ is minimal, then the value of (INP-Obj) must be optimal (and vice versa) follows similarly. □

6.3.3 Integer linear programming (ILP) with linear constraints

This section presents an integer linear program by linearizing the nonlinear constraints (INP-Zip) and (INP-Out).

To linearize constraints (INP-Zip), we introduce a binary variable $a_y^{i,j}$ for each $i, j \in \{1, 2, \ldots, |V|\}$ and $v \in V$ to determine whether there is a transition from vertex $v_i^\dagger$ to vertex $v_j^\dagger$ under label y in the output filter $\texttt{F}^\dagger$. If $a_y^{i,j} = 1$, then the value of term $\prod_{v\in V}(1 - R_v^i + 1 - t_v^y + R_{v_y}^j)$ must be a positive integer. Otherwise, we choose not to build such a transition in the output filter, regardless of the value of the corresponding term. Mathematically, we have:

$$a_y^{i,j} + R_v^i + t_v^y - R_{v_y}^j \leq 2 : \forall i, \forall j, \forall v, \forall y. \qquad \text{(ILP-Zip-1)}$$

Then constraints (INP-Zip) are written as

$$\sum_{1\leq j\leq |V|} a_y^{i,j} \geq 1 : \forall i, \forall y. \qquad \text{(ILP-Zip-2)}$$

For constraints (INP-Out), we similarly introduce a binary variable b_o^i, with value 1 to denote the fact that the term $\prod_{v\in V}(1 - R_v^i + p_v^o)$ has a positive value. If $b_o^i = 0$, then we do not care whether the value of the corresponding term is positive or not. Thus, we add the following constraints:

$$1 - b_o^i + 1 - R_v^i + p_v^o \geq 1 : \forall i, \forall o, \forall v. \qquad \text{(ILP-Out-1)}$$

Then constraints (INP-Out) are linearized as follows:

$$\sum_{o\in C} b_o^i \geq 1 : \forall i. \qquad \text{(ILP-Out-2)}$$

6.3.4 Boolean satisfaction (SAT)

We next treat FM as a sequence of k-FM problems by enumerating the bound on the output filter size. Each k-FM is formalized as a Boolean satisfaction problem, which we call $SAT_{[k]}$. To find the minimal filter, the idea is to solve a $SAT_{[k]}$, and then decrement k until no smaller output filter can be found.

To obtain a $SAT_{[k]}$ instance, we first remove variables q^i $(\forall i)$ and constraints (INP-NESubset) and (INP-Sym) since we do not need (INP-Obj) and only want to find an output filter with size bounded by k. Next, we treat binary variables R^i_v, $a^{i,j}_y$, b^i_o as boolean-valued and write constraints (ILP-Zip-1)–(ILP-Out-2) in conjunctive normal form (CNF).

Given a filter minimization problem with size bounded by k, constraint (INP-ValidCover) is written as a clause:

$$\bigvee_{i \in \{1,2,\ldots,k\}} R^i_{v_0}. \tag{SAT-ValidCover}$$

Constraints (ILP-Zip-1) and (ILP-Zip-2) are written as:

$$\overline{a^{i,j}_y} \vee \overline{R^i_v} \vee \overline{t^y_v} \vee R^j_{v_y} : \forall i, \forall j, \forall v, \forall y \tag{SAT-Zip-1}$$

$$\bigvee_{j \in \{1,2,\ldots,k\}} a^{i,j}_y : \forall i, \forall y. \tag{SAT-Zip-2}$$

And constraints (ILP-Out-1) and (ILP-Out-2) become:

$$\overline{b^i_o} \vee \overline{R^i_v} \vee p^o_v : \forall i, \forall o, \forall v, \tag{SAT-Out-1}$$

$$\bigvee_{o \in C} b^i_o : \forall i. \tag{SAT-Out-2}$$

Notice that consecutive SAT instances share most of their variables and constraints. The $SAT_{[k]}$ instance is equivalent to the $SAT_{[k+1]}$ one but with additional unit clauses $\overline{R^{k+1}_v}$ for all $v \in V$. Instead of making each $SAT_{[k]}$ instance from scratch and solving it, we add unit clauses while decreasing k. This allows the solver to re-use knowledge acquired from previous SAT instances, and leads to an incremental anytime procedure in Algorithm 8. First, we initialize k to be $|V(\mathtt{F})|$ (line 1). Next, we construct a CNF formula $SAT_{[k]}$ (line 2) and invoke the SAT solver (line 3–6). If an assignment is found for $SAT_{[k]}$ within the time budget (line 7), then we set its cover choice to be the smallest one found so far (line 8), update the time budget by the amount of time used in this iteration (line 9), add the unit clauses (line 10), and decrease k (line 11). Otherwise, if the $SAT_{[k]}$ has not been solved within the time budget, then we use the minimum cover found so far to construct the minimal filter (line 14). When given an adequate time budget, the algorithm will find the minimal filter. And running the algorithm for a longer duration increases the chance of finding a smaller filter.

Algorithm 8: $\mathrm{SAT}(\mathtt{F}, \mathit{timeout})$

```
$k \leftarrow |V(\mathtt{F})|$
$\mathrm{CNF} \leftarrow \mathrm{BuildFormula}(\mathtt{F}, k)$
Initialize minimum vertex cover $\boldsymbol{K}_{min}$ to be empty
$solver \leftarrow \mathrm{SATSolver}(\mathrm{CNF})$
while $k \geq 1$ and $timeout > 0$ do
    $result \leftarrow solver.\mathrm{solve}(timeout)$
    if $result.solved$ then
        $\boldsymbol{K}_{min} \leftarrow result.model$
        Reduce $timeout$ by the time used
        Add unit clauses $\overline{R_v^k}$ $(\forall v \in V)$ to $solver$
        $k \leftarrow k - 1$
    else
        break
$\mathtt{F}' \leftarrow \mathrm{FilterConstruction}(\mathtt{F}, \boldsymbol{K}_{min})$
return $\mathtt{F}'$
```

6.3.5 SAT with just-in-time constraints: LazySAT

In SAT, zipped constraints (SAT-Zip-1) and (SAT-Zip-2) are critical to ensure deterministic transitions between subsets of $\boldsymbol{K}$. If a set of vertices in the input filter do not share any common output, then there is no need to check these zipped constraints on any set containing these vertices. In this case, we say that the zipped constraints related to these vertices are inactive. The existence of inactive constraints slows down the resolution of the SAT problem, but detecting and representing all active zipped constraints in FM requires exponential time and space. To speed up the SAT approach without significant overhead, we introduce LazySAT, a just-in-time treatment of the zipped constraints. In LazySAT, we first partition these constraints into non-overlapping sets of clauses, solve the SAT problem without these constraints, and introduce each set of clauses only when a non-zipped cover is returned and it violates these clauses.

Zipped constraints ensure that $\boldsymbol{K}$ covers F as well as being zipped. To treat them lazily, we update constraints (SAT-ValidCover) so that every state in F is contained in at least a subset:

$$\bigwedge_{v\in V}\Bigg(\bigvee_{i\in\{1,2,\ldots,k\}} R_v^i\Bigg). \qquad \text{(LazySAT-ValidCover)}$$

Next, we partition the clauses in the zipped constraints. Let the set of clauses from constraint (SAT-Zip-1) be $\mathbb{A}$. Then $\mathbb{A}$ can be partitioned into non-overlapping subsets according to the vertex v and outgoing label y, i.e., $\mathbb{A} = \cup_{v\in V}\cup_y \mathbb{A}_y^v$. Each subset $\mathbb{A}_y^v$ consists of the following clauses:

$$\bigwedge_{i\in\{1,2,\ldots,k\}}\bigwedge_{j\in\{1,2,\ldots,k\}}\big(\overline{a_y^{i,j}}\vee\overline{R_v^i}\vee\overline{t_v^y}\vee R_{v_y}^j\big).$$

Similarly, the set of clauses from constraint (SAT-Zip-2) is denoted as $\mathbb{B}$, which is parameterized by the outgoing label y. Each subset $\mathbb{B}_y$ consists of the following clauses:

$$\bigwedge_{i\in\{1,2,\ldots,k\}}\Bigg(\bigvee_{j\in\{1,2,\ldots,k\}} a_y^{i,j}\Bigg).$$

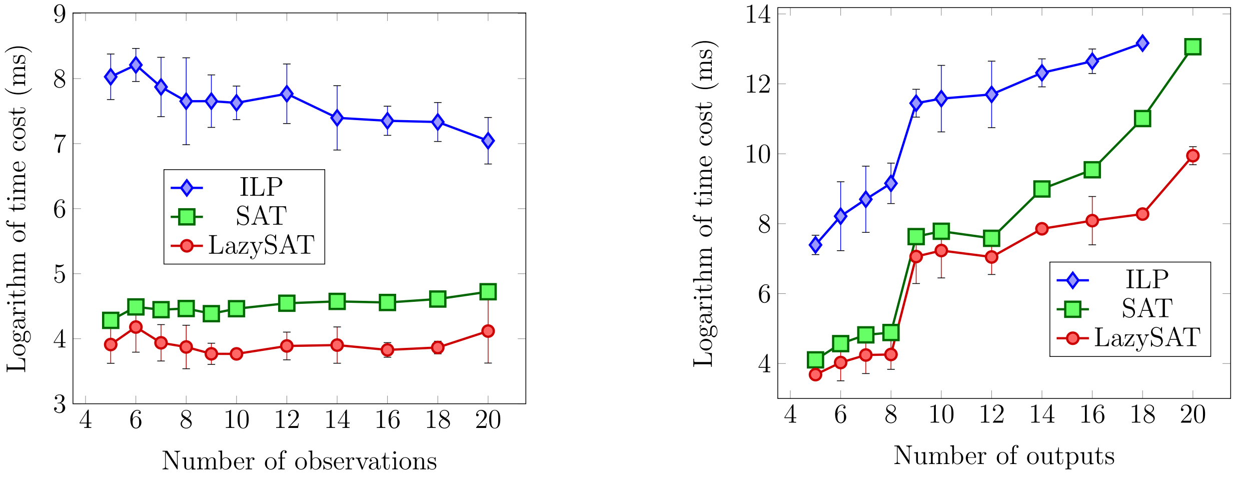
We detect the violation of these clauses, and add the clauses to the solver as needed. Let Y_c be the set of outgoing labels y such that the clauses in $\mathbb{B}_y$ are already present in the solver, and P be the set of vertex v and outgoing label y pairs such that $\mathbb{A}_y^v$ are also added to the solver. Both Y and P are initialized as empty sets. Given a vertex cover $\boldsymbol{K}$ returned from the SAT algorithm, if $\boldsymbol{K}$ is not zipped, then there must exist a set of states $K \in \boldsymbol{K}$ and a label y such that all y-children of vertices in K are not contained in any single subset in the cover $\boldsymbol{K}$. This must be a consequence of violating some missing clauses parameterized by K and y in the zipped constraints. We add these clauses as follows: (i) if $y \notin Y_c$, add y to Y_c and add clauses in $\mathbb{B}_y$ to the solver; (ii) for any $v \in K$, if $(v, y) \notin P$, add (v, y) to P and add clauses in $\mathbb{A}_y^v$ to the solver. Now, repeatedly call the solver, adding clauses if needed, until we find a zipped vertex cover with size no greater than k. To find a minimum solution, follow the same procedure as Algorithm 8.

6.3.6 Experimental results

We implemented INP, ILP, SAT and LazySAT in Python, based on mixed integer nonlinear solver SCIP [72], mixed integer linear solver Gurobi [73], and SAT solver CaDiCaL [74]. All executions are conducted on an OSX laptop with a $2.4\,\mathrm{GHz}$ Intel Core i5 processor, and each algorithm is given $10\,\mathrm{min}$ budget to solve a filter minimization problem.

First, we minimize the filter in Figure 6.6b. INP failed to give a result before timing out, while ILP, SAT and LazySAT give minimal filters with 5 states within $1\,\mathrm{s}$, $2\,\mathrm{s}$ and $100\,\mathrm{s}$, respectively. One such minimal filter is shown in Figure 6.6c. We collected no further results for INP since it appears to be incapable of minimizing filters with more than 10 states within $10\,\mathrm{min}$. To test the performance of the remaining three algorithms, we randomly generated a filter as follows: (i) construct a tree with a root node at layer 0 and w nodes at each of d additional layers, and then connect each vertex from a parent vertex in an earlier layer by drawing a directed edge; (ii) randomly pick m vertices to add self-loops; (iii) randomly pick n vertices to connect to some parent vertex in a later layer, so as to generate cycles. Next we randomly assign n_o outputs to vertices in the filter, where each vertex is assigned with p of them. Similarly, we randomly assign n_y observations to the edges in the filter while keeping the filter deterministic.

To compare ILP and SAT-based approaches, we start with a filter structure randomly generated by parameters d=4, w=3, m=n=2, p=2 and n_o=5. For any given number of observations n_y, we sample 10 filters, and collect the time to minimize these filters for each algorithm in Figure 6.8a. As more observations are added to the filter, fewer states share common observations. The zipped constraints (ILP-Zip-1) and (SAT-Zip-1) will be simplified, since $a_y^{i,j}$ connects with fewer vertices. Hence, the computational time for both ILP and SAT-based approaches tend to decrease. Fixing the number of observations to be $n_y = 5$, we also collect the computation time under varying outputs in Figure 6.8b. This gives an opposite trend as increasing the number of outputs makes the problem harder from two aspects: (i) the number of variables increases; (ii) the number of output constraints (ILP-Out-1) or (SAT-Out-1) increases owing to both an increasing number of outputs and an increasing number of vertices with $p_v^o = 0$ for each output o. Across both studies, SAT-based approaches outperform integer linear programming. We speculate that this is because the constraints for FM are fundamentally combinatorial in nature and can be concisely encoded in CNF. These CNF constraints can be exploited relatively efficiently (e.g., by building a constraint-



(a) Time cost (natural log) for inputs with varying observations.

(b) Time cost (natural log) for inputs with varying outputs.

Figure 6.8: Comparison of logarithmic computational time to minimize filters with different number of outputs and observations.

dependency graph). And a final factor might be that the objective function really takes a limited range of values and its values can be enumerated efficiently.

Observe that in Figure 6.8b, as we increase the number of outputs, LazySAT significantly outperforms SAT since few states share common outputs, so most zipped constraints are inactive and can be removed. We further tested them on a larger filter instance, where many states share common outputs and hence a significant proportion of constraints become active. In Figure 6.9, instead of presenting the time to find a minimal solution, we report the number of clauses used by the solver, and size of the sub-optimal solutions found by the two algorithms along the way. LazySAT is still able to find sub-optimal solutions faster than SAT, and the number of clauses used by LazySAT is much fewer than those in SAT. Treating constraints lazily does incur overhead in detecting and adding the active clauses, but the speedup from just-in-time treatment is seen to outweigh its overhead even when a large number of vertices share common outputs.

6.4 Nondeterministic filter minimization problems and their computational complexity

Now, we generalize toward the problem of minimizing nondeterministic filters.

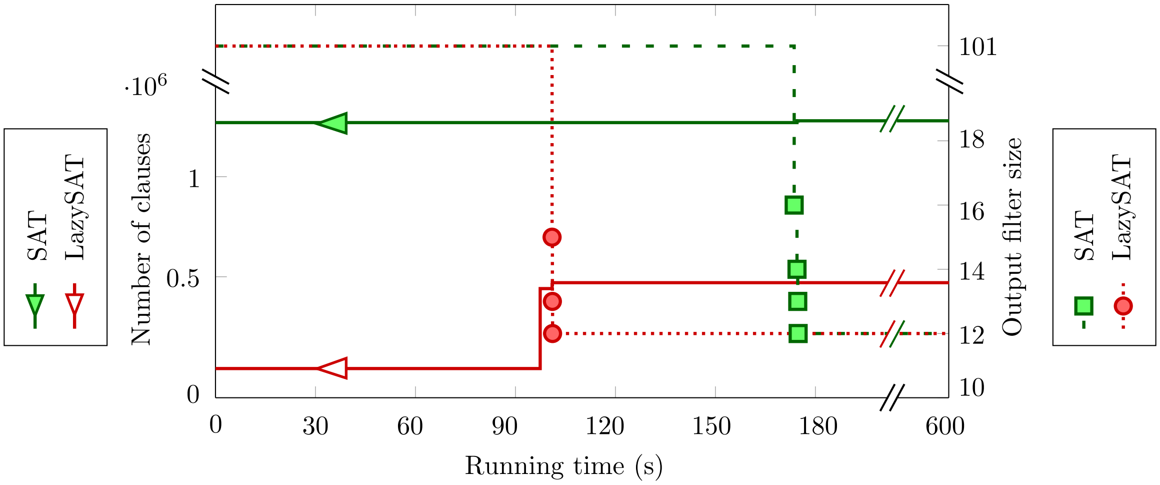


Figure 6.9: The number of constraints used by SAT and LazySAT to find sub-optimal solutions while increasing the running time. The input filter is constructed with parameters $d = 20$, $w = 5$, $m = n = 10$, $p = 1$, with 50 observations and 5 outputs.

6.4.1 Motiving example

Consider a donut environment illustrated in Figure 6.10a: two agents wander in a circular world, and three sensor beams (yielding symbols a, b, and c, respectively) partition the environment into sector-shaped regions (labeled 0, 1, 2). The beams detect if an agent crosses the dividing line but can sense neither the agent's identity nor direction of motion. With the agents starting in some known configuration (one in region 0 and one in region 1), the task is, given a sequence of sensor readings (i.e., a string of a's, b's, c's), to determine whether the two agents are within the same sector or not. A naïve nondeterministic filter is given in Figure 6.10b. The objective is to find a minimal deterministic filter, like the one shown in Figure 6.10d, for this estimation problem.

To arrive at the 4 state minimizer, we first determinize the input filter and produce a filter (Figure 6.10c), and solve it as a FDM problem. This procedure goes from 9 states, to 6, before reaching 4, finally. By way of contrast, consider the nondeterministic 5-state filter in Figure 6.11a. To find a minimal filter, it can be determinized to track the $2^4 = 16$ distinct information states shown in Figure 6.11b. Once that is minimized, it gives the deterministic filter in Figure 6.11c. The growth in the number of vertices, caused by the need to determinize for the minimization algorithm, indicates trouble. Not only does the set increase exponentially, but this much larger object becomes the input for an exponential cost algorithm (since the problem is NP-hard [2]).

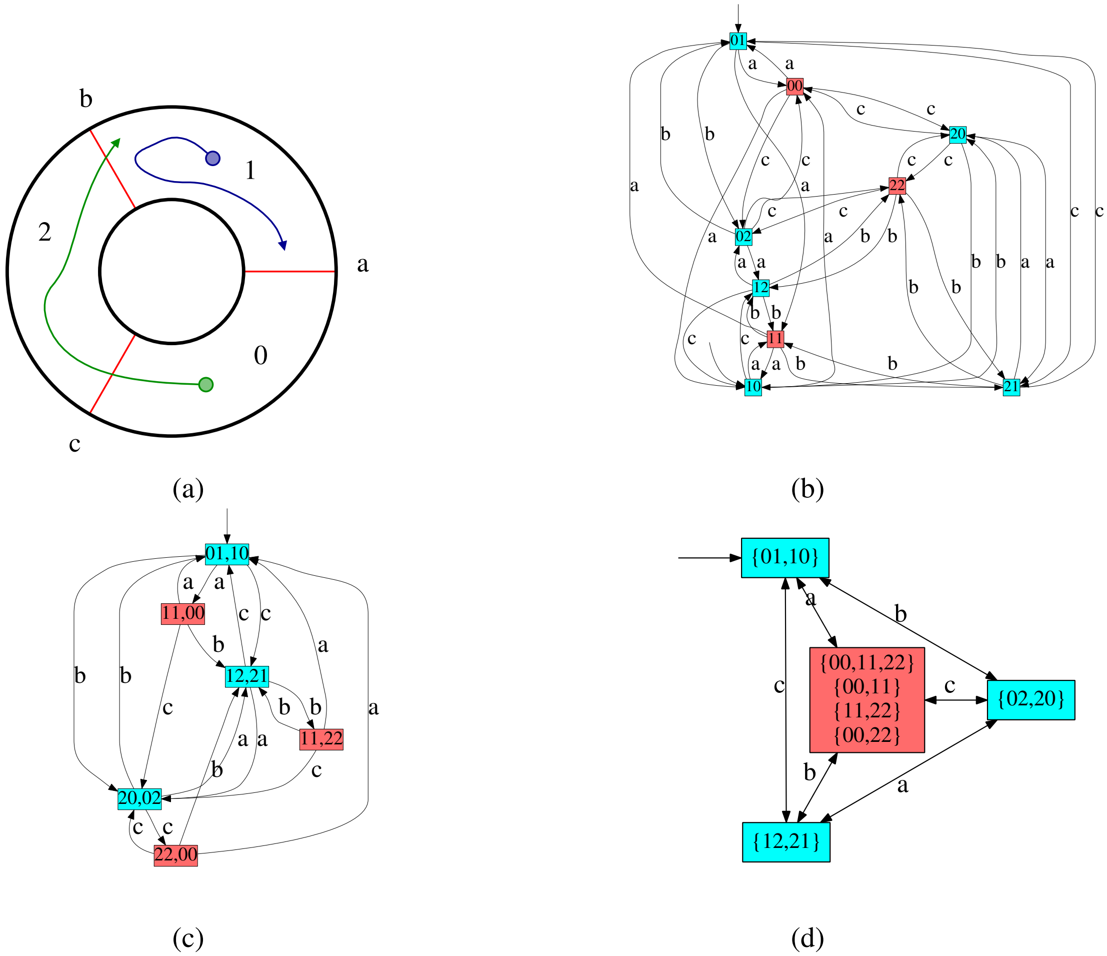


Figure 6.10: Combinatorial filter minimization, as originally motivated by [3]: (a) Two agents move in a circular world with three beam sensors. The environment is partitioned into three regions, indexed by numbers 0, 1 and 2. Letters a, b, and c denote observations from each of the three beams. (b) A nondeterministic filter to estimate whether these two agents are in the same region (red) or not (cyan). (c) The deterministic version of the same. (d) A minimal deterministic filter.
Double trouble.

To by-pass this expansion, one requires filter reduction methods that are able to consume nondeterministic filters directly as input. Looking again at Figures 6.11a and 6.11c, the dramatic compression that cancels the extreme expansion raises some questions. Do large deterministic instances arising from small nondeterministic ones really induce hard minimization problems? Or are they instead structured in some special (sparse, say, or otherwise low-density) form, reflecting conservation of underlying information? Computational complexity can provide clues: for example, in characterizing the space requirements of direct nondeterministic filter to deterministic minimizer computation.

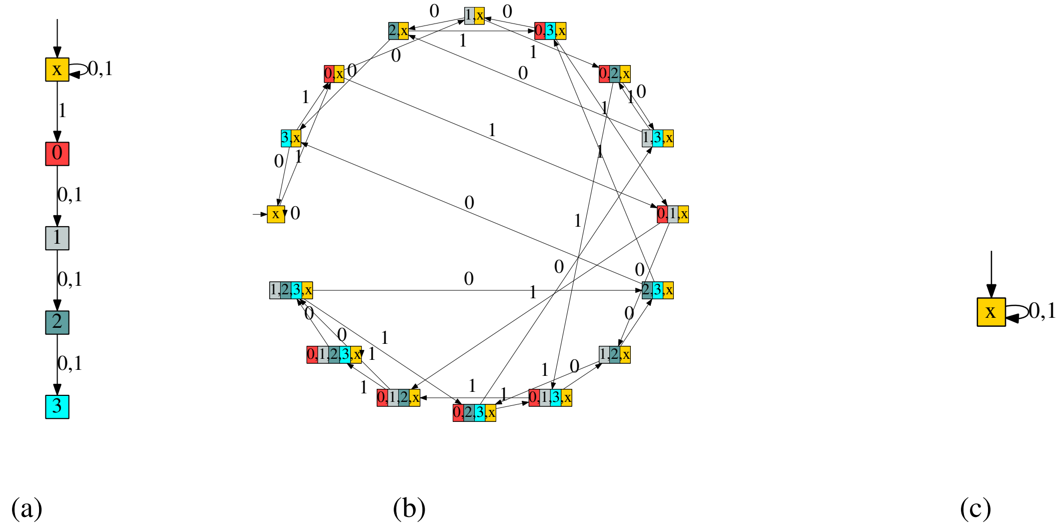


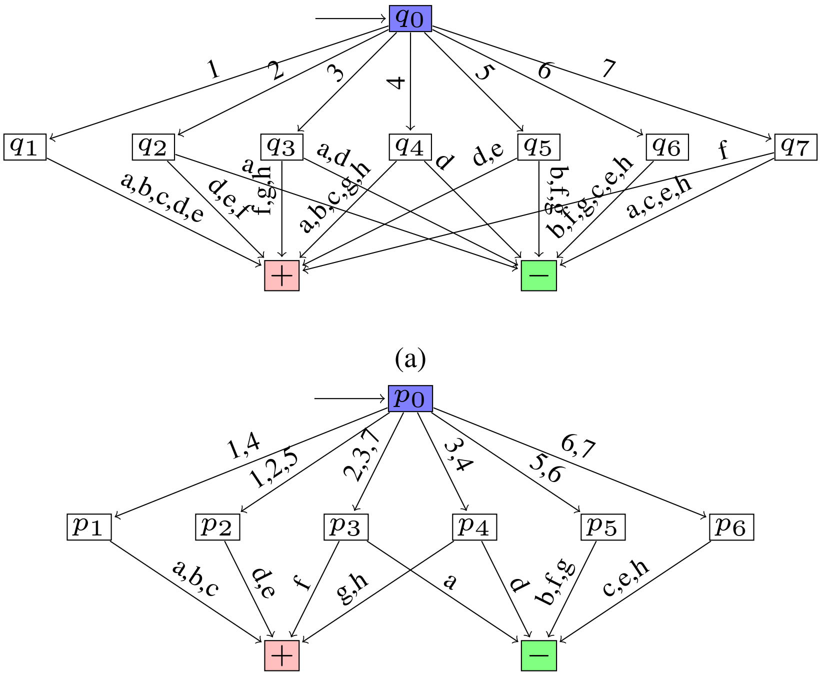
Figure 6.11: (a) A nondeterministic filter. (b) A deterministic form obtained via the power set construction. (c) A minimal filter.

If nondeterminism can be of added value as input to a minimization algorithm, what about as its output? In finite automata minimization, the smallest nondeterministic automata can be smaller than any deterministic one. Typical examples exploit the fact that accepting a string in the nondeterministic automaton requires that *some* tracing arrive at an accepting state. For filters, analogous instances fail owing to their differing semantics (stated formally in the next section). The analogous fact, however, does hold. A small example suffices to show this: the deterministic input filter given in Figure 6.12a cannot be reduced any further when only deterministic minimizers are considered (reason: no two states agree on their common extensions—see the definition of compatibility in the previous section. But it has a nondeterministic minimizer with fewer states, shown in Figure 6.10b. Nondeterminism, then, provides extra freedom that can be exploited to further reduce filter size.

To summarize: (i) nondeterminism in the input allows minimization to proceed directly on models of certain problems, potentially saving on expensive intermediate steps; (ii) permitting nondeterminism in the filters produced as output can deliver greater compression. Thus, nondeterminism may be of considerable practical importance.

6.4.2 Nondeterministic combinatorial filter minimization problems

Here is the formal definition of the nondeterministic filter minimization problem:



(a)

(b)

Figure 6.12: (a) A deterministic filter that has no deterministic minimizer with fewer than 10 states. (b) A minimal nondeterministic 9-state minimizer for the filter above.

Problem: P-filter Minimization (PFM)

Input: A p-filter F.

Output: A p-filter $\text{F}^\dagger$ with fewest states, such that $\text{F}^\dagger$ output simulates F.

This is a generalization of its deterministic version PFDM, which only dealt with deterministic input and deterministic minimizer. We use ‘PF’ to denote the fact that both the input and output of this problem can be general nondeterministic p-filters, and use ‘M’ for minimization. Additionally, we designate the problem of producing a *deterministic minimizer* for a nondeterministic input filter ‘PFDM’, a four-letter word where ‘D’ stands for deterministic.

6.5 Automata and their minimization

In this section, we will present some preliminary results from automata theory.

A nondeterministic finite automaton (NFA) is a tuple $(Q, Q_0, \Sigma, \delta, A)$, where Q, Q_0, Σ, δ, A are the states, initial states, alphabet (observations), transition function, and accepting states. Both filters and NFAs are similar, both being transition structures. Different from a filter, an NFA A has accepting states not outputs (colors). For automata, we are interested in the strings that reach some accepting states, which we term the accepting language $\mathcal{L}^A(\mathtt{A})$. An NFA with a singleton Q_0 and deterministic transition structure is also called a DFA.

Here are some results from automata theory:

Lemma 6.11 (NFA equivalence[75])**.** Given two NFAs $\mathtt{A}$ and $\mathtt{B}$, it is PSPACE-complete to check if $\mathcal{L}^A(\mathtt{A}) = \mathcal{L}^A(\mathtt{B})$.

Lemma 6.12 (NFA universality[76])**.** For a given NFA $\mathtt{A} = (Q, Q_0, \Sigma, \delta, A)$, it is PSPACE-complete to check whether $\mathcal{L}^A(\mathtt{A}) = \Sigma^*$.

Lemma 6.13 (DFA union universality[77])**.** Given a set of DFAs $\{\mathtt{A}_1, \mathtt{A}_2, \ldots, \mathtt{A}_n\}$ with common alphabet Σ, it is PSPACE-complete to check if $\cup_{1\leq i\leq n}\mathcal{L}^A(\mathtt{A}_i) = \Sigma^*$.

6.6 Computational complexity of nondeterministic filter minimization

In this section, we leverage prior hardness results from automata theory to show that the problems of finding minimizers, including deterministic and nondeterministic minimizers, for nondeterministic input filters are hard. Specifically, we will show that the decision version of PFM is PSPACE-complete, and the decision version of PFDM is PSPACE-hard.

The decision problem of the PFM problem is:

Decision Problem: P-filter Minimization (PFM-DEC)

Input: A p-filter $\mathtt{F}$ and $k \in \mathbb{N}^+$.

Output: YES if there exists a p-filter $\mathtt{F}^\dagger$ with no more than k states, such that $\mathtt{F}^\dagger$ output simulates $\mathtt{F}$. NO otherwise.

Similarly, we denote the decision version of PFDM as PFDM-DEC.

6.6.1 The hardness of PFM and PFDM

Now, we will show that the decision versions of PFM and PFDM are, respectively, PSPACE-complete and PSPACE-hard. Consequently, both PFM and PFDM are PSPACE-hard.

It is helpful, as a first step, to introduce a product operator for constructing of the product graph of two filters in polynomial time. This operator will be used to check output simulation requirement.

Definition 6.18 (tensor product)**.** Given two filters $\mathtt{F}_1 = (V^1, V_0^1, Y^1, \tau^1, C^1, c^1)$ and $\mathtt{F}_2 = (V^2, V_0^2, Y^2, \tau^2, C^2, c^2)$, their product, a graph denoted $(\mathtt{F}_1 \odot \mathtt{F}_2)$, is constructed to capture strings in $\mathcal{L}^I(\mathtt{F}_1)$ as follows:

1. List all pairs of vertices in $V(F_1) \times (V(F_2) \cup \{\ominus\})$, where $\ominus$ is a placeholder for an empty vertex.
2. Mark vertex (v, w) an initial state in graph $(\mathtt{F}_1 \odot \mathtt{F}_2)$.
3. Build a transition from (v, w) to (v', w') under label y if $y \in \tau^1(v, v')$ and $y \in \tau^2(w, w')$. Notice that if y is not an outgoing label of vertex v, then we say $y \in \tau^1(v, \ominus)$.
4. Remove the pairs that are not reached from any initial state in $(\mathtt{F}_1 \odot \mathtt{F}_2)$.

Notice that the tensor product of two filters is a transition structure with initial states, i.e., a graph.

Lemma 6.14. PFM-DEC is in PSPACE.

Proof. We show first that representing and searching for a filter takes polynomial space, and then that only polynomial space is needed to ascertain whether a filter output simulates $\mathtt{F} = (V, V_0, Y, \tau, C, c)$. Since PFM-DEC requires we encode filters of size k, we need to keep track of at most k^2 transitions, at most $|Y|$ labels for each transition, at most $|C|$ colors for each state, and at most k initial states. The space needed to enumerate output filters is $O(k^2 \times |Y| + k \times |C|)$.

To show that it also takes polynomial space to check whether a filter $\mathtt{F}'$ output simulates the filter $\mathtt{F}$ that was provided as input, we need to check the language inclusion property and then output consistency.

First, we will show that it takes polynomial space to establish that $\mathcal{L}^I(\mathtt{F}) \subseteq \mathcal{L}^I(\mathtt{F}')$ by converting it to the problem of NFA equivalence, which is in PSPACE. We form product graph $\mathtt{G} = \mathtt{F} \odot \mathtt{F}'$.

If there is no vertex $(v, \ominus)$ in $\mathtt{G}$ such that $v \in V(\mathtt{F})$, then we claim that $\mathcal{L}^I(\mathtt{F}) \subseteq \mathcal{L}^I(\mathtt{F}')$ since every string reaching a vertex in $\mathtt{F}$ also reaches some vertex in $\mathtt{F}'$. If there exists some such a vertex $(v, \ominus)$, then we must determine whether the strings reaching $(v, \ominus)$ also reach some vertex in $\mathtt{F}'$. We build an NFA $\mathtt{A}$ from $\mathtt{G}$ by treating all states $\{(v, \ominus) \mid v \in V(\mathtt{F})\}$ as accepting states. Next, we create another NFA, $\mathtt{B}$, from $\mathtt{F}'$ by treating every state in $\mathtt{F}'$ as accepting. Then we want to show that strings reaching every $(v, \ominus)$ are accepted by $\mathtt{B}$. Or, in other words, whether NFAs $\mathtt{A}$ and $\mathtt{A} \cap \mathtt{B}$ are equivalent (where $\cap$ is automata intersection). Creating automata $\mathtt{A}$ and $\mathtt{A} \cap \mathtt{B}$, and showing their equivalence is in PSPACE.

Secondly, verifying output consistency also needs only polynomial space. Begin by removing the states of the form $(v, \ominus)$ from $\mathtt{G}$. Then, for every state (v, w) in $\mathtt{G}$ such that $c(v) \not\supseteq c(w)$, to output simulate, for every output $o \in c(w) \setminus c(v)$, strings reaching (v, w) must reach some state u in $\mathtt{F}$ with $o \in c(u)$. Otherwise, o is not a legal output for some string, and $\mathtt{F}'$ is not output simulating $\mathtt{F}$. To whether o is a legal output, we build an automaton $\mathtt{M}$ from $\mathtt{G}$ by treating (v, w) as accepting states, and another automaton $\mathtt{N}$ from $\mathtt{F}$ by treating the states with color o as accepting states. If $\mathcal{L}^A(\mathtt{M}) \subseteq \mathcal{L}^A(\mathtt{N})$, then o is safe. If every $o \in c(w) \setminus c(v)$ is safe, then the output of $\mathtt{F}'$ is consistent on that of $\mathtt{F}$. Otherwise, (v, w) is an evidence of violation for output consistency. This procedure also takes polynomial amount of space.

Therefore, PFM-DEC is in PSPACE. □

Lemma 6.15. PFM-DEC is PSPACE-hard.

Proof. We give a polynomial time reduction from NFA universality to PFM-DEC. To show the accepting language of a given NFA $\mathtt{A} = (Q, Q_0, \Sigma, \delta, A)$ is Σ^*, we first create a filter $\mathtt{F}$ from $\mathtt{A}$ as follows:

1. Add the states, transitions, initial states of $\mathtt{A}$ to the states, transitions, initial states of $\mathtt{F}$.
2. Add a new initial state v to $\mathtt{F}$, with a self-loop bearing all labels Σ from $\mathtt{A}$.
3. Add a new vertex w to $\mathtt{F}$. For every state in $\mathtt{F}$ arising from an accepting state in $\mathtt{A}$, add a transition to w under some new label z, where z is not a symbol from Σ.

4. Add one more vertex u to $\mathtt{F}$, and a transition from v to u under z.

5. Color u blue, the others green.

This procedure takes polynomial time.

Now, the interaction language for this filter is $\Sigma^* z$. Further, the outputs of strings $\mathcal{L}^A(\mathtt{A})z$ are both green and blue, while the outputs for the strings $(\Sigma^* \setminus \mathcal{L}^A(\mathtt{A}))z$ are blue only.

If $\mathcal{L}^A(\mathtt{A})$ is Σ^*, then the minimal filter for $\mathtt{F}$ has only one green state and it has a self-loop bearing $\Sigma \cup \{z\}$. If $\mathcal{L}^A(\mathtt{A})$ is not Σ^*, then there there exists some string $s \notin \mathcal{L}^A(\mathtt{A})$ where s only outputs green, and sz only outputs blue. There must, therefore, be at least two states (one colored green, and one colored blue) in its minimizer. As a consequence, if the minimizer of $\mathtt{F}$ has only one state, then $\mathcal{L}^A(\mathtt{A})$ is Σ^*. Otherwise, $\mathcal{L}^A(\mathtt{A})$ is not Σ^*.

Therefore, we get a polynomial time reduction from NFA universality problem to PFM-DEC. PFM-DEC is PSPACE-hard since NFA universality is PSPACE-complete. □

Lemma 6.16. PFM-DEC is PSPACE-complete.

Proof. Combine Lemmas 6.14 and 6.15. □

Theorem 6.4. PFM is PSPACE-hard.

Proof. This is a direct consequence of Lemma 6.16. □

Having considered the case where both the input and the minimizer may be nondeterministic, next we show that limiting nondeterminism to only the input filter (what we dubbed PFDM-DEC) still retains its hardness.

Theorem 6.5. PFDM-DEC is PSPACE-hard.

Proof. We will show the PFDM-DEC is PSPACE-hard by reducing the DFA union universality problem to PFDM-DEC. Given a set of DFAs, $\mathtt{A}_1, \mathtt{A}_2, \ldots, \mathtt{A}_n$, let the union of their alphabet be Σ. The DFA union universality problem is to check $\mathcal{L}^A(\mathtt{A}_1) \cup \mathcal{L}^A(\mathtt{A}_2) \cup \cdots \cup \mathcal{L}^A(\mathtt{A}_n) = \Sigma^*$. For each DFA $\mathtt{A}_i$, we first, we construct a DFA $\mathtt{A}'_i$, such that $\mathcal{L}^A(\mathtt{A}'_i) = \mathcal{L}^A(\mathtt{A}_i)$ and $\mathcal{L}^I(\mathtt{A}'_i) = \Sigma^*$:

1. Initialize $\mathtt{A}'_i$ as a copy of $\mathtt{A}_i$.

2. If $\mathcal{L}^I(\mathtt{A}'_i) \neq \Sigma^*$, then add a trap state v' with a self-loop bearing all labels in Σ for each DFA $\mathtt{A}'_i$. For each state w' in $\mathtt{A}'_i$ and every outgoing event $y \in \Sigma$, if y crashes when traced from w', build a transition from w' in $\mathtt{A}'_i$ to the trap state v' under y.

3. Make all states corresponding to accepting states in $\mathtt{A}_i$ the accepting states for $\mathtt{A}'_i$.

Next, we build an NFA $\mathtt{B}'$ as the union of all these $\mathtt{A}'_i$'s, so as to have $\mathcal{L}^A(\mathtt{A}_1) \cup \mathcal{L}^A(\mathtt{A}_2) \cup \cdots \cup \mathcal{L}^A(\mathtt{A}_n) = \mathcal{L}^A(\mathtt{B}')$. Additionally, no strings in Σ^* crash on $\mathtt{B}'$. The task, then, is to check whether $\mathcal{L}^A(\mathtt{B}') = \Sigma^*$ holds or not.

To do so, create a filter $\mathtt{F}$ from $\mathtt{B}'$ as follows:

1. Add the initial states, states, transitions of $\mathtt{B}'$ to those of $\mathtt{F}$;.

2. Color the copies of the accepting states in $\mathtt{B}'$ green, and the copies of the non-accepting states red.

3. Add one more state, and color it green. Make this state the destination reached from one goal state under a fresh symbol z (i.e., where z is not a symbol from Σ).

By adding the new symbol z, we known that there is some string ending with z which outputs only green in $\mathtt{F}$.

Supposing $\mathtt{H}$ is a deterministic minimizer of $\mathtt{F}$, there are two cases. First, if $\mathtt{H}$ is a one-state filter, then it must be green because there are some strings that must output only green. But then, since the one-state filter output simulates $\mathtt{F}$, every string in Σ^* must output at least green in $\mathtt{F}$. Hence, every string in Σ^* must reach the accepting states in $\mathtt{B}'$, and we conclude $\mathcal{L}^A(\mathtt{B}') = \Sigma^*$. Alternatively, if $\mathtt{H}$ has more than one state, then there is at least one green state and one red state (Otherwise, $\mathtt{H}$ is not minimal.). As a consequence, there must be some string in Σ^* that can only output red. Those strings with only red output never reach the accepting states in $\mathtt{B}'$. So, consequently, $\mathcal{L}^A(\mathtt{B}') \neq \Sigma^*$.

The procedure to solve PFDM-DEC involves checking whether there is a one-state minimizer for $\mathtt{F}$. If there is such a minimizer, then the accepting language of the union of all DFAs is Σ^*. Otherwise, it is not. Having given a polynomial time procedure to reduce the DFA union universality problem to PFDM-DEC, which is itself known to be PSPACE-complete, shows that PFDM-DEC is PSPACE-hard. □

Since PFDM can be no easier than its decision version, we can claim that PFDM is PSPACE-hard in terms of space complexity.

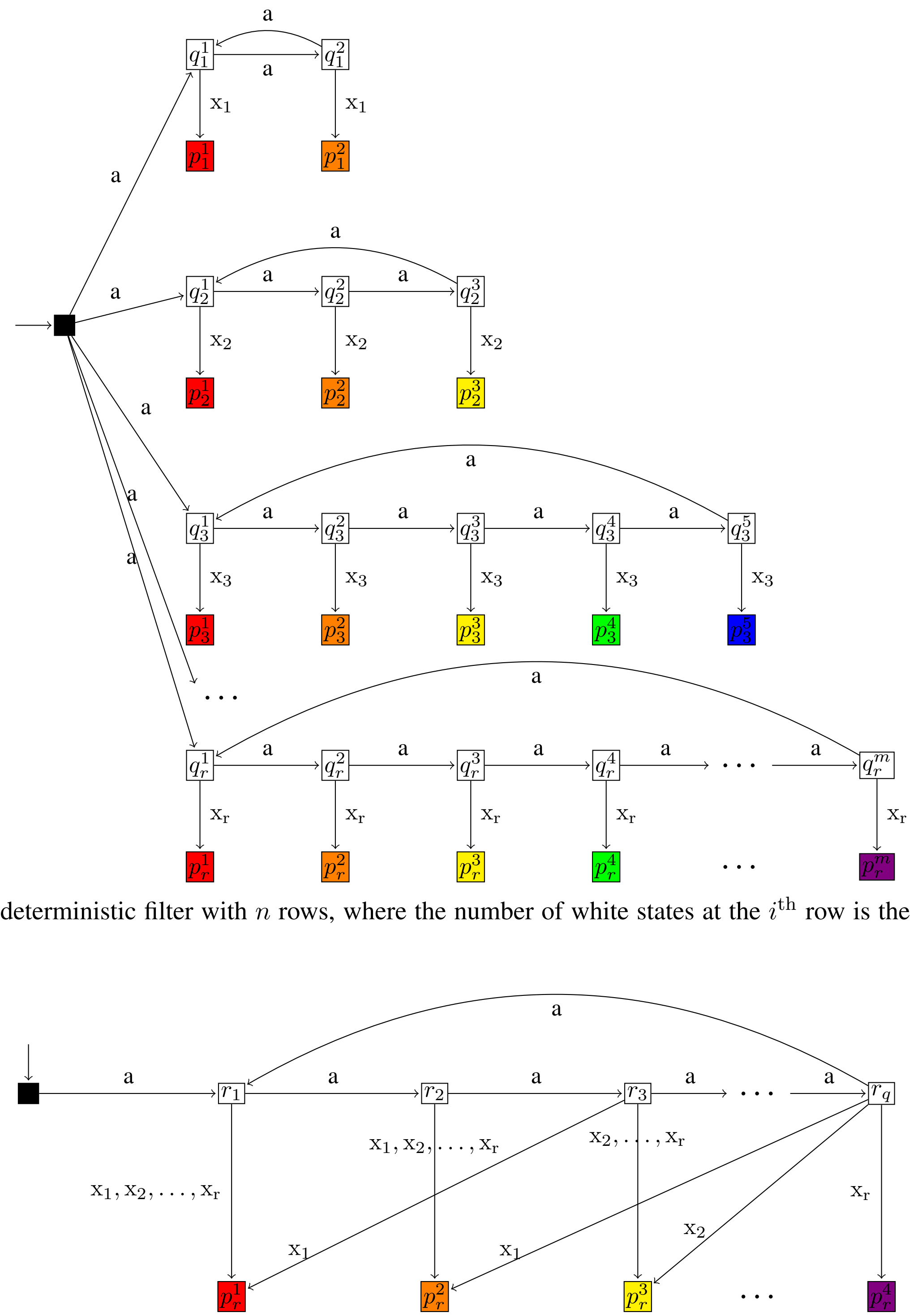
Theorem 6.6. PFDM is PSPACE-hard.

6.6.2 Is PFDM-DEC PSPACE-complete?

It seems natural to suppose that PFDM-DEC is simpler than PFM-DEC and should also be in PSPACE, since the problem is narrower, focusing on more constrained (deterministic) minimizers. However, from the perspective of space consumption, this needn't be the case. The minimizer for PFM is always smaller than (or equal to) the size of the filter provided as input. But this may no longer be true in PFDM as the deterministic minimizer may be larger than the nondeterministic input filter. Take the problem shown in Figure 6.12, and exchange the roles of the two graphs: the filter in Figure 6.12a is a deterministic minimizer for the nondeterministic filter shown in Figure 6.12b. There, the deterministic minimizer has one more state than its nondeterministic input filter.

But how much of a difference can there be? We give a construction for a family of filters demonstrating that the size of the deterministic minimizer may grow so that its size is beyond any polynomial in the input size. First, we make a nondeterministic input filter, then we follow that by giving its deterministic minimizer.

Construction 6.1. Fix some natural number r, and construct the nondeterministic input filter with r rows depicted in Figure 6.13a. Create a cycle of white states under a, where the number of white states at row $i \in \mathbb{N}^+$ is the cycle of length p_i, the i^{th} prime number. For example, the number of white states in rows 1, 2, 3 are 2, 3, 5, respectively. Create a black initial state connects, via a, to one state at each of these r rows. At each row, starting from the state connected with the initial one, build a transition to a new child state. Next, we will color the child state with one color from



(a) A nondeterministic filter with n rows, where the number of white states at the i^{th} row is the i^{th} prime number.

(b) A minimal deterministic filter for (a), where the total number of white states is the product of first n prime numbers.

Figure 6.13: An example to show that the number of states in the deterministic minimizer is larger than polynomial size of the nondeterministic input filter.

the color list $[o_1, o_2, \ldots, o_{p_r}]$, which does not contain black and white. Each child state is colored as the first one that is not chosen in the row.

An equivalent deterministic filter, as shown in Figure 6.13b, is produced via the power set construction. Note that all states in the nondeterministic filter shown in Figure 6.13a that are reached by the same string share the same color. Hence, there is but a single color for each state in the deterministic filter: color the state in the deterministic filter with the color of the corresponding states reached by the same string. (Part of the next lemma will show this to be the minimizer.)

Lemma 6.17. The deterministic minimizer of a nondeterministic input filter can exceed any polynomial of the input size.

Proof. First we argue that the deterministic form of the nondeterministic input filter from Construction 6.1 is a deterministic minimizer, then show that the gap between the size of the nondeterministic input filter and its deterministic minimizer is larger than any polynomial of the input size.

The deterministic filter shown in Figure 6.13b is already a minimal one for the filter depicted in Figure 6.13a. The n colors must be included as they are each produced by some string; the white vertices could only be merged if there was a common divisor in the cycle lengths, but the cycle lengths are all distinct primes. Hence, no pair of states in Figure 6.13b can be merged since they either have different outputs or disagree on the outputs of their common extensions.

Let n be the total number of states in this nondeterministic input filter. Then we have $n = 2 \cdot S(r) + 1$, where $S(r) = \sum_{i=1}^{r} p_i$ is summation of the first r prime numbers, and Bach and Shallit[78] show that $S(r) \sim \frac{1}{2} r^2 \ln r$ holds asymptotically. When $1 < r$, $n = r^2 \ln r + 1 < r^3$.

Let z represent the total number of states in the deterministic filter. Then we have $z = 1 + P(r) + p_r$, where $P(r) = \prod_{i=1}^{r} p_i$ is the primorial denoting the product of the first r prime numbers. According to the prime number theorem and the first Chebyshev function, we have that $P(r) \sim$

$e^{(1+o(1))r\log r}$ and $p_r \sim r\log r$ holds asymptotically. Hence,

$$\begin{aligned} z &= 1 + P(r) + p_r \\ &= 1 + e^{(1+o(1))r\log r} + r\log r > e^r \quad \text{for large } r. \end{aligned}$$

Since $r > \sqrt[3]{n}$, we have $z > e^{\sqrt[3]{n}}$. So we write this lower bound of z as $f(n) = e^{\sqrt[3]{n}} = \sum_{m=0}^{\infty} \frac{n^{\frac{m}{3}}}{m!}$ (Taylor series).

Now consider any polynomial of n of degree k and write it as $g(n,k) = \sum_{m=0}^{k} \alpha_m n^m$. Let $c = \max\{\alpha_0, \alpha_1, \ldots, \alpha_k\}$. If $n > c \cdot (k+1)$, then we have for all $i \leq k$, the coefficients have $\alpha_i n^i < c \cdot n^k$, and the sum $\sum_{i=0}^{i=k} \alpha_i n^i < n^{k+1}$.

To bring the two bounds in relation to one another: when $n > (3k+6)!$, then $f(n) > \frac{n^{\frac{3k+6}{3}}}{(3k+6)!} > \frac{n^{\frac{3k+6}{3}}}{n} = n^{k+1}$, Hence, $f(n) > n^{k+1}$ if $n > (3k+6)!$. Thus for $n > \max\{c \cdot (k+1), (3k+6)!\}$, we have that $z > f(n) > n^{k+1} > g(n,k)$. This is true for any k, so the size of the deterministic minimizer, z, is larger than any polynomial of n.

Therefore, the number of states in the deterministic minimizer can exceed any polynomial of the input size. □

One implication of the preceding example is that:

Lemma 6.18. PFDM is not in P.

Proof. Since the size of the minimizer can be larger than any polynomial of the input size, it takes more than polynomial time to output the minimizer. Therefore, PFDM $\notin$ P. □

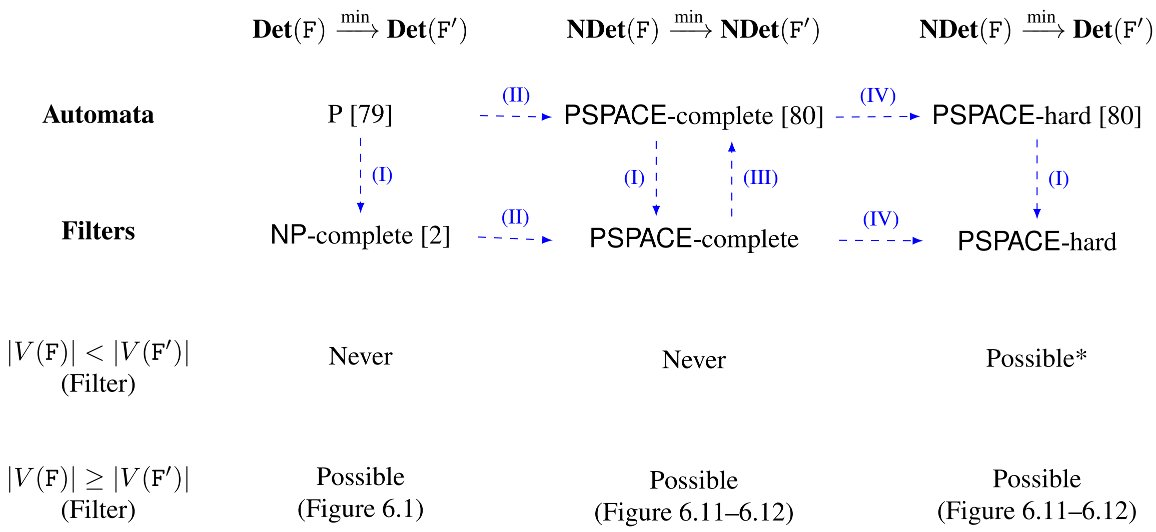
Then, considering time complexity further, we can conclude that PFDM is strictly NP-hard.

Theorem 6.7. PFDM is NP-hard, but not in P.

Proof. The deterministic input to deterministic output filter minimization problem, the decision problem form of which is NP-complete [2], is properly contained in PFDM (one just happens to select an input that is deterministic). We have that PFDM is NP-hard, and combining with Lemma 6.18, we can conclude that PFDM is strictly NP-hard. □

To summarize, Construction 6.1 and Lemma 6.17 show that the gap between the size of the deterministic minimizer can be larger than polynomial of the input size. It indicates that constructing and storing the deterministic minimizer in its entirety to determine its size would disqualify PFDM-DEC from PSPACE. Of course, other cleverer means may exist, so whether PFDM-DEC is PSPACE (as a consequence, PFDM-DEC is PSPACE-complete) or not remains an open question.

6.7 A comparison between automata minimization and filter minimization



	Det$(\mathtt{F}) \xrightarrow{\text{min}}$ **Det**$(\mathtt{F}')$	**NDet**$(\mathtt{F}) \xrightarrow{\text{min}}$ **NDet**$(\mathtt{F}')$	**NDet**$(\mathtt{F}) \xrightarrow{\text{min}}$ **Det**$(\mathtt{F}')$
Automata	P [79]	PSPACE-complete [80]	PSPACE-hard [80]
Filters	NP-complete [2]	PSPACE-complete	PSPACE-hard
$\lvert V(\mathtt{F})\rvert < \lvert V(\mathtt{F}')\rvert$ (Filter)	Never	Never	Possible*
$\lvert V(\mathtt{F})\rvert \geq \lvert V(\mathtt{F}')\rvert$ (Filter)	Possible (Figure 6.1)	Possible (Figure 6.11–6.12)	Possible (Figure 6.11–6.12)

(I) Extra DOF in the output of the crashed strings.

(II) Extra DOF in the behavior of the strings in both $\mathtt{F}$ and $\mathtt{F}'$.

(III) Extra DOF in the output of the states being nondeterministically reached.

(IV) The size of minimizer may be substantially larger than the input size.

* The size of the output filter can be substantially larger than the input size.

Figure 6.14: A comparison between hardness results of decision versions of automata minimization and filter minimization.

With the preceding hardness results for filter minimization problems established, we now compare them with the hardness of automata minimization. It is worthwhile to try distill intuition for a couple of reasons: firstly, the automata hardness results were used in the arguments above, so their relationship might seem obvious at first blush. But the notion of equivalence is quite different, as specifically are requirements on interaction vs. accepting languages. Also, secondly, the initial supposition that the deterministic filter and deterministic automata minimization problems were identical, was wrong.

To help elucidate, we introduce some lightweight notation for the problems, explicitly showing the types of their inputs and outputs. We denote deterministic and nondeterministic structures as Det and NDet respectively. Specifically, we write deterministic and nondeterministic automata as DFA and NFA, and those for filters as DF and NF. Then the minimization problems can be written as $\mathrm{A}\xrightarrow{\text{min}}\mathrm{B}$, which converts a structure of type A to a minimal structure of type B. Figure 6.14 maps the connections using this convention.

Examining the first column: $\mathrm{DFA}\xrightarrow{\text{min}}\mathrm{DFA}$ can be solved efficiently by identifying Myhill—Nerode equivalence classes [79], while the decision version of $\mathrm{DF}\xrightarrow{\text{min}}\mathrm{DF}$ is NP-complete. The main reason for this hardness separation between these two problems is the extra degree of freedom (DOF) for filter minimization. Filters can choose to assign *any* output for the strings that crash in the filter (DOF I). To exploit this degree of freedom optimally, it is equivalent to searching for a minimum clique cover in the compatibility graph of the input filter [81], which makes the problem computationally hard.

The other two columns of Figure 6.14: As we consider nondeterminism in the input or output structure, the hardness separation between automata minimization and filter minimization disappears. Informally speaking, it appears that the hardness arising from DOF I is dominated by other sources of complexity. When nondeterminism appears in both input and output, both $\mathrm{NFA}\xrightarrow{\text{min}}\mathrm{NFA}$ and $\mathrm{NF}\xrightarrow{\text{min}}\mathrm{NF}$ are, as decision problems, PSPACE-complete [80]. For both, there could be multiple states simultaneously reached by the same string (DOF II) though it takes no more than polynomial space to check the outputs of those states. Though both are PSPACE-complete, the problems differ in the degrees of freedom they have—though, clearly, this difference is not enough to manifest as a hardness gap. On the one hand, $\mathrm{NF}\xrightarrow{\text{min}}\mathrm{NF}$ has DOF I while $\mathrm{NFA}\xrightarrow{\text{min}}\mathrm{NFA}$ does not. On the other, $\mathrm{NF}\xrightarrow{\text{min}}\mathrm{NF}$ requires all outputs of all states reached by the string be constrained, whereas

NFA$\xrightarrow{\text{min}}$NFA can choose to accept the states or not, as long as at least one of them is accepted (DOF III),

If we keep nondeterminism in the inputs but remove it from the outputs, the problems do not become any easier. When outputs are restricted to be deterministic, the size of the output can be substantially larger than that of the input filter, (IV). If one were to think of this as a search problem, a more restrictive type can drastically increases the size of the search space. In particular, the size of the minimal DFA for an n-state NFA can be 2^n [82], and the size of the minimal DF for an n-state NF can be larger than any polynomial of n (Figure 6.13). It only ever takes polynomial space for DFA$\xrightarrow{\text{min}}$DFA and DF$\xrightarrow{\text{min}}$DF. But it is unclear whether this holds for NFA$\xrightarrow{\text{min}}$DFA and NF$\xrightarrow{\text{min}}$DF, and the increase in output size is unfavourable, though inconclusive, evidence to the contrary.

6.8 Discussions, open questions, and conjectures

In this section, we will revisit some of the results, build connections to the results from automata theory, propose open questions and conjectures.

6.8.1 Examining the gap between input and output for the problems of minimizing automata and filters

In both the filter minimization and automata minimization, when the input is deterministic, or output is nondeterministic, the minimizer is always no larger than the input. In those problems, the output is always less constrained than the input. But this is not true for the problem of finding a deterministic minimizer for a nondeterministic input. The size of the minimizer can be larger than the input filter. In fact, for the automata version $NFA \xrightarrow{\text{min}} DFA$, the minimizer can be exponentially larger than the input. One example is to treat Figure 6.11a and 6.11b as automata with the states containing label 3 as accepting states. Then Figure 6.11b is the minimal DFA for the NFA in Figure 6.11a. The minimizer has 2^{5-1} states, and is exponentially larger than the input. On the other hand, for the filter version $NF \xrightarrow{\text{min}} DF$, we can only show in Figure 6.13a that the minimizer is larger than any polynomial of the input. Whether the deterministic minimizer can be exponentially larger than the output filter still remains an open question.

Another question is how this gap is being created. In the automata treatment of Figure 6.11, the exponential gap is constructed due to the rollout of the self-loop at the initial state. Similarly in the minimization of nondeterministic Moore Machine, an example of a nondeterministic Moore Automaton with a self-loop at the initial state is given in [83, Fig. 6], and the deterministic minimizer of this example is also exponentially larger than the input. However, the strategy of looping at the initial state does not work with filter minimization. The filter minimization algorithm may choose to simply cut the nondeterministic edges, and obtain a minimizer with the same or smaller size. An example is given in Figure 6.11c. We conjecture that the self-loops at the initial states of the filter do not create an exponential gap between the filter and its minimizer.

6.8.2 Special cases

We will present some special but valuable automata and discuss about their implications to filter and filter minimization.

6.8.2.1 Coherent Moore Automata and string single-outputting filters

A particular type of automaton, named coherent Nondeterministic Moore Automaton or c-NMA, is proposed to allow limited nondeterminism in the output of the strings [83]. In a c-NMA, all states reached by the same string must agree on their output. A deterministic Moore Machine is a special c-NMA, since there is at most one state being reached by each string. But the nondeterminism makes c-NMA more succinct than a deterministic Moore Machine to represent the same input and output.

A similar object to c-NMA is the string single-outputting filter, which is defined below:

Definition 6.19 (string single-outputting)**.** A filter $\mathtt{F}$ is string single-outputting if $\forall s \in \mathcal{L}^I(\mathtt{F})$, $|\mathcal{C}(\mathtt{F}, s)| = 1$.

The filter shown in Figure 6.13a is a string single-outputting filter, and its deterministic minimizer in Figure 6.13b is larger than any polynomial of the size of the input filter.

The string single-outputting filters are interesting because they comprise the solution set (or space) of nondeterministic minimizers for deterministic vertex single-outputting filters, which are also string single-outputting.

Lemma 6.19. There is always a string single-outputting minimizer for the problem PFM with string single-outputting input filters.

Proof. Given an input filter $\mathtt{F}$ and its minimizer $\mathtt{F}'$, if $\mathtt{F}'$ is not string single-outputting, then we will construct a string single-outputting minimizer $\mathtt{F}''$ and prove $\mathtt{F}''$ is also minimizer. First, the initial states of $\mathtt{F}'$ must share the same output. Second, every state in $\mathtt{F}'$ must have exactly one output. Otherwise, the states with multiple outputs can be removed and $\mathtt{F}'$ is not minimal. Third, for every state v' in $\mathtt{F}'$ and every label y, if it transitions to vertices with different outputs under y, then keep an arbitrary edge transitioning from v'' under y. This procedure will give a new filter $\mathtt{F}''$. In addition, we have $\mathcal{L}^I(\mathtt{F}') = \mathcal{L}^I(\mathtt{F}'')$, and $\mathcal{C}(\mathtt{F}'', s) \subseteq \mathcal{C}(\mathtt{F}', s)$.

Next, we claim $\mathtt{F}''$ is also a minimal solution. Otherwise, there exists a string that does not have the correct output. If there is a string $s \notin \mathcal{L}^I(\mathtt{F}'')$ such that $\mathcal{C}(\mathtt{F}, s) \not\supseteq \mathcal{C}(\mathtt{F}'', s)$, then we know $\mathcal{C}(\mathtt{F}, s) \not\supseteq \mathcal{C}(\mathtt{F}', s)$, which contradicts with the fact that $\mathtt{F}'$ is a minimal solution. □

One question is about the computational complexity of reducing string single-outputting filters. Since the problem of minimizing a string single-outputting filter falls into the problem PFM, where the input does not need to be string single-outputting, the space complexity of minimizing a string single-outputting filter is in PSPACE. One may ask what is the time complexity? Is the problem in NP? Or can the problem of verifying the output simulating property of a solution on a string single-outputting filter be solved in polynomial time? Unfortunately, we are not aware of any algorithm to verify both language inclusion and output consistency in polynomial time. It remains to be an open question to show whether the minimization of a string single-outputting input filter is NP or not.

6.8.2.2 Automata and filters with finite languages

In automata theory, there is also work studying automata with finite accepting languages. A classical method to specify a finite language is to construct an acyclic deterministic finite automaton, and such automata can be minimized as a trie [84]. A new type of structure, called cover automata, was introduced by Câmpeanu, Santean and Yu [85]. These automata have accepting languages, where the strings are restricted to be within some finite length. Compared to classical finite automata, finite cover automata have an extra degree of freedom to include strings longer than a particular length and those extra strings will not affect its accepting language. By including those extra strings, a cover finite automaton can be potentially smaller than the acyclic finite deterministic automata. There exist efficient algorithms to minimize the finite cover automata [85].

The ability to include extra strings is akin to the adjustment from language equality to language inclusion, which has been the working requirement for all the filter minimization problems studied here. In filter minimization, the filters with finite interaction languages have been studied as tree filters, and the minimization of a tree filter is NP-hard [63]. This complexity difference stimulates one to think that there are some special additional properties on the tree filter that might reduce computational complexity of the tree filter minimization to be in polynomial time. In the minimization of a cover automata, it can only introduce additional strings that are longer than a particular length. This motivates us to define a tree filter such that none of the strings within a particular length crashes on the filter:

Definition 6.20 (complete proper tree filter)**.** Given a deterministic filter $\mathtt{F}$ with a finite interaction language, let $\ell = \max\{|s| \mid s \in \mathcal{L}^I(\mathtt{F})\}$ and Σ be the alphabet, then $\mathtt{F}$ is called a complete proper tree filter, if (i) every state in $\mathtt{F}$ has a unique parent, (ii) the incoming edge bears only one label, and (iii) $\Sigma^\ell \subseteq \mathcal{L}^I(\mathtt{F})$, where Σ^ℓ is the set of all strings within length ℓ.

We define a level function $L_{\mathtt{F}}(v)$ for each state v in a complete proper tree filter F: (i) for the $v_0 \in V_0(\mathtt{F})$, $L_{\mathtt{F}}(v_0) = 0$; (ii) for $v, w \in V(\mathtt{F})$, $L_{\mathtt{F}}(v) = L_{\mathtt{F}}(w) + 1$ if there exists a label y such that $y \in \tau(w, v)$. In a complete proper tree filter, the level function is complete and well-defined.

Property 6.1. In a deterministic complete proper tree filter $\mathtt{F}$, for any state $v, w \in V(\mathtt{F})$, their levels have implications for the extensions: (i) if $L_{\mathtt{F}}(v) < L_{\mathtt{F}}(w)$, then $\mathcal{L}^e_F(v) \supset \mathcal{L}^e_F(w)$; (ii) if $L_{\mathtt{F}}(v) = L_{\mathtt{F}}(w)$, then $\mathcal{L}^e_F(v) = \mathcal{L}^e_F(w)$;

Lemma 6.20. To find a minimizer for a vertex single-outputting complete proper tree filter, we only need to look for an equivalence relation.

Proof. In general, a deterministic minimizer of a deterministic filter is induced by a vertex cover. Given a cover that yields a minimizer, we will construct a partition by picking an arbitrary group for the states that are contained in multiple groups in the cover. By doing this, we will get a partition and all states in each equivalence class are mutually compatible. Accordingly, merging the states in the same equivalence class, we can obtain a deterministic filter with the same size as that of the minimizer. The new filter has to output simulate the input filter. Otherwise, the given minimizer fails to output simulate the input filter. Therefore, the deterministic filter constructed from the partition is also a minimizer. □

We then extend the level function from a state to a set of states, such that the level of a set is the minimum level from all states in the set, i.e., $L_{\mathtt{F}}(V) = \min\{L_{\mathtt{F}}(v) \mid v \in V\}$.

Definition 6.21 (greedy partition)**.** A partition $\boldsymbol{K} = \{K_1, K_2, \dots, K_m\}$ on a vertex single-outputting complete proper tree filter $\mathtt{F}$, is a *greedy partition*, if (i) the states in each subset K_i are mutually compatible, (ii) there does not exist a state $v \in K_i$, such that v is compatible with all states from a subset K_j with a smaller level, i.e., $K_i \neq K_j$, $L_{\mathtt{F}}(K_j) < L_{\mathtt{F}}(v)$, and $\forall w \in K_j, v \sim_c w$.

Lemma 6.21. Given a vertex single-outputting complete proper tree filter $\mathtt{F}$ and three states v_1, v_2, $v_3 \in V(\mathtt{F})$ such $L_{\mathtt{F}}(v_1) < L_{\mathtt{F}}(v_2)$ and $L_{\mathtt{F}}(v_1) < L_{\mathtt{F}}(v_2)$, if $v_1 \sim_c v_2$ and $v_1 \sim_c v_3$, then $v_2 \sim_c v_3$.

Lemma 6.22. There is always a greedy partition on a vertex single-outputting complete proper tree filter that yields a minimizer.

Proof. According to Lemma 6.20, there is always a minimizer that is induced from a partition. Given any partition, if it is not greedy, we can move the states to the partition with the smallest index and create a greedy partition. The filter induced from the greedy partition must be deterministic, have the same size, and output simulate the input filter. Otherwise, the given minimizer is not a solution. As a consequence, the greedy partition also yields a minimizer. □

Now, let's pick one representative for each equivalence class in the partition, the state with the minimum level. If there are several states with the same level, we pick an arbitrary one. The states with the minimum level are the representatives, since their extensions consist of all those from the states in the equivalence classes following Property 6.1:

Property 6.2. For each equivalence class $[v]$ in a complete proper tree filter $\mathtt{F}$ with v as the representative, $\mathcal{L}^e_{\mathtt{F}}(v) = \cup_{w \in [v]} \mathcal{L}^e_{\mathtt{F}}(w)$.

Lemma 6.23. In a greedy partition $\boldsymbol{K} = \{K_1, K_2, \ldots, K_n\}$ on a vertex single-outputting complete proper tree filter, the representatives from different equivalence classes must be mutually incompatible.

Proof. Suppose v_i and v_j are representatives from equivalence classes K_i and K_j. Each of v_i and v_j contains all the extensions of the states in their equivalence classes. If they are compatible, then all states in K_i and K_j are mutually compatible, which contradicts with the fact that $\boldsymbol{K}$ is a greedy partition. □

Lemma 6.24. All greedy partitions on a vertex single-outputting complete proper tree filter must share the same size.

Proof. Suppose there are two greedy partitions $\boldsymbol{K_1}$ and $\boldsymbol{K_2}$, where $|\boldsymbol{K_1}| < |\boldsymbol{K_2}|$. From $\boldsymbol{K_2}$, we have $|\boldsymbol{K_2}|$ mutually incompatible representatives. Two of them must be in the same equivalence class in $\boldsymbol{K_1}$ according to the pigeon hole principle, which contradicts with the fact that $\boldsymbol{K_1}$ respects the compatibility relation. □

Theorem 6.8. Minimizing a vertex single-outputting complete proper tree filter is equivalent to finding a greedy partition, which can be done in polynomial time.

Proof. According to Lemma 6.22 and 6.24, we can construct a vertex single-outputting complete proper tree filter minimizer from any greedy partition. In addition, finding a greedy partition can be done in polynomial time. □

6.9 Summary

In this section, we revisited and corrected existing ideas for combinatorial filter minimization, by proposing a single-step compatibility relation, zipper constraints to enforce determinism, and vertex cover representation to model both merging and split operations. These new ideas contribute to the first known complete and practical algorithm to minimize a combinatorial filter. We further generalized the notion of compatibility relation to minimizing vertex multi-outputting filters. Next, we examined the hardness results for a family of filter minimization problems, and depicted the degrees of freedom that distinguish them from classical automata minimization. Due to the close connection between filters and automata, we examined and compared their input-output gap, discussed special filters inspired by coherent Moore Automata and cover automata, and the minimization problems on these special filters.

7. CONCLUSION AND FUTURE WORK

This dissertation examines privacy-preserving estimation and planning tasks from the design perspective. It extends the existing estimation and planning tasks by stipulating the information that is disclosed by or learned from the robot, and contributes algorithms to search for feasible solutions for the robot's estimation and planning tasks from the designer's perspective. It also contributes hardness results and algorithms to minimize the resource used by the estimators.

We first study privacy-preserving tracking problems, where the robot can strategically change its sensor configuration to manipulate its estimation in a certain way. By fully characterizing a class of privacy-preserving tracking strategies, we present impossibility results regarding the robot's tracking and privacy bounds, we show that the feasibility of privacy-preserving tracking is sensitive to the robot's initial belief, and we prove the asymptotical results of privacy-preserving tracking with respect to the robot's sensing power.

Then we turn to the privacy-preserving planning problems, where the robot is interacting with the world and their interaction is being observed by an observer via a side-channel. We model the imperfection of the side channel as an information disclosure policy, and develop algorithms to jointly search for both plans and information disclosure policies. Here, the information from the side channel is divulged in an online fashion. We also model the prior knowledge that is disclosed to the observer offline, and examine its implications on the observer's estimate. We further develop algorithms to search for the solutions under different prior knowledge for the observer, even the plan to be sought is the one that is disclosed.

We also examine sensor design in the privacy-preserving planning problems. The sensor is modeled as a generalized information disclosure policy between the world and the robot, capturing the imperfections about the robot's observations of the world state. We abstract the senors as covers, and propose an upper cover representation as a representative for a set of sensors. This allows us to enumerate all abstract sensors that suffice for a privacy-preserving planning task. The sensor fabrication constraints are also pushed to the sensor enumeration process.

In the privacy-preserving task, the observer uses a combinatorial filter to construct an estimate. We observe that the structure of the filter used by the observer has a significant impact on its estimate. To understand the filter structure and its functionality, we study the filters from the resource minimization perspective, where the objective is to minimize the state complexity of the filter while preserving its functionality. We revisit and correct the existing ideas for filter minimization, and propose the first known complete and practical algorithm for both classical vertex single-outputting filter minimization problem and its vertex multi-outputting version. We further examine the complexity for a family of nondeterministic filter minimization problems, and depict the degrees of freedom that can be exploited by the algorithm to minimize a filter versus an automaton. We observe that those degrees of freedom create a complexity gap between deterministic minimization problems, but are not strong enough to create a gap between the nondeterministic ones. Among the nondeterministic minimization problems, there is a special class of problems where the minimizer is larger than the input filter. We examine the gap size for such problems in both automata and filter minimization, and discuss how to construct such gap. We are also inspired by special cases from automata minimization, and discuss about the minimization of special filters including string single-outputting filters and complete proper tree filters.

For future work, we would like to consider the computational complexity of finding nondeterministic minimizers for both deterministic filters and string single-outputting filters. These two classes of filter minimization problems fall into the category of nondeterministic filter minimization problems, which are proved to be PSPACE-hard. But they also possess extra properties, properties of both their inputs and their minimizers. For example, both of their minimizers are string single-outputting. It is not clear whether these extra properties will make them different from general nondeterministic filter minimization. In particular, we would like to ask whether these two problems are in NP. In addition to the complexity results, we would also like to develop algorithms for nondeterministic filter minimization problems. To solve a PSPACE-hard problem, we need to reduce it to the existing PSPACE-complete ones, such as nondeterministic automata minimization. Similar to the deterministic filter minimization problems, one important part is to efficiently specify the constraints for output simulating in nondeterministic filters, without explicitly determinizing them.

Another thread is to extend filter minimization toward plan minimization. In planning problems, plans are represented as graphs. The number of states in the plan has an impact on the amount of resources used to execute this plan or to store the plan. Plan minimization is either to reduce the number of states in an existing plan, or to find a plan with the smallest number of states that achieves the goal in the world. Finding the smallest plan in the world can be treated as compressing a plan closure while avoiding cycles. One may think that plans can be treated as filters where the observations are the inputs and actions are the outputs. In a plan, the future observations depend on which action is chosen. However, this is not true in filters where the future events only depend on the states rather than the output of the states. On the other hand, plan minimization is different from filter minimization in terms of the extra strings the algorithm can introduce: (i) plan minimization can only include additional strings that start with new observations, since introducing strings starting with new actions will make the plan unsafe in the world; (ii) it only needs to choose at least one action if there are multiple actions available at each action state. To understand the connections and differences between plan minimization and filter minimization, plan minimization algorithms have to be developed.

REFERENCES

[1] J. M. O'Kane, "On the value of ignorance: Balancing tracking and privacy using a two-bit sensor," in *Algorithmic Foundation of Robotics VIII*, pp. 235–249, Springer International Publishing, 2010.

[2] J. M. O'Kane and D. A. Shell, "Concise planning and filtering: hardness and algorithms," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 4, pp. 1666–1681, 2017.

[3] B. Tovar, F. Cohen, L. Bobadilla, J. Czarnowski, and S. M. Lavalle, "Combinatorial filters: Sensor beams, obstacles, and possible paths," *ACM Transactions on Sensor Networks*, vol. 10, no. 3, pp. 1–32, 2014.

[4] S. Särkkä, *Bayesian filtering and smoothing*. USA: Cambridge University Press, 2013.

[5] R. Bajcsy, "Active perception," *Proceedings of the IEEE*, vol. 76, no. 8, pp. 966–1005, 1988.

[6] S. M. LaValle, *Planning algorithms*. USA: Cambridge University Press, 2006.

[7] L. Vaas, "Privacy dust-up as Roomba maker mulls selling maps of users' homes." Available at `https://nakedsecurity.sophos.com/2017/07/26/privacy-dust-up-as-roomba-maker-mulls-selling-maps-of-users-homes/`, 2017.

[8] S. Keren, A. Gal, and E. Karpas, "Privacy preserving plans in partially observable environments," in *Proceedings of International Joint Conference on Artificial Intelligence*, (New York, NY), pp. 3170–3176, 2016.

[9] Y.-C. Wu and S. Lafortune, "Synthesis of insertion functions for enforcement of opacity security properties," *Automatica*, vol. 50, no. 5, pp. 1336–1348, 2014.

[10] Y. Song, C. X. Wang, and W. P. Tay, “Privacy-aware kalman filtering,” in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, (Calgary, Canada), pp. 4434–4438, 2018.

[11] J. M. O’Kane and S. M. LaValle, “On comparing the power of robots,” *International Journal of Robotics Research*, vol. 27, no. 1, pp. 5–23, 2008.

[12] F. Z. Saberifar, S. Ghasemlou, D. A. Shell, and J. M. OKane, “Toward a language-theoretic foundation for planning and filtering,” *International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 236–259, 2019.

[13] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of applied cryptography*. USA: CRC Press, Inc., 1st ed., 1996.

[14] C. Gentry, *A fully homomorphic encryption scheme*. PhD thesis, Stanford, CA, USA, 2009.

[15] C. Dwork, “Differential privacy: A survey of results,” in *Proceedings of International Conference on Theory and Applications of Models of Computation*, (Xi’an, China), pp. 1–19, 2008.

[16] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: A distributed anonymous information storage and retrieval system,” in *Designing Privacy Enhancing Technologies*, (Berlin, Heidelberg), pp. 46–66, 2001.

[17] P. Kamat, Y. Zhang, W. Trappe, and C. Ozturk, “Enhancing source-location privacy in sensor network routing,” in *Proceedings of IEEE International Conference on Distributed Computing Systems*, (Columbus, OH), pp. 599–608, 2005.

[18] P. Kamat, W. Xu, W. Trappe, and Y. Zhang, “Temporal privacy in wireless sensor networks: Theory and practice,” *ACM Transactions on Sensor Networks*, vol. 5, no. 4, p. 28, 2009.

[19] S. Guha, K. Tang, and P. Francis, “Noyb: Privacy in online social networks,” in *Proceedings of ACM Workshop on Online Social Networks*, (Seattle, WA), pp. 49–54, 2008.

[20] S. V. Albrecht and P. Stone, “Autonomous agents modelling other agents: A comprehensive survey and open problems,” *Artificial Intelligence*, vol. 258, pp. 66–95, 2018.

[21] T. Chakraborti, A. Kulkarni, S. Sreedharan, D. E. Smith, and S. Kambhampati, “Explicability? legibility? predictability? transparency? privacy? security? the emerging landscape of interpretable agent behavior,” in *Proceedings of the International Conference on Automated-Planning and Scheduling*, (Berkeley, CA), pp. 86–96, 2019.

[22] “Special Issue of *Artificial Intelligence* on Autonomous Agents Modelling Other Agents (Eds.: S. V. Albrecht, P. Stone, and M. Wellman),” 2020. Call for submissions archived and available at `https://web.archive.org/web/20180901231812/https://www.journals.elsevier.com/artificial-intelligence/call-for-papers/special-issue-on-autonomous-agents-modelling-other-agents`.

[23] M. Wooldridge and A. Lomuscio, “Multi-agent $\mathcal{VSK}$ logic,” in *Proceedings of European Workshop on Logics in Artificial Intelligence*, (Berlin, Heidelberg), pp. 300–312, Springer-Verlag, 2000.

[24] F. Kominis and H. Geffner, “Beliefs in multiagent planning: From one agent to many,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, (Jerusalem, Israel), 2015.

[25] J. M. O’Kane and D. A. Shell, “Automatic design of discreet discrete filters,” in *Proceedings of IEEE International Conference on Robotics and Automation*, (Seattle, WA), pp. 353–360, 2015.

[26] Z. Zhang, L. Zhou, and P. Tokekar, “Strategies to design signals to spoof kalman filter,” in *Proceedings of American Control Conference*, (Milwaukee, WI), pp. 5837–5842, IEEE, 2018.

[27] J. Le Ny, “Differentially private kalman filtering,” in *Differential Privacy for Dynamic Data*, pp. 55–75, Springer-Verlag, 2020.

[28] S. Eick and A. I. Antón, "Enhancing privacy in robotics via judicious sensor selection," in *Proceedings of IEEE International Conference on Robotics and Automation*, (Paris, France), pp. 7156–7165, 2020.

[29] Y.-C. Wu, V. Raman, S. Lafortune, and S. A. Seshia, "Obfuscator synthesis for privacy and utility," in *Proceedings of International Symposium on NASA Formal Methods*, (New York, NY), pp. 133–149, 2016.

[30] S. Mohajerani, Y. Ji, and S. Lafortune, "Efficient synthesis of edit functions for opacity enforcement using bisimulation-based abstractions," in *Proceedings of IEEE Conference on Decision and Control*, (Miami Beach, FL), pp. 4849–4854, 2018.

[31] R. I. Brafman, "A privacy preserving algorithm for multi-agent planning and search," in *Proceedings of International Joint Conference on Artificial Intelligence*, (Buenos Aires, Argentina), 2015.

[32] S. Maliah, G. Shani, and R. Stern, "Collaborative privacy preserving multi-agent planning," in *Proceedings of International Conference on Autonomous Agents and Multi-agent Systems*, (São Paulo, Brazil), pp. 493–530, 2017.

[33] L. Takayama, D. Dooley, and W. Ju, "Expressing Thought: Improving Robot Readability with Animation Principles," in *Proceedings of International Conference on Human-Robot Interaction*, (Lausanne, Switzerland), pp. 69–76, 2011.

[34] A. D. Dragan, "Robot Planning with Mathematical Models of Human State and Action." arXiv preprint arXiv:1705.04226, 2017.

[35] R. A. Knepper, C. I. Mavrogiannis, J. Proft, and C. Liang, "Implicit Communication in a Joint Action," in *Proceedings of International Conference on Human-Robot Interaction*, (Vienna, Austria), pp. 283–292, 2017.

[36] A. D. Dragan, R. Holladay, and S. S. Srinivasa, "Deceptive Robot Motion: Synthesis, Analysis and Experiments," *Autonomous Robots*, vol. 39, no. 3, pp. 331–345, 2015.

[37] P. Masters and S. Sardina, “Deceptive Path-Planning,” in *Proceedings of International Joint Conference on Artificial Intelligence*, (Melbourne, Australia), pp. 4368–4375, 2017.

[38] A. Kulkarni, M. Klenk, S. Rane, and H. Soroush, “Resource bounded secure goal obfuscation,” in *AAAI Fall Symposium on Integrating Planning, Diagnosis and Causal Reasoning*, (Arlington, VA), 2018.

[39] A. Kulkarni, S. Srivastava, and S. Kambhampati, “A unified framework for planning in adversarial and cooperative environments,” in *Proceedings of AAAI Conference on Artificial Intelligence*, vol. 33, (Honolulu, HI), pp. 2479–2487, 2019.

[40] J. F. Fisac, C. Liu, J. B. Hamrick, S. Sastry, J. K. Hedrick, T. L. Griffiths, and A. D. Dragan, “Generating plans that predict themselves,” in *Algorithmic Foundations of Robotics XII*, pp. 144–159, Springer International Publishing, 2020.

[41] A. Q. Nilles, D. A. Shell, and J. M. O’Kane, “Robot design: Formalisms, representations, and the role of the designer,” in *Proceedings of IEEE ICRA Workshop on Autonomous Robot Design*, (Brisbane, Australia), May 2018. `https://arxiv.org/abs/1806.05157`.

[42] S. Fuller, E. Wilhelm, and J. Jacobson, “Ink-jet printed nanoparticle microelectromechanical systems,” *Journal of Microelectromechanical Systems*, vol. 11, no. 1, pp. 54–60, 2002.

[43] A. M. Hoover and R. S. Fearing, “Fast scale prototyping for folded millirobots,” in *Proceedings of IEEE International Conference on Robotics and Automation*, (Pasadena, CA), pp. 886–892, 2008.

[44] I. Fitzner, Y. Sun, V. Sachdeva, and S. Revzen, “Rapidly prototyping robots: Using plates and reinforced flexures,” *IEEE Robotics & Automation Magazine*, vol. 24, no. 1, pp. 41–47, 2017.

[45] J. Ziglar, R. Williams, and A. Wicks, “Context-aware system synthesis, task assignment, and routing,” *arXiv preprint arXiv:1706.04580*, 2017.

[46] K. S. Luck, J. Campbell, M. Jansen, D. Aukes, and H. B. Amor, “From the lab to the desert: Fast prototyping and learning of robot locomotion,” in *Proceedings of Robotics: Science and Systems*, (Cambridge, MA), 2017.

[47] A. Schulz, C. Sung, A. Spielberg, W. Zhao, R. Cheng, E. Grinspun, D. Rus, and W. Matusik, “Interactive robogami: An end-to-end system for design of robots with ground locomotion,” *International Journal of Robotics Research*, vol. 36, no. 10, pp. 1131–1147, 2017.

[48] A. M. Mehta, J. DelPreto, K. W. Wong, S. Hamill, H. Kress-Gazit, and D. Rus, “Robot creation from functional specifications,” in *Robotics Research: Volume 2*, pp. 631–648, Springer International Publishing, 2018.

[49] A. Censi, “A Class of Co-Design Problems With Cyclic Constraints and Their Solution,” *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 96–103, 2017.

[50] A. Pervan and T. D. Murphey, “Low complexity control policy synthesis for embodied computation in synthetic cells,” in *Algorithmic Foundations of Robotics XIII*, pp. 602–618, Springer International Publishing, 2020.

[51] M. V. Law, N. Dhawan, H. Bang, S.-Y. Yoon, D. Selva, and G. Hoffman, “Side-by-side human–computer design using a tangible user interface,” in *Proceedings of International Conference on Design Computing and Cognition*, (Milan, Italy), pp. 155–173, Springer-Verlag, 2018.

[52] Y. L. Z. Shi, H. Bang, G. Hoffman, D. Selva, and S.-Y. Yoon, “Cognitive style and field knowledge in complex design problem-solving: A comparative case study of decision support systems,” in *Proceedings of International Conference on Design Computing and Cognition*, (Milan, Italy), pp. 341–360, Springer-Verlag, 2018.

[53] T. Kirk, R. Malak, and R. Arroyave, “Computational Design of Compositionally Graded Alloys for Property Monotonicity,” *Journal of Mechanical Design*, vol. 143, 11 2020. 031704.

[54] M. A. Erdmann, “Understanding action and sensing by designing action-based sensors,” *International Journal of Robotics Research*, vol. 14, no. 5, pp. 483–509, 1995.

[55] M. A. Erdmann and M. T. Mason, “An exploration of sensorless manipulation,” *IEEE Journal on Robotics and Automation*, vol. 4, no. 4, pp. 369–379, 1988.

[56] M. T. Mason, “Kicking the Sensing Habit,” *AI Magazine*, vol. 14, no. 1, pp. 58–59, 1993.

[57] B. R. Donald, “On Information Invariants in Robotics,” *Artificial Intelligence — Special Volume on Computational Research on Interaction and Agency, Part 1*, vol. 72, no. 1–2, pp. 217–304, 1995.

[58] S. Ghasemlou, J. M. O’Kane, and D. A. Shell, “Delineating boundaries of feasibility between robot designs,” in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System*, (Madrid, Spain), pp. 422–429, 2018.

[59] F. Z. Saberifar, S. Ghasemlou, J. M. O’Kane, and D. A. Shell, “Set-labelled filters and sensor transformations,” in *Proceedings of Robotics: Science and Systems*, (Ann Arbor, MI), 2016.

[60] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. USA: MIT press, 2018.

[61] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. USA: Addison-Wesley Longman Publishing Co., Inc., 3 ed., 2006.

[62] J. M. O’Kane and D. A. Shell, “Automatic reduction of combinatorial filters,” in *Proceedings of IEEE International Conference on Robotics and Automation*, (Karlsruhe, Germany), pp. 4082–4089, 2013.

[63] F. Z. Saberifar, A. Mohades, M. Razzazi, and J. M. O’Kane, “Combinatorial filter reduction: Special cases, approximation, and fixed-parameter tractability,” *Journal of Computer and System Sciences*, vol. 85, pp. 74–92, 2017.

[64] H. Rahmani and J. M. O’Kane, “Integer linear programming formulations of the filter partitioning minimization problem,” *Journal of Combinatorial Optimization*, vol. 40, pp. 431–453, 2020.

[65] H. Rahmani and J. M. O'Kane, "On the relationship between bisimulation and combinatorial filter reduction," in *Proceedings of IEEE International Conference on Robotics and Automation*, (Brisbane, Australia), pp. 7314–7321, 2018.

[66] S. M. LaValle, "Sensor lattices: A preimage-based approach to comparing sensors," tech. rep., Department of Computer Science, Sept. 2011. Available at `http://msl.cs.uiuc.edu/~lavalle/papers/Lav12.pdf`.

[67] S. M. LaValle, "Sensing and filtering: A fresh perspective based on preimages and information spaces," *Foundations and Trends in Robotics*, vol. 1, no. 4, pp. 253–372, 2010.

[68] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. USA: Addison-Wesley Longman Publishing Co., Inc., 1984.

[69] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuxmv symbolic model checker," in *Proceedings of International Conference on Computer Aided Verification*, (Vienna, Austria), pp. 334–342, Springer-Verlag, 2014.

[70] S. Ghasemlou and J. M. O'Kane, "Accelerating the construction of boundaries of feasibility in three classes of robot design problems," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System*, (Macau, China), pp. 2532–2538, 2019.

[71] I. Méndez-Díaz and P. Zabala, "A cutting plane algorithm for graph coloring," *Discrete Applied Mathematics*, vol. 156, no. 2, pp. 159–179, 2008.

[72] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 7.0," ZIB-Report 20-10, 2020.

[73] LLC Gurobi Optimization, "Gurobi optimizer reference manual," 2020.

[74] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020," in *Proceedings of SAT Competition 2020 – Solver and Benchmark Descriptions*, vol. B-2020-1, pp. 51–53, University of Helsinki, 2020.

[75] L. Stockmeyer and A. Meyer, "Word problems requiring exponential time: preliminary report," in *Proceedings of Symposium on Theory of Computing*, (Austin, TX), pp. 1–9, 1973.

[76] A. R. Meyer and L. J. Stockmeyer, "The equivalence problem for regular expressions with squaring requires exponential space," in *Proceedings of Symposium on Switching and Automata Theory*, (College Park, MD), pp. 125–129, 1972.

[77] N. Rampersad, J. Shallit, and Z. Xu, "The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages," *Fundamenta Informaticae*, vol. 116, no. 1-4, pp. 223–236, 2012.

[78] E. Bach, J. O. Shallit, J. Shallit, and S. Jeffrey, *Algorithmic number theory: Efficient algorithms*, vol. 1. USA: MIT press, 1996.

[79] B. Mikolajczak, *Algebraic and structural automata theory*. Annals of Discrete Mathematics, Elsevier Science Inc., 1991.

[80] T. Jiang and B. Ravikumar, "Minimal NFA problems are hard," *SIAM Journal on Computing*, vol. 22, no. 6, pp. 1117–1141, 1993.

[81] Y. Zhang and D. A. Shell, "Cover combinatorial filters and their minimization problem," in *Algorithmic Foundations of Robotics XIV*, pp. 90–106, Springer International Publishing, 2021.

[82] A. R. Meyer and M. J. Fischer, "Economy of description by automata, grammars, and formal systems," in *Proceedings of Symposium on Switching and Automata Theory*, (East Lansing, MI), pp. 188–191, 1971.

[83] G. Castiglione, A. Restivo, and M. Sciortino, “Nondeterministic moore automata and brzozowskis algorithm,” in *Proceedings of International Conference on Implementation and Application of Automata*, (Blois, France), pp. 88–99, Springer-Verlag, 2011.

[84] J. Daciuk, “Comparison of construction algorithms for minimal, acyclic, deterministic, finite-state automata from sets of strings,” in *Proceedings of International Conference on Implementation and Application of Automata*, (Tours, France), pp. 255–261, Springer-Verlag, 2002.

[85] C. Câmpeanu, N. Santean, and S. Yu, “Minimal cover-automata for finite languages,” *Theoretical Computer Science*, vol. 267, no. 1–2, pp. 3–16, 2001.