

Spring 2022

**Automata-Theoretic Approaches to Planning in Robotics:
Combinatorial Filter Minimization, Planning to Chronicle,
Temporal Logic Planning With Soft Specifications, and Sensor
Selection for Detecting Deviations From a Planned Itinerary**

Hazhar Rahmani

Follow this and additional works at: <https://scholarcommons.sc.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Rahmani, H.(2022). *Automata-Theoretic Approaches to Planning in Robotics: Combinatorial Filter Minimization, Planning to Chronicle, Temporal Logic Planning With Soft Specifications, and Sensor Selection for Detecting Deviations From a Planned Itinerary*. (Doctoral dissertation). Retrieved from <https://scholarcommons.sc.edu/etd/6805>

This Open Access Dissertation is brought to you by Scholar Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact digres@mailbox.sc.edu.

AUTOMATA-THEORETIC APPROACHES TO PLANNING IN ROBOTICS:
COMBINATORIAL FILTER MINIMIZATION, PLANNING TO CHRONICLE, TEMPORAL
LOGIC PLANNING WITH SOFT SPECIFICATIONS, AND SENSOR SELECTION FOR
DETECTING DEVIATIONS FROM A PLANNED ITINERARY

by

Hazhar Rahmani

Bachelor of Science
Iran University of Science and Technology 2008

Master of Science
Sharif University of Technology 2012

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in
Computer Science
College of Engineering and Computing
University of South Carolina
2022

Jason O’Kane, Major Professor

Marco Valtorta, Committee Member

Dylan Shell, Committee Member

Ioannis Rekleitis, Committee Member

Pooyan Jamshidi, Committee Member

Tracey L. Weldon, Interim Vice Provost and Dean of the Graduate School

© Copyright by Hazhar Rahmani, 2022
All Rights Reserved.

ACKNOWLEDGMENTS

I would like to take the opportunity of having this section in my dissertation to express my gratitude to many people who have assisted me. First I want to thank my advisor, Jason O’Kane, for all his forthcoming help and valuable advice during my PhD program. Completing this dissertation would not happen without his mentorship. My committee members, Marco Valtorta, Ioannis Rekleitis, Pooyan Jamshidi, and Dylan Shell offered useful guidance to improve my work and my presentation. Their insightful comments and questions will be very helpful for me in doing research. Special thanks to Dylan Shell for co-authoring several publications with us.

I want to thank all my colleagues and friends at the University of South Carolina, including Shervin Ghasemlou, Marios Xanthidis, Nick Stiffler, Trevor Olsen, Nare Karapetyan, Bharat Joshi, Alireza Salahirad, Alaleh Torkjazi, Alireza Nasiri, and Chrisogonas Odhiambo for all the constructive discussions we have had about our course and research works, and I am thankful for having had cooperation with several other researchers, Diptanil Chaudhuri and Yulin Zhang from Texas A&M University; Aaron Becker, Rhema Ike, Bernard Li, and Ekta Chaurasia from the University of Houston. I was lucky to make many good friends during these years I stayed in Columbia and from here I am sending my gratitude to all of them without listing their names because of the fear of missing someone’s name.

I am very thankful to my father and my siblings, who have always supported me and encouraged me to continue my studies. I am and will always be grateful and thankful to my mother, my greatest supporter, for all she did for me and my family. It was very close for her to see that this dissertation was completed and my PhD

program was finished.

The publications in this dissertation were graciously supported by the National Science Foundation under Grant Nos. 1453652, 1849249, 1849291, 1513203, 1526862, and 1659514.

Thanks to the University of South Carolina and all its staff and administrators for providing a suitable environment for teaching and research.

ABSTRACT

In this dissertation, we present a collection of new planning algorithms that enable robots to achieve complex goals, beyond simple point-to-point path planning, using automata-theoretic methods, and we consider the *filter minimization (FM) problem* and a variant of it, *filter partitioning minimization (FPM) problem*, which aims to minimize combinatorial filters, used for filtering and automata-theoretic planning in systems with discrete sensor data. We introduce a new variant of bisimulation, *compatibility*, and using this notion we identify several classes of filters for which FM or FPM is solvable in polynomial time, and propose several integer linear programming (ILP) formulations of FM and FPM. Then, we consider a problem, *planning to chronicle*, in which a robot is tasked with observing an uncertain time-extended process to produce a ‘chronicle’ of occurrent events that meets a given specification. This problem is useful in applications where we deploy robots to autonomously make structured videos or documentaries from events occurring in an unpredictable environment. Next, we study two variants of temporal logic planning in which the objective is to synthesize a trajectory that satisfies an optimal selection of soft constraints while nevertheless satisfying a hard constraint expressed in linear temporal logic (LTL). We also extend planning to chronicle with the idea of this problem. Then, we consider the problem of planning where to observe the behavior of an agent to ensure that the agent’s execution within the environment fits a pre-disclosed itinerary. This problem arises in a range of contexts including in validating safety claims about robot behavior, applications in security and surveillance, and for both the conception and the (physical) design and logistics of scientific experiments.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	iii
ABSTRACT	v
LIST OF FIGURES	x
CHAPTER 1 INTRODUCTION	1
1.1 Automata-theoretic approach for planning	2
1.2 Combinatorial filter minimization	4
1.3 Planning to chronicle	7
1.4 Temporal logic planning with soft constraints and specifications	9
1.5 Planning to chronicle with soft constraints	12
1.6 Sensor selection for detecting deviations from a planned itinerary	14
1.7 Dissertation organization	17
CHAPTER 2 RELATED WORK	18
2.1 Filter minimization problem	18
2.2 Planning to chronicle	20
2.3 Temporal logic planning with conflicting soft specifications	22
2.4 Planning to chronicle with soft constraints	24
2.5 Sensor selection for detecting deviations from a planned itinerary	26

CHAPTER 3	NOTIONS OF EQUIVALENCE FOR STATE SPACE MINIMIZATION OF COMBINATORIAL FILTERS	28
3.1	Combinatorial filters for robotic tasks	29
3.2	Definitions	35
3.3	Filter minimization via quotient operations	39
3.4	Maximum difference between the sizes of optimal solutions to FM and FPM	49
3.5	Bisimulation and filter reduction	52
3.6	Strong filter minimization	55
3.7	Simulation and filter reduction	59
3.8	The union of all compatibility relations and the mergeability relation	63
3.9	Special classes of filters	72
CHAPTER 4	INTEGER LINEAR PROGRAMMING FORMULATIONS OF FM AND FPM PROBLEMS	78
4.1	ILP formulations of the FPM problem	78
4.2	Experimental results for the FPM problem	90
4.3	ILP formulation of the FM problem	98
CHAPTER 5	PLANNING TO CHRONICLE	103
5.1	The problem	103
5.2	Algorithm description	108
5.3	Representation-invariance of expected time	117
5.4	Faster algorithms for special structures	122
5.5	Greedy algorithm	129

5.6	Construction of specification languages	130
5.7	Case studies	133
CHAPTER 6 OPTIMAL LTL PLANNING WITH LDL_f SOFT CONSTRAINTS		144
6.1	Preliminary definitions: Words, transition systems, LTL, Büchi automata, and LDL_f	144
6.2	Optimal preference planning	151
6.3	Algorithm description	152
6.4	Implementation and computed examples	161
CHAPTER 7 LTL PLANNING WITH SOFT LTL SPECIFICATIONS		164
7.1	Definitions and problem statement	164
7.2	Algorithm description	166
7.3	Case studies and experiments	174
CHAPTER 8 PLANNING TO CHRONICLE WITH SOFT CONSTRAINTS		183
8.1	Preliminaries and Problem Definition	183
8.2	Algorithm Description	188
8.3	Case study	195
CHAPTER 9 SENSOR SELECTION FOR DETECTING DEVIATIONS FROM A PLANNED ITINERARY		200
9.1	Definitions and problem statement	200
9.2	World graph-itinerary product automata	206
9.3	Hardness of MSSVI	208
9.4	MSSVI via integer linear programming	214

9.5 Case studies	217
CHAPTER 10 CONCLUSIONS AND FUTURE WORK	221
10.1 State space reduction of combinatorial filters	221
10.2 Planning to chronicle	224
10.3 Temporal logic planning with conflicting soft constraints	225
10.4 Sensor selection for detecting deviations from a planned itinerary . .	226
10.5 Open problems and future work	226
BIBLIOGRAPHY	230

LIST OF FIGURES

Figure 1.1	<p>a) A discrete sequence of states showing the behavior of the system b) A discrete structure modeling the set of all possible behaviors of the system c) An automaton that specifies the set of all desirable behaviors of the system d) A product automaton made from the system model and the finite automaton. The solution to the planning problem is found on the product automaton.</p>	3
Figure 1.2	<p>a) An environment with four landmarks 1-4, in which a mobile robot (not pictured) moves along a continuous path. The robot can sense at any time the cyclic order of landmarks as observed from its current position, but it does not know the positions of the landmarks and it does not have a compass nor odometers. The environment is (virtually) divided into 10 regions $a-j$, where at all points within a region, the same cyclic order of landmarks is perceptible. The task of the robot is to provably tell at any time whether it is in region f or not. b) A naïve filter the robot can use to accomplish its task. Notice that each state of the filter except state $a-j$ has also a loop labelled by the same cyclic order perceptible by the regions of that state, but we have avoided drawing those loops for simplicity. c) The smallest filter equivalent to the naïve filter.</p>	5
Figure 1.3	<p>a) An environment in which a mobile robot patrols the bathroom and the kitchen such that each must be visited infinitely often. b) A transition system model of this environment.</p>	10
Figure 1.4	<p>a) A wildlife reserve.</p>	14

Figure 1.5 Three quite distinct settings but which can be treated via the formulation described in this chapter. **a)** Wishing to ensure that claims made about a particular robot’s behavior will hold, an engineer chooses to instrument her laboratory’s testing environment (in this case, with a set of motion tracking cameras) so she can subject it to adequate scrutiny. **b)** An ornithologist would like to test a new hypothesis concerning the migration patterns of a species of bird, so assesses the cost of acquiring tracking capabilities with adequate power of discernment. **c)** Rather than purchasing sensors to maximize coverage, if benign activities can be precisely characterized, then negating that description permits surveillance which is sufficient to identify anomalous activities but with far fewer sensors. 16

Figure 3.1 **a)** A Venn diagram showing the connection between the bisimilarity relation \sim_F , the union of all compatibility relations \cup_F , and the mergeability relation \bowtie_F for a typical filter F with state space V . Note that it is possible that two or all these three relations coincide for some filters. **b)** A Venn diagram, over the space of state-state relations, showing the relationships between the set of all bisimulation relations (B), the set of all compatibility relations (C), and the set of all compatibility equivalence relations (M) for a typical filter F with state space V 29

Figure 3.2 The connections between problems we study in this chapter and notions we provide to solve those problems. The main problem we study is FM, but we also study two stronger variants of it, FPM and SFM. FM and FPM are NP-hard, while SFM is in P. To solve FM and FPM, we introduce the notion of compatibility, which is a relaxed variant of bisimulation. The diagram shows the connection between this notion and optimal solutions to FM and FPM. In addition, it shows the connection between the notion of bisimulation to those three problems, while establishing a connection between the notion of simulation and those problems. It also shows three special relations, the bisimilarity relation, the union of all compatibility relations, and the mergeability relation. 30

Figure 3.3	<p>a) A map of the third floor of ACES building of University of Texas at Austin [7, 8, 96]. The map is overlaid with the generalized Voronoi graph (GVG) of the environment in red.</p> <p>b) A simple environment and its GVG for illustration purposes.</p> <p>c) A naïvely-constructed combinatorial filter, which the robot can use to navigate through the environment in part b) to reach point 3, starting from full location uncertainty.</p>	32
Figure 3.4	<p>(left) A region divided by n sensor beams, in which two agents are located. When an agent crosses a beam, the system knows which beam sensor it was, but it does not know which agent it was, nor does it know the direction of crossing. The agents do not simultaneously cross beams. The task of the system is to determine at each time whether the agents are in the same region or not. (right) The smallest filter the system can use for accomplishing its task for an environment of 3 sensor beams.</p>	35
Figure 3.5	<p>a) A sample filter F_3. b) A minimal filter F_4 such that $F_3 \stackrel{L(F_3)}{\equiv} F_4$ and $L(F_3) = L(F_4)$. c) A minimal filter F_5 such that $F_3 \stackrel{L(F_3)}{\equiv} F_5$.</p>	39
Figure 3.6	<p>It illustrates the induction in the proof of Lemma 1, Lemma 2, and Lemma 3. The induction is on the lengths of observation sequences and it states that for each observation sequence s, if filter F reaches state v by tracing s from its initial state, then filter F/R, the quotient of F under a compatibility equivalence relation R, reaches state $[v]_R$ by tracing the same observation sequence from its own initial state.</p>	42
Figure 3.7	<p>An illustration of the proof of Lemma 5. Filter F_2 is a minimal filter for which $F_1 \stackrel{L(F_1)}{\equiv} F_2$ holds. The state space of F_2 corresponds to the quotient of the state space of F_1 under the equivalence relation R_f. The assumption for this relation is $(v, w) \in R_f$ but $(r, t) \notin R_f$.</p>	44
Figure 3.8	<p>a) A sample filter F. b) A minimal solution of FPM for filter F. c) The minimal solution of FM for filter F.</p>	46
Figure 3.9	<p>An illustration of the proof of Theorem 2. a) A sample filter F with n states. b) An optimal solution of FPM with input filter F. c) An optimal solution of FM with input filter F.</p>	50

Figure 3.10 **a)** The construction of filter F_n , mentioned in Theorem 3. The quotient of this filter under \sim_{F_n} does not reduce its size. **b)** Filters F_n^* and F_n^{**} , which are identical, are respectively the optimal solutions to FM and FPM with input F_n . State v_{n+2} in filter F_n and state $\{v_{n+2}\}$ in filters F_n^* and F_n^{**} have color 2; all other states in both filters have color 1. 54

Figure 3.11 Part of the proof of Theorem 4, which shows by contradiction that if $F \simeq F^*$ then $F \xrightarrow{L(F)} F^*$ and $L(F) = L(F^*)$. **a)** The contradiction assumption that $L(F) \neq L(F^*)$. Without loss of generality, we assume s to be an observation sequence in $L(F)$ but not in $L(F^*)$. Integer k is assumed to be the smallest integer such that $s_{1..k} \notin L(F^*)$ but $s_{1..k-1} \in L(F^*)$. Given that state $\delta^*(v_0, s_{1..k-1})$ has a transition for observation s_k but state $\delta'^*(v'_0, s_{1..k-1})$ does not have a transition for that observation, by the second condition of Definition 9, those two states are not bisimilar, i.e., $\delta^*(v_0, s_{1..k-1}) \not\sim_{(F,F^*)} \delta'^*(v'_0, s_{1..k-1})$, and this, via a series of implications ($\delta^*(v_0, s_{1..k-2}) \not\sim_{(F,F^*)} \delta'^*(v'_0, s_{1..k-2})$, $\delta^*(v_0, s_{1..k-3}) \not\sim_{(F,F^*)} \delta'^*(v'_0, s_{1..k-3})$, ...) implies that v_0 is not bisimilar with v'_0 , meaning that F and F^* are not bisimilar, which is a contradiction. A similar argument applies for when we assume that there is an observation sequence s such that $s \in L(F^*)$ but $s \notin L(F)$. **b)** The contradiction assumption that for an observation sequence $s \in L(F) \cap L(F^*)$, $c(\delta^*(v_0, s)) \neq c'(\delta'^*(v'_0, s))$. Given that those two states $\delta^*(v_0, s)$ and $\delta'^*(v'_0, s)$ do not have the same color, by the first condition of Definition 9, they are not bisimilar, i.e., $\delta^*(v_0, s) \not\sim_{(F,F^*)} \delta'^*(v'_0, s)$ and this, via a series of implications ($\delta^*(v_0, s_{1..|s|-1}) \not\sim_{(F,F^*)} \delta'^*(v'_0, s_{1..|s|-1})$, $\delta^*(v_0, s_{1..|s|-2}) \not\sim_{(F,F^*)} \delta'^*(v'_0, s_{1..|s|-2})$, ...) implies that $v_0 \not\sim_{(F,F^*)} v'_0$, meaning that F is not bisimilar to F^* , which is a contradiction. 57

Figure 3.12 **a)** A sample filter F_6 **b)** An example of a filter that simulates F_6 . For both filters, states in the left column have color 1, states in the middle column have color 2, and states in the right column have color 3. The similarity relation for (F_6, F_7) has been depicted by dashed lines. 61

Figure 3.13 A filter for which the union of all compatibility relations is not an equivalence relation. State v_0 has color 1, states in the middle column have color 2, state v_4 has color 3, and state v_5 has color 4. 66

Figure 3.14	<p>a) A filter for which the union of all compatibility relations does not coincide with its mergeability relation b) The graph of the union of all compatibility relations for filter F_8 c) The helper graph created for filter F_8 by Algorithm 2. d) The graph of the mergeability relation for filter F_8. In this example, states 2 and 3 are compatible, but they are not mergeable. States 0 and 1 have color white. States in the middle column have color blue. State 5 has color light-green, and state 6 has color red.</p>	68
Figure 4.1	<p>a) Filter F_1. States in the left column have color 1, states in the middle column have color 2, state 6 has color 3, and state 7 has color 4. b) The compatibility enforcement graph of filter F_1. c) A minimal filter equivalent to F_1.</p>	80
Figure 4.2	<p>a) A donut-shaped environment, in which an agent moves within n regions separated by n beam sensors. When the robot crosses a beam, the system knows which sensor it was, but it does not know the direction of that crossing. The task of the system is to determine at any time whether the agent is definitely in region 1 or not. b) A naïve filter which is used to accomplish the task for an instance problem with $n = 5$ regions. c) The smallest filter the system can use to accomplish its task.</p>	91
Figure 4.3	<p>The performance of the algorithms in computing optimal filters for: (top) naïve filters of single-agent-donut instances, (bottom) naïve filters of two-agent-donut instances. Notice that the vertical axis in both charts is in logarithmic scale. For each filter size, the experiment was performed for only one filter of that size.</p>	94
Figure 4.4	<p>The performance of the algorithms in computing optimal filters for random filters for which the number of compatible pairs was linear to the number of states of the filter. For each of the filter sizes, the plotted time is the average time to minimize the filter over 10 randomly generated filters of that size. If an algorithm could not find an optimal solution for a filter on a given size in half an hour, then it was considered as a failure, and as a result, the average time is not plotted. Note again that the vertical axis of the chart is in logarithmic scale.</p>	95

Figure 4.5	The trade-off chart of running time vs quality of solution of reduced filters found by the algorithms on (top) the naïve filter of two-agents-donut problem with 13 regions (bottom) the naïve filter of two-agents-donut problem with 15 regions. The former filter has 92 states, and the later has 121 states.	97
Figure 5.1	a) An event model \mathcal{M} with its observation model \mathcal{B} . b) A DFA \mathcal{D} , specifying event sequences that contain at least one event. c) The Goal POMDP $\mathcal{P}_{(\mathcal{M},\mathcal{B};\mathcal{D})}$, constructed by Definition 18. (Self-loop transitions of the goal states have been omitted to try reduce visual clutter.)	111
Figure 5.2	a) A DFA specifying all stories that contain at least an event over the event set $\{e_1, e_2\}$. b) A sample event model, in which event e_1 happens at s_1 and event e_2 happens at s_2 . c) The Goal MDP obtained from the product of the DFA and the event model in parts (a) and (b). component.	115
Figure 5.3	a) A DFA specifying all stories that contain at least an event over the event set $\{e_1, e_2\}$. b) A sample event model, in which event e_1 happens at s_1 and s_2 while event e_2 happens at s_1 . c) The Goal MDP obtained from the product of the DFA and the event model in parts (a) and (b). component.	116
Figure 5.4	a) A loop-omitted acyclic DFA. b) A sample event model. c) The Goal MDP obtained from the product of the DFA and the event model in parts (a) and (b). d) It shows the strongly connected components of graph underlying the MDP in part c. Each blue circle is a strongly connected component. The self-loops and the edges between states of different SCCs have been omitted to reduce visual clutter.	125
Figure 5.5	A sample of an MDP that is a directed acyclic graph when all the self-loops are removed.	128

Figure 5.6 **a)** Districts of Oulu that William is touring. **b)** An event model that describes how a tourist visits those districts. **c)** A DFA specifying that the captured story must contain events k and h and at least one of c or t . **d)** The optimal policy for the RTM/FOM problem for Oulu. Subfigures **e-j** consist of a histogram showing, for 5,000 simulations, the distribution of the number of hours (steps) William (system) circulated (ran) until the robot recorded a story specified by the DFA, and a pie chart showing the distribution of recorded sequences in these simulations. Pairs **e)** and **f)** are for the RTM/FOM problem, **g)** and **h)** for the RTM problem, and **i)** and **j)** are for the RTM/FHM problem. In the left column the robot uses the general algorithm, while in the right column it uses the greedy approach. Moving down the column, the average number of steps to record a story increases as the robot's perception of the world's state diminishes. While the distribution of the recorded sequences by the general algorithm differs from the distribution of the recorded sequences by the greedy algorithm, for each of the greedy algorithm and the general algorithm, the distributions of the recorded sequences under different levels observability were similar. . 134

Figure 5.7 **a)** The event model for the behavior of a person attending a wedding reception, which has six states: I_i , the state of arriving; E_i , the state of being entertaining; C_i , for consuming coffee; B_i , for drinking other beverages; D_i , for dancing; and S_i , for smoking. Each histogram shows the average number of steps to record a desired story for 5,000 simulations of the wedding reception scenario. The left column, parts **b**, **d**, **f**, **h**, and **j**, is for the cases where the robot uses the general algorithm, while the right column, parts **c**, **e**, **g**, **i**, and **k** are for the cases where the robot uses the greedy algorithm. Parts **d** and **e** are for when there is a smoke detector, providing the observation of whether someone is smoking, and a microphone, capable of detecting that someone is dancing or being entertained. For parts **f** and **h**, there is only a smoke detector, while for parts **h** and **i** there is only a microphone. It seems, at first, that the RTM problem with a smoke detector might be incomparable with RTM using a microphone, but the single useful observation in the former guarantees that at least one guest is in the state of smoking, while the single useful observation in the later case guarantees that at least one guest is either in state of dancing or in state of being entertained, which is less informative. This experiment also shows that increasing observability will decrease the time to capture a desired story. Furthermore, it shows that although the general algorithm often outperformed the greedy algorithm in terms of average number of steps, here the greedy algorithm gives a reasonable approximation to the optimal solutions. . . . 137

Figure 5.8 **a)** An example of a race source, which we have divided into 8 sections s_0 through s_7 form an event model for the race. **b)** The event model for a single runner $i \in \{1, 2\}$, including events r_i , runner i is running; h_i , runner i is crossing the race flag at the middle of the race field; f_i , runner i is crossing the finish line. The event model for the two runners John and James is made by the product of the event models for each of them. Two more events, p_{12} and p_{21} , are introduced to the joint event model. Event p_{12} denotes that John is overtaking James, while p_{21} means that James is overtaking John. Each of these two events happens with probability 0.5 at each state that represents the runners are in the same section. **c)** The DFA specifying the set of all desirable videos for the race example. **d)** A diagram showing the number of states of the MDP created by the product of the event model and the DFA in this race example. **e)** A diagram showing the number of edges of the graph underlying the MDP. Parts **d** and **e** together show how the size of the MDP increases as the size of the problem increases. They show as the problem's size doubles, the MDP's size approximately quadruplicates. **f)** A diagram showing the computation time for classical value iteration, which we use in our general algorithm, the computation time of the topological value iteration algorithm, which use to solve our problem where the DFA is loop-omitted acyclic, and the computation time of the topological value iteration, which we use to solve our problem when Goal MDP is a loop-omitted DAG. For this experiment, the topological single iteration was on average 6.5 faster than the topological value iteration and also the latter on average was 6 times faster than the general, classical value iteration algorithm. **g)** A diagram showing, for topological single value iteration, the breakdown of its computation time into the three phases of the algorithm: decomposing the MDP into its SCCs, finding a topological ordering of the SCCs, and performing the value iteration. **h)** A diagram showing, for the topological single value iteration algorithm, how much each of the two phases of the algorithm, namely, finding a topological ordering of the states of the MDP, and computing an optimal action for each state via solving the single variable equations, contribute to the computation time of the algorithm. The results for each of the section sizes 30-120 in the diagrams in parts **(f)**, **(g)**, and **(h)** are the average computation times across 10 trials. 143

Figure 6.1	A Büchi automaton A_φ for LTL formula $\varphi = \Box\Diamond k \wedge \Box\Diamond h$. For the transition system in Figure 1.3, both the formula and the corresponding automaton shown here specify trajectories that infinitely often visit both the kitchen and the bathroom.	147
Figure 6.2	(a) A DFA \mathcal{D}_1 accepting traces satisfying LDL _f formula $\psi_1 = [true^*]\neg d_1$ (b) A DFA \mathcal{D}_2 accepting all the traces satisfying LDL _f formula $\psi_2 = [true^*]\neg d_2$ (c) Filter \mathcal{F} constructed by Lemma 11 for \mathcal{D}_1 and \mathcal{D}_2	154
Figure 6.3	An environment consisting of a corridor, an office, a kitchen, two rooms, a conference hall, and a security room.	161
Figure 6.4	Formulation and results for two instances of the OPP problem.	161
Figure 6.5	A race car traveling around a circular track.	163
Figure 7.1	Showing our algorithm for finding an optimal lasso $r_{\mathcal{P}} = r_1(r_2)^\omega$ over product automaton \mathcal{P} . Each C_i is a strongly connected component of the graph underlying \mathcal{P} . Component C_5 contains the suffix of an optimal lasso. Set $F_{C_5, \mathcal{P}}$ contains those state in C_5 that are accepting for \mathcal{P} , i.e., $F_{C_5, \mathcal{P}} = F_{\mathcal{P}} \cap C_5$. For each $i \in \{1, 3, 4, 7\}$, set F_{i, C_5} are those states in C_5 that are accepting for Büchi automaton \mathcal{B}_i —the one who represents preference ψ_i	172
Figure 7.2	A hospital, in which a delivery robot is tasked to deliver first aid items to emergency and primary care departments, deliver medicine to pharmacy, and visit the maintenance section. The robot’s task is expressed by LTL formula $\varphi = \Box\Diamond(p \wedge f) \wedge \Box\Diamond(e \wedge f) \wedge \Box\Diamond(h \wedge m) \wedge \Box\Diamond n$	175
Figure 7.3	a) A retirement home in which a social enrichment robot visits each of the common rooms to perform juggling and to make animal balloons. Its primary mission is expressed by the LTL formula $\varphi = \Box\Diamond r_1 \wedge \Box\Diamond r_2 \wedge \Box\Diamond t$. b) A transition system that models the robot’s state within this environment.	177
Figure 7.4	Results of our experiment comparing our algorithm with the brute-force algorithm. The average times are in seconds.	180
Figure 7.5	The distribution of approximation ratios of the lasso sizes generated by our greedy algorithm for 100 generated random graphs with 100 states.	181

Figure 8.1	<p>a) A transition system showing the locations of interest from the University of South Carolina. The transitions of this structures is based on the motion capabilities of the robot, that is, a transition from a location w_1 to location w_2 in this transition system exists only if the robot can move from w_1 to w_2 in 10 minutes. b) The event-location model that represents the occurrences of the events of interest in this environment. Those events include exercising, e; working, r; eating food, f; and drinking coffee c.</p>	197
Figure 9.1	<p>a) An example of an environment, which shows the floor map of a department. This environment is guarded by beam sensors b_1, b_2, and b_3, and by occupancy sensors o_1, o_2, o_3, and o_4. b) A multigraph representing the environment. Each edge e is labeled by a world-observation, a set of events received when an agent moves from the region represented by the source vertex of e to the region represented by the target vertex of e.</p>	201
Figure 9.2	<p>Two cases in the construction of the relation R in the proof of Lemma 23. In each case, R is expanded, given that states q and p are already related by R. a) In this case, pair $\delta_{\mathcal{P}}(q, e)$ and $\delta_{\mathcal{P}}(p, d)$ are added to R. b) In this case, pair $\delta_{\mathcal{P}}(q, e)$ and p are added to R.</p>	210
Figure 9.3	<p>a) An instance of the set cover problem. b) The instance of the MSSVI problem for this set cover instance. c) A physical environment representing the MSSVI instance.</p>	213
Figure 9.4	<p>a) An aggregate of tracking data, reproduced from [138], showing the journeys across parts of North Africa and Central Asia made by eagles over a period of one year. Fig. b) The decomposition of the map into subregions. Subregions who fall substantially within a single country have been assigned a color representing the potential of purchasing data roaming service from that country (there are 10 colors, representing sensors set S). The optimal sensor selection to certify the hypothesis of circling the Caspian Sea has six sensors, consisting of the sets of hexagons of the following colors: $\color{green}\hexagon$, $\color{red}\hexagon$, $\color{blue}\hexagon$, $\color{purple}\hexagon$, $\color{orange}\hexagon$, and $\color{cyan}\hexagon$.</p>	220

CHAPTER 1

INTRODUCTION

Robots have taken part in many aspects of our lives from education and personal assistance to industry and space exploration, and with each passing year they are becoming more advanced and their role in human life is broadening. As such roboticists face new planning problems, beyond the classical motion planning problem. Motion planning is perhaps the most known planning problem in robotics and, in fact, many may still think of planning in robotics as only motion planning. The aim of this problem is to choose a sequence of suitable actions that moves a robot from a source location to a goal location without hitting the obstacles such that an objective function such as the path length is optimized.

In this dissertation we present several such problems. The first problem studies state space minimization of discrete structures used for planning and filtering in robotics. This problem is useful for robots with limited physical and computational resources and it may help to convey salient information about robotic tasks those structures are used for. The second problem addresses using autonomous robots to make structured videos from events occurring in unpredictable environments. This problem is also important because it helps us to use autonomous robots instead of humans for making a video or a documentary that might be laborious for a human to do or it must be taken in places that might be dangerous for humans to go. The third problem studies temporal logic planning given both hard specifications of the mission and soft constraints or specifications to accomplish the mission. Temporal logic planning is important because it helps humans to use a user-friendly, high-level

language to specify complex tasks for a robot while ensuring that the specification language is unambiguous and algorithmically manipulable. Planning with soft specifications is helpful in situations where not all of the specifications can be satisfied as a whole. In such cases, one needs to decompose the specifications into several parts and impose a priority over the parts. It is also helpful in situations where a robot is asked by several recipients to do a task within an execution or where there are several ways to do a task and the user has preferences on how to accomplish the task. The fourth problem applies the ideas of temporal logic planning with soft constraints on the problem of using robots to record structured videos. The fifth problem considers the problem of optimal sensor selection for detecting whether an agent's movement in the environment has deviated from a promised itinerary or not. This problem is useful in security and surveillance applications, and can be used to check a hypothesis about a system behavior, and thus, it is important. We show how to solve all of these problems using a general form of automata-theoretic approach. This approach provides a unified technique to tackle new planning problems.

In this chapter, we first introduce the general approach of automata-theoretic for planning, and then briefly introduce the problems we consider in this dissertation.

1.1 AUTOMATA-THEORETIC APPROACH FOR PLANNING

Figure 1.1 shows the general theme of an automata-theoretic approach to planning. In our problems we consider robotic systems with discrete models. In those kinds of systems, the behavior of the system is assumed to be a discrete sequence of states. Depending on application, that sequence could be finite or infinite. The set of all possible behaviors of the system is modeled using a discrete structure, such as a *transition system*, a *hidden Markov model*, and a *multi-graph*, and the set of all desirable behaviors for planning is specified using an automaton, such as a *deterministic finite*

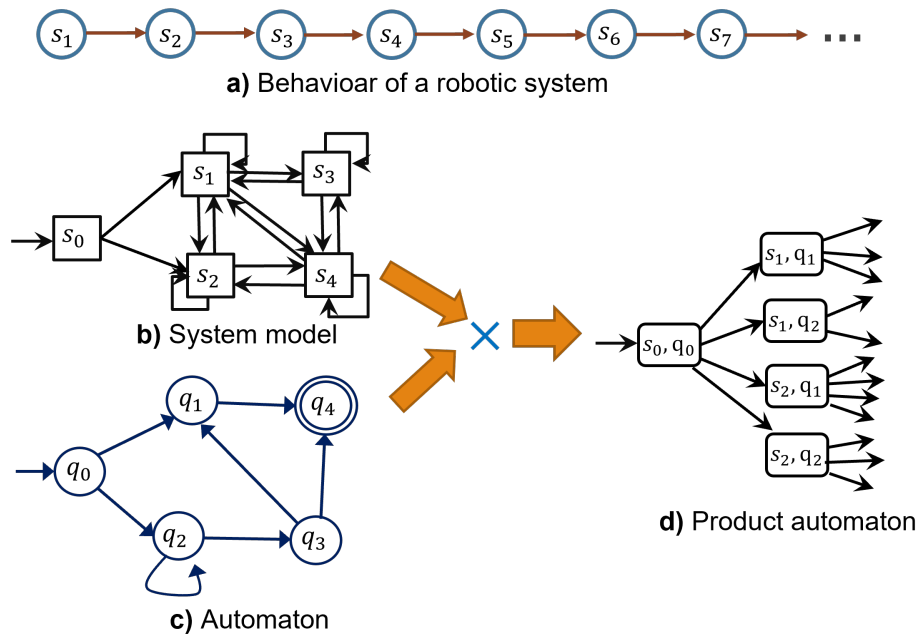


Figure 1.1: **a)** A discrete sequence of states showing the behavior of the system **b)** A discrete structure modeling the set of all possible behaviors of the system **c)** An automaton that specifies the set of all desirable behaviors of the system **d)** A product automaton made from the system model and the finite automaton. The solution to the planning problem is found on the product automaton.

automaton (DFA), a Büchi automaton, and a deterministic Rabin automaton. The solution to the planning problem is then found on a product automaton made from the system model and the automaton.

Combinatorial filters are discrete structures that can be used for any part of an automata-theoretic approach for planning. They can be used to model the system, to specify the set of all desirable behaviors, or they can be a plan that are obtained as a solution to a planning problem. All the other problems we consider in this dissertation are solved using an instance of the general automata-theoretic approach to planning. We first consider the problem of state space reduction of combinatorial filters.

1.2 COMBINATORIAL FILTER MINIMIZATION

The ability of robots to perceive and maintain salient information about their environments—and their own place within those environments—is rightly considered to be an essential ingredient for many forms of autonomy. As a result, a central thread in modern robotics research is an effort to design and understand filtering and estimation methods.

A number of robotics researchers in recent years have considered these kinds of problems from the perspective of combinatorial filters, which model those processes as discrete transition systems. Combinatorial filters, first proposed by LaValle [77, 78], are a general class of models for reasoning about systems that process discrete (rather than continuous) sensor data. Variations on the combinatorial filtering approach have, under various names, been used for a wide spectrum of tasks including localization [2], navigation [83, 145, 149], exploration [74], manipulation [73], mapping [139], target tracking [10, 38, 169], and story validation [168]. The essence of the approach can be understood as describing filtering processes as directed graphs, in which the vertices represent the distinct states of the information maintained by the filter, the directed edges are labeled with sensor readings that induce transitions between states, and the vertex labels indicate the output of the filtering process at each state. The combinatorial filters considered in this dissertation are closely related to well-known probabilistic filtering methods such as recursive Bayesian estimation [58, 88] and Kalman filtering [63]. Each of these types of filters are used to estimate a system’s internal state over time using the most recent data (observations) received from the sensors. Note in particular that probabilistic filters, when implemented with finite precision, can in principle be expressed as combinatorial filters, in which each state of the filter corresponds to a tuple of concrete values of the variables being estimated and each transition encodes the updates made to that internal state in response to new sensor data. In addition, combinatorial filters are also suitable for other forms of

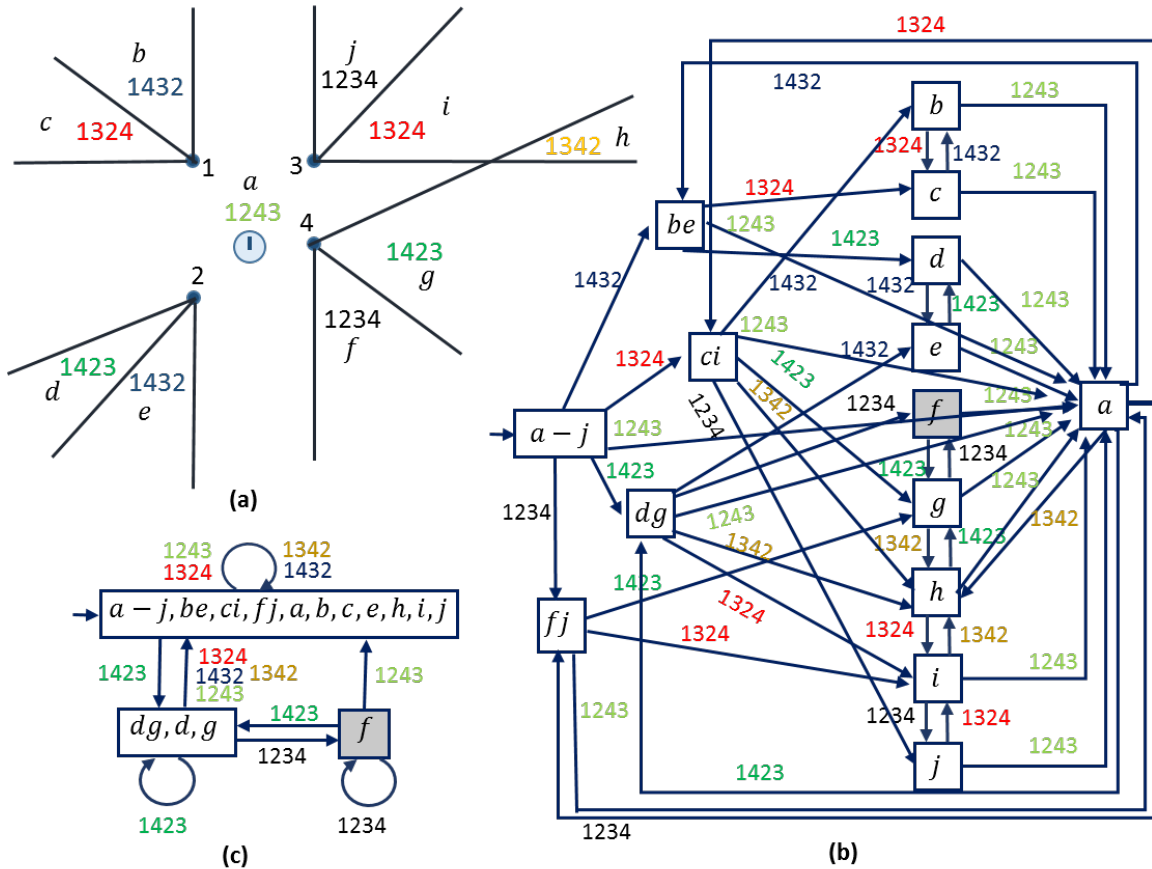


Figure 1.2: **a)** An environment with four landmarks 1-4, in which a mobile robot (not pictured) moves along a continuous path. The robot can sense at any time the cyclic order of landmarks as observed from its current position, but it does not know the positions of the landmarks and it does not have a compass nor odometers. The environment is (virtually) divided into 10 regions $a-j$, where at all points within a region, the same cyclic order of landmarks is perceptible. The task of the robot is to provably tell at any time whether it is in region f or not. **b)** A naïve filter the robot can use to accomplish its task. Notice that each state of the filter except state $a-j$ has also a loop labelled by the same cyclic order perceptible by the regions of that state, but we have avoided drawing those loops for simplicity. **c)** The smallest filter equivalent to the naïve filter.

filtering that rely upon combinatorial, rather than probabilistic reasoning. For more details about Bayesian filtering and the Kalman filter, see [23].

Central to much of the research on combinatorial filters is the notion of *minimality*. Specifically, questions are asked about the minimal state information required to be maintained in the filter, in order to accurately express the desired behavior. This

concern naturally leads to questions about *equivalences* between states in the filter: If we can construct a filter for a given task in which no pair of distinct states is equivalent, in the (thus far informal) sense that the distinction between them is irrelevant to the filter’s outward behavior, then the states utilized by that filter may illuminate the information requirements of the task. Thus, for a given combinatorial filter, we are interested in the *filter minimization problem (FM)* of finding the smallest equivalent filter. This problem was addressed by O’Kane and Shell [98, 100] who proved that it is NP-hard. They proposed a heuristic algorithm to solve FM, which forms the reduced filter by merging pairs of states, who are identified to be mergeable using iteratively forming and coloring conflict graphs. This algorithm can also be used for a variant of FM, called the *filter partitioning minimization problem (FPM)*, which requires the reduced filter to partition the state space of the original filter.

Overview of results

This dissertation studies FM and FPM by considering several distinct state-equivalence relations in the context of those problems. Specifically, we show that the well-known state-equivalence concept of *bisimulation* —which is used widely for the minimization of other discrete transition structures [127]— does not, as intuition might suggest, correctly capture the notion of equivalence between states necessary to minimize a filter optimally. Nonetheless, using bisimulation we identify, in Section 3.9, classes of filters for which FPM or FM is solvable in polynomial time. The notion of bisimulation also provided inspiration for the correct notions of *compatibility* and *mergeability* for FM and FPM, which we introduce in this dissertation.

For the naïve filter in Figure 1.2, which has 15 states, applying the technique of bisimulation minimization does not reduce the state space of the filter at all, while the minimal equivalent filter has only 3 states. In Theorem 3, we describe a class of combinatorial filters on which bisimulation minimization does not reduce the state

space of the original filter at all, whereas the optimally reduced filter has only two states.

We also establish a connection between FM and the notion of *simulation*. We show that the FM problem is equivalent to the problem of finding for a given filter, a state-minimal filter that simulates the given filter. We also introduce two new relations over states called *compatibility* and *mergeability* and establish their relevance to the FM and FPM. By analyzing where those two relations become an equivalence relation, we identify several classes of filters for which FM or FPM is solvable in a time polynomial to the size of the input filter. We propose three integer linear programming (ILP) formulations of FPM and show, via experiments, that those ILP formulations outperform the algorithm of O’Kane and Shell. We also consider an integer linear programming formulation of FM.

1.3 PLANNING TO CHRONICLE

Imagine that the next wedding you attend features an autonomous robot camera operator capable of roaming about the event and shooting film footage of the reception party. Afterwards, the robot automatically assembles the raw clips into a series of videos, each customized to its recipient, that are distributed to the happy couple and guests. These videos might help preserve special memories worth cherishing, they might summarize aspects of the evening and show the event from particular points of view, and they might, most tenderly, foretell the future wedding of a pair who just met for the first time that evening. Then again, they might serve as evidence in a court case brought against wedding crashers documenting the fracas that ensues. As different videos highlight different aspects of the evening, no single compilation of clips will serve all these purposes effectively. Given the myriad of stories to tell—each being a fairly complex, temporally extended series of events, some of which may unfold

on the day, others which do not on this particular night—how does the robot best strategize its movements to collect this raw footage? For this kind of problem we are interested in robotic planning in which the goals are expressed as time-extended sequences of discrete events whose occurrence the robot cannot causally influence.

Many applications can be cast as the problem of producing a finite-length sensor-based recording of the evolution of some process. As the wedding example emphasizes, one might be interested in recordings that meet rich specifications of the event sequences that are of interest. When the evolution of the event-generating process is uncertain/non-deterministic and sensing is local (necessitating its active direction), then one encounters an instance from this class of problem. The broad class encompasses many monitoring and surveillance scenarios. An important characteristic of such settings is that the robot has influence over what it captures via its sensors, but cannot control the process of interest.

Overview of results

Our incursion into this class of problem involves two lines of attack. The first is a wide-embracing formulation in which we pose a general stochastic model, including aspects of hidden/latent state, simultaneity of event occurrence, and various assumptions on the form of observability. Secondly, we specify the sequences of interest via a deterministic finite automaton (DFA), and we define several language mutators, which permit composition and refinement of specification DFAs, allowing for rich descriptions of desirable event sequences. The two parts are brought together via our approach to planning: we show how to compute an optimal policy (to satisfy the specifications as quickly as possible) via a form of product automaton.

Our solution to this problem fits the general automata-theoretic approach. It models the system and the occurrences of events using a variant of hidden Markov model, and specifies the set of all desirable event sequences using a DFA. The product

automaton made from the hidden Markov model and the DFA is treated as a partially observable Markov decision process (POMDP) or a Markov decision process (MDP), depending on observability of the current state of the hidden Markov chain.

Beyond the pragmatics of planning, a theoretical contribution of dissertation is to prove a result on representation independence of the specifications. That is, though multiple distinct DFAs may express the same regular language and despite the DFA being involved directly in constructing the product automaton used to solve the planning problem, we show that it is merely the language expressed that affects the resulting optimal solution. Returning to mutators that transform DFAs, enabling easy expression of sophisticated requirements, we distinguish when mutators preserve representational independence too.

1.4 TEMPORAL LOGIC PLANNING WITH SOFT CONSTRAINTS AND SPECIFICATIONS

As techniques to solve traditional problems in motion and path planning —those concerned with constructing a finite trajectory that starts from an initial state and ends in a goal state while avoiding obstacles [77]— have matured, the community’s focus has expanded to include richer classes of problems, including those grounded in temporal logic. Temporal logic planning problems [9, 40, 136] extend the classical conception of motion planning in several ways, most notably by generalizing the constraint of merely avoiding obstacles to any user-specified temporal or spatial constraints, and by allowing the trajectory to be infinite. Progress on these problems has been achievable thanks, first, to the maturity of temporal logics, primarily Linear Temporal Logic (LTL), which allow the user to use simple, high-level logical formulas to express complex missions; and second, to the maturity of model checking techniques that enable the robot to automatically make plans for those missions.

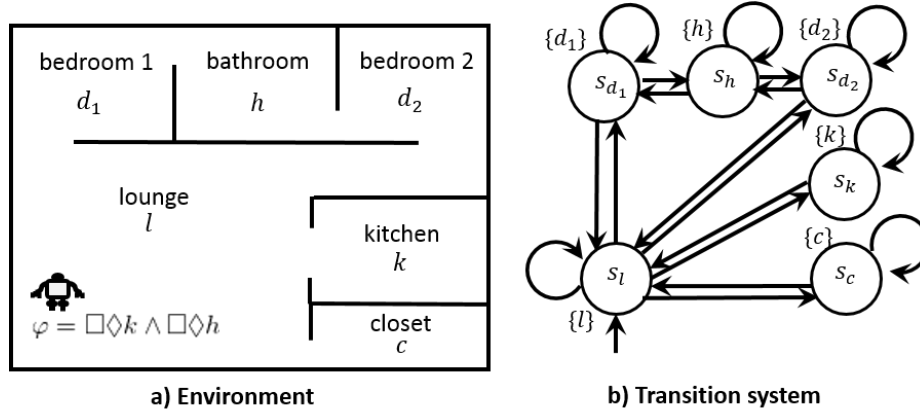


Figure 1.3: **a)** An environment in which a mobile robot patrols the bathroom and the kitchen such that each must be visited infinitely often. **b)** A transition system model of this environment.

Temporal logics, including linear temporal logic (LTL) specifically, offer high-level, user-friendly languages for specifying complex missions and tasks. In fact, as simple and intuitive as temporal logic is for humans to understand, it is also precise and rigorous for robot algorithms to manipulate.

In this dissertation, we consider a temporal logic planning problem in which a robot is tasked to accomplish a mission specified by an LTL formula while optimally satisfying a set of additional, possibly conflicting, logical formulas. These extra constraints could be user preferences, safety rules, soft goals, or other constraints.

We illustrate the idea using an example in which the soft logical formulas are used for specifying user preferences. Consider the environment in Figure 1.3a, in which a mobile robot in a home is tasked to patrol the kitchen (k) and bathroom (h) to detect water leakage. This environment is modeled as a transition system, as shown in Figure 1.3. The patrolling mission for this robot can be characterized as “visit the bathroom and the kitchen, each one infinitely often,” expressed in the LTL formula $\square\Diamond k \wedge \square\Diamond h$.

Such a mission can, in general, be satisfied by many plans. But the user may prefer, if possible, some additional properties to hold on the robot’s plan. Therefore,

he or she may, in advance, guide the planner by specifying their preferences. The planner must generate a plan that satisfies these preferences, if it is possible to do so. For example, the user of the robot in Fig. 1.3 might have preferences such as:

- (a) “The robot should not go into room d_1 .”
- (b) “The robot should not go into room d_2 .”
- (c) “If the robot enters a bedroom d_1 or d_2 , it should first put in the slippers from the closet c .”

In general, it will not be possible for the robot to satisfy all of the specified preferences at once. We assume, therefore, that the user has specified the preferences in order of importance, from most important to least important.

We are interested in planning algorithms that allow the robot to complete its mission while satisfying as many of the preferences as possible, subject to the stipulation that a higher priority preference should not be sacrificed in order to satisfy lower-priority preferences. In our example, an optimal trajectory would, starting from l , travel to c , then alternate between k and h , traveling via d_2 . This infinite trajectory would satisfy preferences (a) and (c), but not (b). Note that no trajectory can satisfy all three of the given preferences.

Overview of results

In this dissertation, we present two general formulations of this type of problem, in both of which the overall mission is specified as an LTL formula but in one of them the soft specifications are expressed as an ordered list of formulas in linear dynamic logic for finite traces (LDL_f), while in the other one, the soft specifications are expressed as an ordered list of LTL formulas. Because by LDL_f we are able to specify only finite traces but not infinite traces, we use LDL_f to impose constraints on the finite

prefixes of the infinite trace that satisfies the LTL formula. In this case, the LDL_f can be used for specifying safety rules or user preferences on how the mission must be completed.

A limitation of the first problem is that the LDL_f soft constraints cannot express soft goals that are satisfied only by infinite (rather than finite) trajectories. As an example, a task that requires the robot to infinitely often check the first bedroom cannot be expressed by an LDL_f formula.

The second problem also can be used for situations where an LTL task cannot be accomplished by a robot because the specifications are conflicting or merely because of physical limitations. The idea is to decompose the LTL task into smaller tasks, and let the user to prioritize the smaller tasks so that the robot follows a trajectory that satisfies an optimal selection of those tasks.

The difference in the language used to express the soft constraints not only improves the expressivity of the approach, but it leads to significant (and new, compared to the LDL_f case) algorithmic challenges.

Again we use an automata-theoretic approach to solve these two variants of temporal logic planning. In both, the system is modeled using a transition system. LTL formulas are converted into Büchi automata while LDL_f formulas are converted into DFAs. In both cases, the product automaton made from the automata and the transition system is treated a Büchi automaton, on which we compute a *lasso* for a solution of our problem.

1.5 PLANNING TO CHRONICLE WITH SOFT CONSTRAINTS

One limitation of our initial formulation of the planning to chronicle problem is that the user specifies only the temporal aspect of desirable event sequences and it does not specify spatial constraints on how an event sequence must be captured. As an

example, for an event that can take place in several locations in an environment, the user might want the robot to record that event only in some of those locations. Also, the user might want the robot to avoid visiting certain locations in the environment or impose more complex, temporal and spatial constraints on the movement of the robot in the environment, and some of those constraints might be conflicting. The aim of this problem is to allow the user to specify not only the desirable event sequences but also the robot’s motion in the environment and that there might be preferences or conflicting soft constraints on how to record a desirable event sequence.

Overview of results

To illustrate, consider a wildlife reserve shown in Figure 1.4. This wildlife reserve has four main locations, a veld, v ; a command post, c ; a jungle, j ; and a riverside, r . In this wildlife reserve, a pride of lions, a troop of baboons, a flamboyance of flamingos, and a gang of buffaloes live. Events of interest in this example are lions-hunting-a-buffalo, l_o ; lions-eating-an-impala, l_i ; lions-eating-a-baboon, l_b ; baboons-hunting-an-impala, b_i ; an-alligator-eating-a-baboon, a_b ; an-alligator-eating-an-impala, a_i ; and two-flamingos-mating, f_m . The story along with the hard spatial-temporal constraints are specified using an LDL_f formula. An example is a story in which first either an alligator eating an impala or a lion eating an impala and then a lion is eating a buffalo in the veld and the story also contains either two flamingos mating or an alligator eating a baboon and the the robot does not stay in the jungle for more than two hours. This story is specified using the LDL_f formula $\varphi_1 = \langle true^* \rangle ((a_i | l_i) \wedge \langle true^* \rangle (l_b \wedge v)) \wedge \langle true^* \rangle (f_m \wedge a_b) \wedge [true^*] ((j \wedge \langle true \rangle j) \rightarrow \neg \langle true; true \rangle j)$. This formula specifies not only the story but also how the story is captured. It also imposes a constraint on the movement of the robot.

Apart from the LDL_f formula, a list of ordered soft constraints is given. Some of those soft constraints might be conflicting. This extension of our planning to chronicle

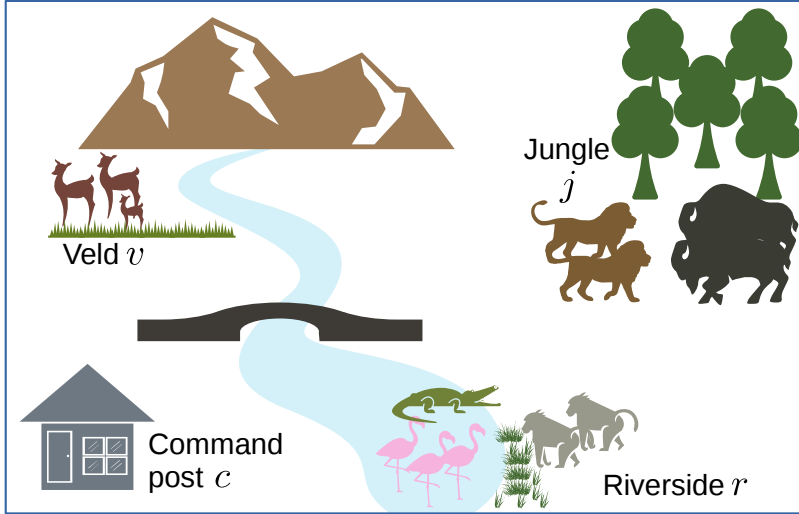


Figure 1.4: **a)** A wildlife reserve.

problem has two objectives. One objective is to minimize the expected number of steps to record a desired story and the other objective is to minimize the expected cost of violation of the soft constraints, that is, the robot must synthesize a policy that guarantees to violate the soft constraints with lower priorities if it has to violate some of the constraints.

Solving this problem needs to compute a set of Pareto optimal policies rather than a single optimal policy, and computing that set is a challenge problem considering that even for very small instances of the problem it might not be feasible to compute the set of all those policies.

1.6 SENSOR SELECTION FOR DETECTING DEVIATIONS FROM A PLANNED ITINERARY

Suppose an agent asserts that it will move through an environment in some way. When the agent executes its motion, how does one verify the claim? The problem arises in a range of contexts including validating safety claims about robot behavior, applications in security and surveillance, and for both the conception and the (phys-

ical) design and logistics of scientific experiments. Given a set of feasible sensors to select from, we ask how to choose sensors optimally in order to ensure that the agent’s execution does indeed fit its pre-disclosed itinerary.

Determining how agents within an environment are behaving and understanding that behavior, for instance by recognizing whether it fits some pattern, is crucial to the problem of situational awareness, which broadly encompasses agent detection and tracking, activity modeling, general sense-making, and semantically-informed surveillance. It forms an important capability for intelligent systems and is a topic of interest for the robotics community for at least three reasons. First, as information consumers: such information could enhance the ability of a robot to act within context, improving the responsiveness and appropriateness of robot actions to other events. Secondly, as information producers: we may wish to task a robot with providing raw sensor information to enable coverage and facilitate such situational awareness. The third reason is one shared of technical interest: the methods and algorithms that enable such situational awareness have substantial overlap with those used for estimation on-board robots and, historically, cross-pollination between the two has been fruitful. This problem fits within the vein of work concerned with guarding an environment [99], though is closer to the minimalist spirit of [144] both in terms of sensors—we adopt a simple model well suited to information-impooverished sensors such as occupancy and beam sensors—but also in the use of combinatorial filters—for fusing sequential observations and estimating state.

Overview of results

This problem was inspired by Yu and LaValle [167], who consider the question of validating a story: given a polygonal environment, a claimant provides a sequence of locations which they assert to have visited and the system is tasked with determining whether a given sequence of sensor readings is consistent with that claim.

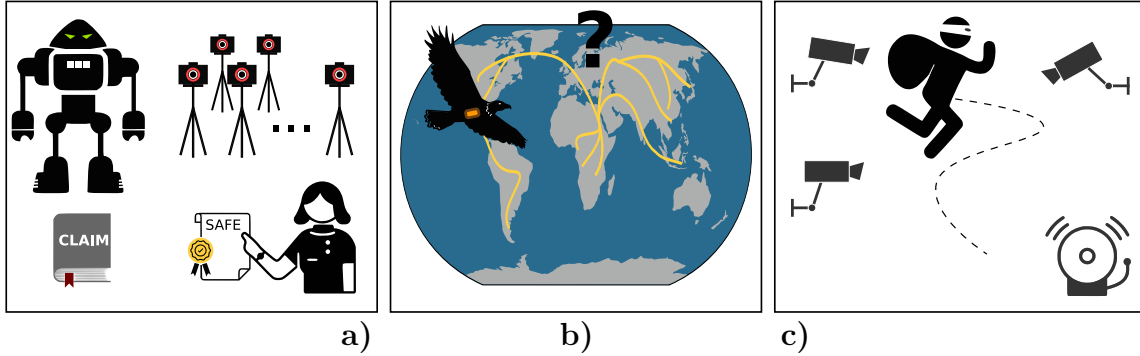


Figure 1.5: Three quite distinct settings but which can be treated via the formulation described in this chapter. **a)** Wishing to ensure that claims made about a particular robot’s behavior will hold, an engineer chooses to instrument her laboratory’s testing environment (in this case, with a set of motion tracking cameras) so she can subject it to adequate scrutiny. **b)** An ornithologist would like to test a new hypothesis concerning the migration patterns of a species of bird, so assesses the cost of acquiring tracking capabilities with adequate power of discernment. **c)** Rather than purchasing sensors to maximize coverage, if benign activities can be precisely characterized, then negating that description permits surveillance which is sufficient to identify anomalous activities but with far fewer sensors.

That is, does the sensor history contain any evidence that the given sequence of locations was in fact not visited? First in [167], and then, with several refinements and under weaker assumptions, in the follow-up [168], Yu and LaValle provide an efficient method for this problem. Their approach is sound and complete in that it identifies inconsistencies between the story and the sensed history if and only if such inconsistencies exist. However, the strength of the validation (or, more correctly, the method’s *inability to invalidate*) must be understood modulo sensor data. The faculty to detect contradictions depends critically on sensor history, on the evidence that the sensors provide. The more limited the sensing, the fewer fibs you can catch.

A natural concern, then, is how to choose sensors. Suppose that the given story describes a pre-declared itinerary, a future path or structured collection of possible paths through an environment. Now, given a set of possible sensors one could deploy, when some path has been executed, which ones suffice to detect deviations from the itinerary? The present work considers an optimization variant where we ask for a minimal set of sensors that can accomplish this. Fig. 1.5 illustrates the breadth of use

cases for this scenario, in which the goal is to ensure detection of all deviations. It is important to note that with a given environment, given itinerary, and set of sensors, this may be impossible—the whole set of sensors may be inadequate. A concrete sort of application, based on selection of a suite of beam and occupancy sensors in an indoor environment, appears in Fig. 9.1.

1.7 DISSERTATION ORGANIZATION

This dissertation contains results from previous work by Rahmani and O’Kane [106–110]; Rahmani, Shell, and O’Kane [111–113]; Zhang, Rahmani, Shell, and O’Kane [171];

The organization of this dissertation is as follows. Chapter 2 reviews related work. Chapter 3 studies FM and FPM through covering and equivalence relations identifying mergeable states. Chapter 4 presents four integer linear programming formulations of FM and FPM. Chapter 5 concerns the problem of using autonomous robots to capture a structured video from the events occurring in an unpredictable environment. Chapter 6 addresses temporal logic planning with LTL hard specifications and LDL_f soft constraints. Chapter 7 studies temporal logic planning with LTL hard specifications and LTL soft constraints. Chapter 8 combines ideas from Chapter 5 and Chapter 6. Chapter 9 studies optimal sensor selection for detecting deviations from a planned itinerary. Chapter 10 concludes remarks and draws several future directions.

CHAPTER 2

RELATED WORK

2.1 FILTER MINIMIZATION PROBLEM

Building on a foundation of prior work on minimalism in robotics [37, 50, 87], combinatorial filters were originally formulated by LaValle [77, 78]. The key idea is to make, from the data accessible to the robot, a smallest abstraction still adequate to solve a given task.

Interest in forming combinatorial filters that are minimal, in the sense of minimizing the number of states, is motivated not only by the reduction in resources needed to execute such filters, but also by the insight into the nature of the underlying problems that arises from identifying the information required to solve those problems. The problem of performing this reduction automatically was first studied by O’Kane and Shell [98, 100], who proved via a reduction from the graph 3-coloring problem that the filter minimization problem is NP-hard. Saberifar *et al.* [125] showed that several special cases of filters, including tree and planar filters, remain hard to minimize, and that the filter minimization problem is NP-hard even to approximate. Recently, Zhang and Shell [172] proved that FM can be solved by finding a covering of the filter’s state space and then presented an algorithm that uses a SAT formulation to compute an exact solution to FM. In that work, the number of constraints of the SAT formula was exponential in the size of the input filter in the worst case. Our own prior work [108] proposed three different integer linear programming formulations of the filter partitioning minimization (FPM) problem, a variant of FM in which it is

required the reduced filter to partition the state space of the original filter. Recent work by Zhang, Rahmani, Shell, and O’Kane [171] proposed a SAT and an integer linear programming formulation of FM with only a polynomial number of constraints, which improves upon the previous formulation in [172]. The SAT formulation is improved even further by letting the constraints to be added to the formula lazily, only when the current solution is not a deterministic combinatorial filter.

In this work we also study the connection between filter minimization and the well-known notion of bisimulation, which is widely used in many fields for minimization other discrete structures. Bisimulation was discovered independently in at least three different fields: in modal logic, by van Benthem [153]; in process theory, by Milner [94] and Park [102]; and in set theory, by Forti and Honsell [44] (See the elaboration of Sangiorgi [127] upon the origins of bisimulation and simulation.) It is currently used across many fields, including automata and language theory [121, 123], coalgebra and coinduction [43, 124], and dynamical and control systems [54, 154]. Generally speaking, bisimulation can be used for at least two purposes: either to prove that two objects are behaviorally equivalent, or to minimize the size of a structure by forming the quotient under the coarsest bisimulation equivalence relation between elements of the original structure. This dissertation focuses on the latter application. Computing this coarsest bisimulation equivalence relation is generally performed using partition refinement algorithms [64, 101]. Details about bisimulation quotient algorithms appear in the survey by Cleaveland and Sokolsky [27].

We also establish a connection between filter minimization and the notion of simulation. The original notion of simulation was introduced by Milner [91–93], and later refined by Park [102]. This notion—which unlike bisimulation, is “uni-directed”—is used to prove that an object (for example, a state or a transition system) mimics another object; to show that a system is a correct implementation of a smaller, abstract system; and to make a smaller structure who with the original structure mutually

simulate each other. In the latter case, the smaller structure is obtained by making the quotient of the original structure under the *simulation equivalence* relation—the equivalence kernel of the largest simulation relation over the state space of the original structure. Various algorithms for simulation-based minimization and computing the largest simulation relation have been suggested by Cleaveland et al. [26], Henzinger et al. [57], Tan and Cleaveland [141], Ranzato and Tapparo [116], Bustan and Grumberg [13], Ranzato [115], and Gentilini et al. [47]. The algorithm of the latter was later corrected by van Glabbeek and Ploeger [155].

In the context of transition systems, apart from *simulation equivalence* and *bisimulation equivalence*, a variety of other kind of notions of equivalence on the state space of transition systems, including *trace equivalence*, *failure equivalence*, and *readiness equivalence*, among many others, have been introduced. For surveys, see [32, 52, 134, 156, 157].

The idea of using (bi)simulation-based equivalences for state space reduction have been used for a variety of other structures than transition systems, including tree automata [1, 1, 59], Markov chains [4], Markov decision processes [49], fuzzy transition systems [165], and timed systems [103].

2.2 PLANNING TO CHRONICLE

Our interest in understanding robot behavior in terms of the robots’ observations of a sequence of discrete events is, of course, not unique. One of the related problems is the *story validation problem* [167, 168], which can be viewed as an inverse of our problem. The aim there is to determine whether a given story is consistent with a sequence of events captured by a network of sensors in the environment. In our problem, it is the robot that needs to capture a sequence of events that constitute a desired story.

Video summarization is the problem of making a ‘good’ summary of a given video by prioritizing sequences of frames based on some selection criterion (importance, representativeness, diversity, etc.). Various approaches include identifying important objects [79], finding interesting events [53], selection using supervised learning [51], and finding inter-frame connections [84]. For a survey on video summarization see [150], which one might augment with the more recent results of [61, 85, 104, 170]. Girdhar and Dudek [48] considered the related *vacation snapshot problem*, in which the goal is to retain a diverse subset from data observed by a mobile robot. However, in such summarization techniques, the problem is essentially to post-process a collection of images already recorded. This dissertation, by contrast, addresses the problem of deciding which video segments the robot should attempt to capture in the first place.

For text-based and interactive narratives, a variety of methods are known for narrative planning and generating natural language stories [117, 118].

Closely related research is [133], which introduces the idea of using a team of autonomous robots, coordinated by a planner, to capture a sequence of events specifying a given narrative structure. That work raised (but did not answer) several questions, among which is how the robot can formulate effective plans to capture events relevant to the story specification. Here we build upon that prior effort showing how such plans can be formed in a principled way.

Related to our problem are also the theories of Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs), which are surveyed in [12, 77, 120, 131]. Our problems for planning to capture events in an unpredictable environment reduce to planning on product automata that are constructed from the inputs of the problems, and each of those product automata is either an MDP or a POMDP, depending on the amount of information available to the robot.

2.3 TEMPORAL LOGIC PLANNING WITH CONFLICTING SOFT SPECIFICATIONS

A vibrant community of researchers has been investigating the use of formal methods to plan the motions of robots for some time. Such techniques play an established and growing role in robotics for specifying the robots’ goals, verifying correctness of parts of systems, and synthesizing controllers with desired properties [24, 28, 33, 41, 42, 56, 67, 95, 114, 135]. A recent survey by Kress-Gazit, Lahijanian, and Raman reviews the current status of the field [72].

Our temporal logic planning is related to, but distinct from, several threads of prior work, which consider temporal logic planning in situations where no plan satisfying a given temporal logic formula can be synthesized. Fainekos [39] introduced an LTL revision problem, which, upon failures to plan a trajectory for an LTL formula, provides information about why that failure occurred, and how the LTL formula can be revised so that the transition system has a satisfiable trajectory for the revised formula. Kim *et al.* [67, 68] consider the *minimal revision problem* (MRP), which aims to find for a given specification (Büchi) automaton, a “closest” specification automaton for which the system has a trajectory. They prove that MRP is NP-hard, and then provide a SAT-based encoding and a heuristic algorithm for solving MRP. Lahijanian *et al.* [75] propose, based on a user-defined priority over atomic propositions, which they assumed to be low level tasks, an approach to measure how “close” is a trajectory to satisfy a given formula, and accordingly, propose an algorithm that generates a trajectory that has the minimum distance to the satisfaction of that given formula. Lahijanian and Kwiatkowska [76] later extended that idea for probabilistic environments modeled by MDPs.

Two recent results by Dimitrova *et al.* [35] and Tomita *et al.* [143] consider a problem, called *maximum realizability problem*, which is a synthesis problem from a

hard constraint and a set of soft constraints in the form of LTL formulas. The aim of this problem is to synthesize a *reactive transition system* (rather than a trajectory within a transition system). They consider the case where the soft constraints are of a specific kind of LTL formulas, those who assert that something *globally* holds. Their ideas are based on optimally refining or relaxing the soft specifications such that the resulting soft specifications along with the hard specification are realizable by a reactive transition system.

The closest work to our LTL planning with soft LTL constraints is by Tumova *et al.* [151], who address a similar problem but without the hard constraint; they consider the problem of making a trajectory that maximizes the sum of rewards from satisfying a set of conflicting LTL formulas. Their algorithm first makes a *generalized Büchi automaton* for each LTL formula, and then from those automata, using the idea of converting a generalized Büchi automaton to a Büchi automaton [5], it makes a transition-weighted Büchi automaton, in which the Cartesian product of the state spaces of the automata are copied into different layers, a layer for each of the original automata, to keep track of the set of LTL formulas for which a run over the automaton is satisfying. From this transition-weighted Büchi automaton and the transition system, a product automaton is constructed, on which an accepting lasso is synthesized using a modified version of nested-DFS [29]. Our algorithm, which is simpler, constructs a state-weighted (rather than a transition-weighted) product automaton, and then uses a greedy approach to synthesize on this product automaton, a short accepting lasso with minimum weight; we prove that an accepting lasso with minimum length and weight is computationally hard to find.

Our synthesis process over this product automaton is performed in two passes, first is the one-pass DFS of Tarjan’s algorithm [142] to compute the set of strongly connected components (SCCs) of the product automaton, and second is a pass that synthesizes the prefix of the lasso as a simple path from the initial state to a leader

of a SCC with minimum weight and the suffix of the lasso as a cycle within that SCC using BFS iteratively. The work of Tumova *et al.* [151] does not consider synthesis of a shortest accepting lasso. Two other results from the same authors [20, 152] consider for the classical setting of path planning, generating a finite trajectory that minimizes the amount of time the robot deviates only the less important ones of a set of conflicting safety rules. This problem is for finite trajectories and is treated differently.

Smith *et al.* [136] consider synthesizing an optimal trajectory satisfying an LTL formula. The generated plans are optimal in the sense of minimizing the maximum time delay between visiting states of a certain property. Our approach uses a different optimality criterion, based on user-specified preferences.

Finally, recent work by Wilde *et al.* [163] and by Nardi and Stachniss [97] shows how to elicit and exploit users' preferences about the robot's movements through its environment.

2.4 PLANNING TO CHRONICLE WITH SOFT CONSTRAINTS

There are several related work to this extension of our problem. Fu [45] considered preference planning over temporal goals in stochastic environments modeled by MDPs. She first proposes a general model, using a finite automaton, to specify preferences over temporal goals and then uses a mixed integer linear programming formulation to solve the problem of maximally satisfying the preferences within a finite-time execution. This problem, unlike our problem, does not require computing the set of Pareto optimal policies.

Cai *et al.* [16] consider an LTL planning problem in probabilistic environments, modeled by probabilistic MDPs, with two objectives where one objective is to maximize the probability of satisfying an LTL formula and the other objective is to

minimize the execution cost of the policy. They also consider a similar problem [15] where the transition probabilities of the MDP are unknown to the robot. For this problem, they provide a learning-based strategy.

The settings of these two problems are different from the setting of our problem in several respects: the tools used to model the environment, the languages used to specify the mission, and the objectives to optimize. In addition, these two problems, unlike our problem, do not involve satisfying an ordered list of soft specifications. They also consider in [14], temporal logic planning in deterministic environments with time-varying rewards that are only locally observable to the robot. The aim of this problem is to synthesize a trajectory that minimizes the violation of an LTL soft formula and maximizes accumulated reward during operation while nonetheless satisfying an LTL hard formula. This problem was treated differently than our problem and a single trajectory rather than a set of Pareto optimal trajectories was computed for it.

The problem of computing a policy that maximizes the probability of satisfying an LTL_f formula on a given MDP by Wells *et al.* [162] is also related. They compare the scalability of two approaches. This problem is not a multi-objective minimization problem and is reduced to the problem of synthesizing a policy that maximizes the probability of reaching the goal states of a product automaton treated as an MDP.

Related are also the problems of computing optimal policies for multi-objective MDPs (MOMDPs), MDPs with multiple reward functions, with lexicographic preferences over the rewards. For an example, see the work of Wray *et al.* [164]. Those problems usually do not require to compute a set of Pareto optimal policies but only an optimal policy.

Li *et al.* [81] considers planning with preferences over multiple LTL goals, and this problem also does not require computing a set of Pareto optimal policies.

Our problem reduces to the problem of computing the set of Pareto optimal

polices for a given multi-objective Markov decision process (MOMDP), and there are a variety of methods for computing such polices. See [119] for a survey. We chose the *convex hull value iteration algorithm*, by Barrett and Narayanan [6], among those methods for our implementation and combine this method with the idea of Mandow *et al.* [86].

2.5 SENSOR SELECTION FOR DETECTING DEVIATIONS FROM A PLANNED ITINERARY

The problem of reducing the sensor readings needed to establish some property has been the subject of extensive study in the discrete event systems literature (see the survey by Sears and Rudie [129]). That literature distinguishes sensor selection (e.g., [55]) from sensor activation (e.g., [19, 161]). The former considers, as studied in this dissertation, a one-shot decision at initialization of whether to adopt a sensor or not; the latter is an online variant that switches sensors on/off across time. This dissertation aims to fill the niche between such sensor-oriented work and the problem of story validation (as exemplified by Yu and LaValle [167, 168]).

The NP-completeness of minimal sensor selection for the properties of observability and diagnosability [126] was established by Yoo and Lafortune [140]. Recently, Yin and Lafortune [166] proposed a general approach to optimizing sensor selection that applies to a very wide set of problems and subsumes several previous methods. Their approach is capable of enforcing what they term ‘information state-based properties,’ essentially arbitrary predicates defined on states, which allow one treat a variety of fault detection and diagnosis tasks (including observability and diagnosability). Our itinerary validation problem, however, doesn’t fall within this class as it is not enough to ask whether a state is visited or not; specific transitions between states matter too.

The characteristic that differentiates itinerary validation is that the basic mathe-

mathematical objects under consideration are trajectories. This emphasizes an important distinction between language- and automata-based properties (cf. our [111, 113]), with fairly subtle implications for our model and approach. Itineraries will be taken to describe sequences of edges and in order to relate the world's structure to those sequences, we will form a product. That product may partially unroll or unfold the world, so that the choice of a sensor can affect elements less 'locally' than one might naturally expect. One implication for the model is that we directly treat situations where multiple sensors may be selected as a single logical unit as this occurs in the product anyway. Thence, the further implication is that our computational complexity (hardness) result utilizes a cover selection problem directly.

CHAPTER 3

NOTIONS OF EQUIVALENCE FOR STATE SPACE

MINIMIZATION OF COMBINATORIAL FILTERS

In this chapter, we study the filter minimization (FM) problem and the filter partitioning minimization (FPM) problem through the lens of relations on the state space of the original filter that specify pairs of mergeable states. We address FM and FPM by considering several distinct state-equivalence relations in the context of those problems. Figure 3.1 shows the connection between all those relations we study here. A detailed diagram of our contributions in this chapter appears in Figure 3.2.

The material of this chapter is based on the results of our work Rahmani and O’Kane [106, 110].

The organization of this chapter is as follows. In Section 3.1, we present the use of combinatorial filters for several robotic tasks. In Section 3.2, we present the formal definitions of the problems we study; In Section 3.3, we characterize exact solutions to FM and FPM; In Section 3.4, we show that for some filters, the difference between the sizes of the optimal solutions to FM and FPM can be in the order of the size of filter. In Section 3.5, we show that, though the bisimilarity relation always yields feasible solutions to both FM and FPM, it fails to produce optimal solutions for some filters. In Section 3.6, we introduce a variant of our problem that is solvable using the notion of bisimulation. In Section 3.7, we establish a connection between simulation and filter minimization. In Section 3.8, we introduce two relations, the union of all compatibility relations and the mergeability relation, that are useful for

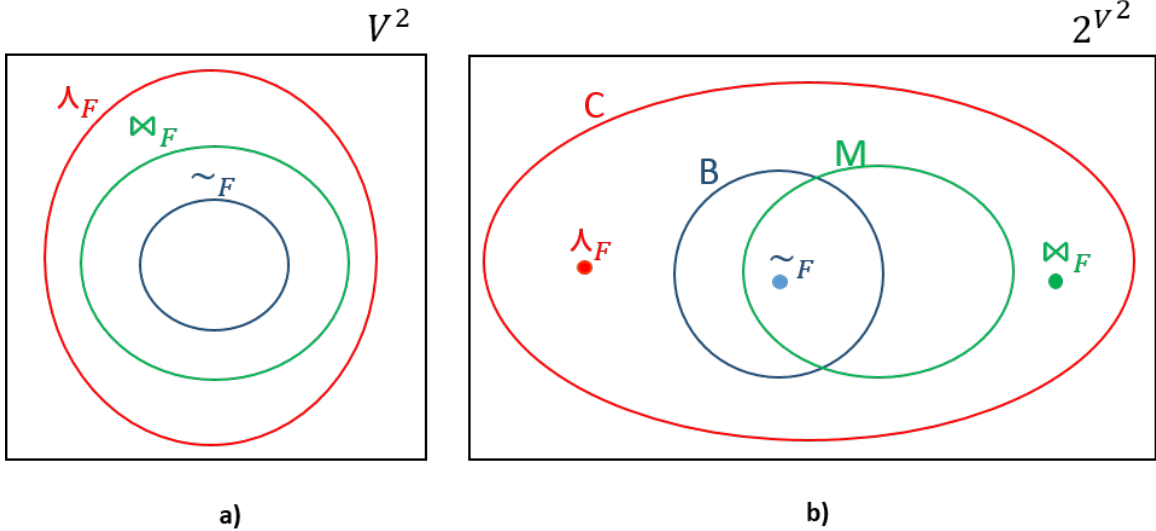


Figure 3.1: **a)** A Venn diagram showing the connection between the bisimilarity relation \sim_F , the union of all compatibility relations \wedge_F , and the mergeability relation \bowtie_F for a typical filter F with state space V . Note that it is possible that two or all these three relations coincide for some filters. **b)** A Venn diagram, over the space of state-state relations, showing the relationships between the set of all bisimulation relations (B), the set of all compatibility relations (C), and the set of all compatibility equivalence relations (M) for a typical filter F with state space V .

identifying some classes of filters for which FPM or FM is solvable in polynomial time. In Section 3.9, we describe several classes of filters for which FM or FPM is solvable in polynomial time.

3.1 COMBINATORIAL FILTERS FOR ROBOTIC TASKS

In this section, we present several examples.

A minimal motivating example

Before previewing our technical results, let us examine a simple example that illustrates the idea of state space minimization of combinatorial filters. Figure 1.2 shows the setting, inspired by the work of Tovar *et al.* [148], where the aim is to construct a

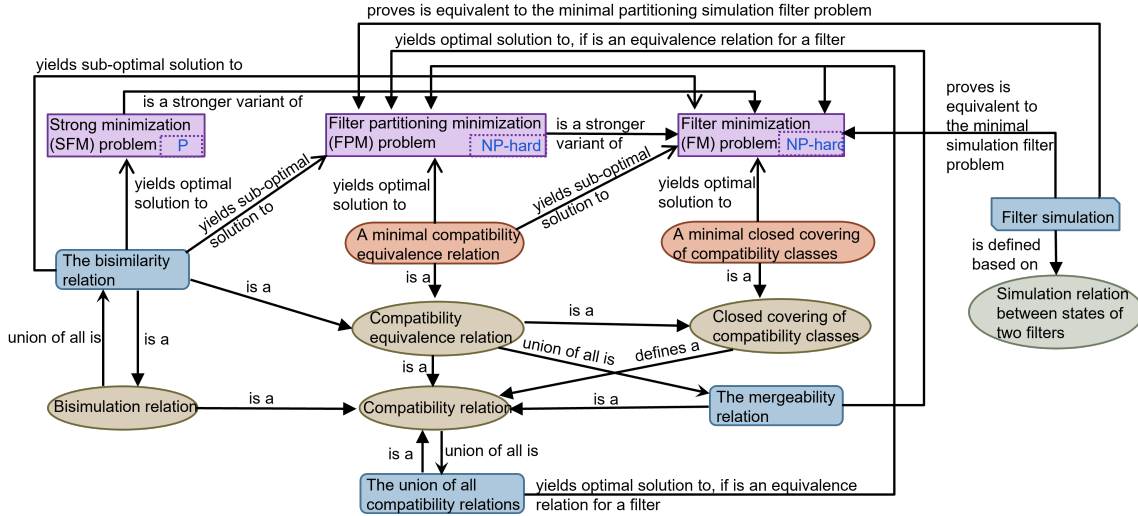


Figure 3.2: The connections between problems we study in this chapter and notions we provide to solve those problems. The main problem we study is FM, but we also study two stronger variants of it, FPM and SFM. FM and FPM are NP-hard, while SFM is in P. To solve FM and FPM, we introduce the notion of compatibility, which is a relaxed variant of bisimulation. The diagram shows the connection between this notion and optimal solutions to FM and FPM. In addition, it shows the connection between the notion of bisimulation to those three problems, while establishing a connection between the notion of simulation and those problems. It also shows three special relations, the bisimilarity relation, the union of all compatibility relations, and the mergeability relation.

combinatorial map of the environment and perform various tasks such as patrolling and navigation using robots with extremely simple sensors.

Suppose a mobile robot moves in an environment in which there are four landmarks 1-4. The robot does not know its location and it does not have a compass nor any odometer, but it does have a sensor by which it can sense the cyclic order of landmarks as observed from its current position. That is, the sensor reports the order in which the landmarks would be observed in a counterclockwise scan starting (without loss of generality) at the landmark with the smallest number. To illustrate, suppose the robot is located as shown in Figure 1.2a, at a location in region a . From that position, the sensor reads 1243 as the cyclic order of landmarks.

Figure 1.2 shows a (virtual) decomposition of the environment into ten regions a - j , where each region is a set of positions from which the robot sees the same cyclic

order of the landmarks. Notice that such a decomposition is obtained by a set of half lines, in which each half line is created for two landmarks whose order is swapped in the cyclic order observed by the robot if the robot crosses that half line.

The task of the robot in this example is to determine at any time, based on the information it receives through sensing the cyclic order of landmarks, whether it is located in region f or not.

A naïve combinatorial filter for this task appears in Figure 1.2b. Each of the fifteen states in this filter corresponds to a set of possible regions that might contain the agent. Edges indicate changes to that set that result from changes of the robot’s perception of the landmarks’ cyclic order. Colors on the states indicate the filter’s output, with the darker node corresponding to certainty that the agent is in region f and the lighter nodes indicating otherwise.

For the original filter in Figure 1.2b, FM has only one optimal solution, with 3 states. This optimally-reduced filter, which is shown in Figure 1.2c, is also an optimal solution to FPM; note that the state space of the reduced filter partitions the state space of the original filter in the sense that each state in the original filter is mimicked by only a single state (rather than several states) in the reduced filter. In regard with FM, it is possible that several states in the reduced filter together play the role of a state in the original filter. This optimal solution in Figure 1.2c is not only preferred to the naïve filter for the robot to keep but it also reveals the fact that perhaps surprisingly the sensor reading ‘1423’ corresponding to region d , which is not adjacent to the target region f , is a crucial one for the robot to achieve its task. In contrast, applying the technique of bisimulation minimization does not reduce the state space of the original filter at all. In Theorem 3, we describe a class of combinatorial filters on which bisimulation minimization does not reduce the state space of the original filter at all, whereas the optimally reduced filter has only two states.

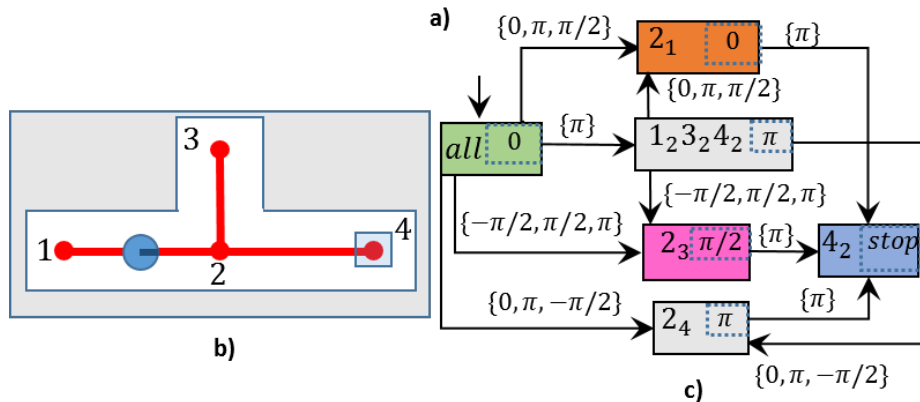
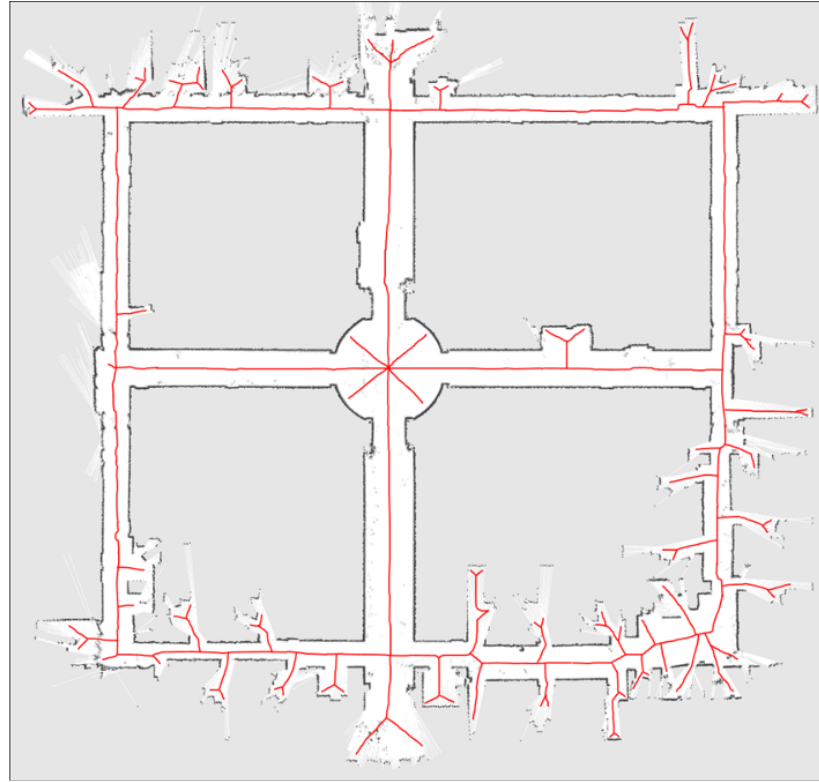


Figure 3.3: **a)** A map of the third floor of ACES building of University of Texas at Austin [7, 8, 96]. The map is overlaid with the generalized Voronoi graph (GVG) of the environment in red. **b)** A simple environment and its GVG for illustration purposes. **c)** A naïvely-constructed combinatorial filter, which the robot can use to navigate through the environment in part b) to reach point 3, starting from full location uncertainty.

An example of a combinatorial filter as a plan

In the specific example we consider here, a combinatorial filter is used to navigate in an environment, using an approximated generalized Voronoi graph (GVG) of the

environment to reach a goal location, starting from full location uncertainty. For an example, see in Figure 3.3 a map of a typical university building [7,8], which we have augmented with its GVG in red. We have also shown a simpler environment with its GVG in Figure 3.3b for illustration. Figure 3.3c shows a naïvely-constructed combinatorial filter, which the robot can use to navigate through that environment to reach point 4, starting from full location uncertainty. Each state of this filter represents a set of possible locations in which the robot can be. Each location is written in the form A_B to mean that the robot has arrived at junction A from junction B . Each state is labeled with an angle ($0, \pi/2, \pi$, etc), which is, in fact, the output or the color of the state, and tells the robot which corridor to follow. For example, angle 0 tells the robot to go straight, while angle π tells the robot to turn around and follow the corridor from which it has entered the junction. Each transition of the filter shows how the state changes in response to those movements. Each transition is labeled with an observation which consists of a list of directions the robot can observe to leave the next junction. For example, the label of the transition from state 'all' to state 2_1 is $\{0, \pi, \pi/2\}$, which means that when the robot is entering junction 2 from junction 1, the set of all possible directions it observes to leave junction 2 are $\{0, \pi, \pi/2\}$. The robot executes the plan represented by a filter by repeatedly doing the movement associated with the current state, observing the set of all possible directions in the junction using its sensor, and transitioning in the graph from the current state using a transition labeled by the perceived observation.

Given a GVG and a goal location, there is a simple algorithm by which one can make a naïve combinatorial filter the robot can use to navigate, starting from total location uncertainty, to reach that goal location. Unfortunately, those naïvely constructed combinatorial filters can be big for real-world applications, but fortunately filter reduction assists reduce resource consumption.

Motivation for minimization

We now consider in Figure 3.4, a simple example, introduced by Tovar et. al. [147], that shows a family of filters for which the minimal filter is much smaller than the original filter. In this example two agents are located in a donut-shaped environment divided into n regions by n sensor beams. The task of the system is to determine at any time whether the two agents are in the same region or not. When an agent crosses a beam, the system knows which beam it was, but it does not know which agent it was, nor does it know if the crossing was clockwise or otherwise. We also assume that at no time, the two agent simultaneously cross a single or two separate beams. Let $R = \{1, 2, \dots, n\}$ be the set of all regions and let (r, r') be a tuple showing that the first agent is in region r and the second agent is in region r' . Accordingly, the set of all such tuples, i.e., the set of all possible region assignments, is $A = R \times R$ and the set of all subsets of possible region assignments is $\mathcal{I} = \text{pow}(R \times R)$. Each element of \mathcal{I} is a configuration the environment could be in, and accordingly, each element of \mathcal{I} represents a state of the naïve filter used for accomplishing the task of the system. Subsequently, for each n , the naïve filter for a donut-shaped environment with n regions has $2^{n^2} - 1$ states in the worst case (including some states that may be unreachable). A minimal filter for accomplishing the same task of the naïve filter has many fewer states. As an example, for an environment of 3 regions, the naïve filter has 511 states while the minimal filter, as shown in the right side of Figure 3.4, has only 4 states.

Note that although there is a process by which we can make a naïve filter for any instance of this problem with any n , it is not known how we can make the minimal filter for that instance. Subsequently, we need to make the naïve filter, but because that filter is very big, we need to minimize it.

In the next section we formally define the problems we consider in this chapter.

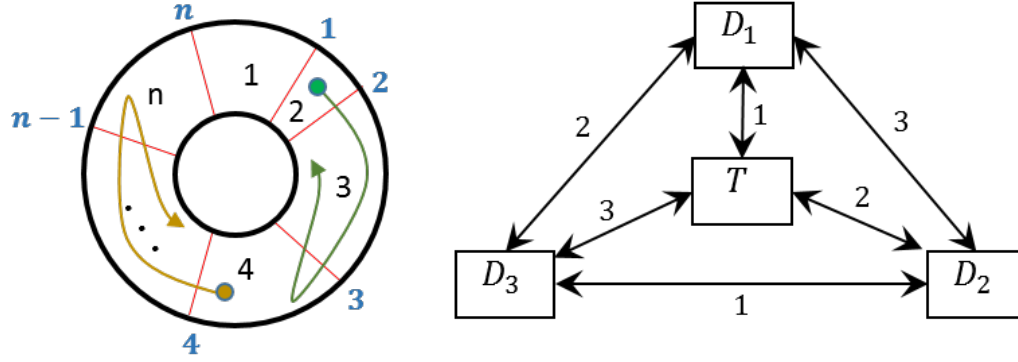


Figure 3.4: **(left)** A region divided by n sensor beams, in which two agents are located. When an agent crosses a beam, the system knows which beam sensor it was, but it does not know which agent it was, nor does it know the direction of crossing. The agents do not simultaneously cross beams. The task of the system is to determine at each time whether the agents are in the same region or not. **(right)** The smallest filter the system can use for accomplishing its task for an environment of 3 sensor beams.

3.2 DEFINITIONS

This section presents basic definitions used in this chapter and the next chapter.

We are interested in filters that model the behavior of a robot in response to a discrete, finite sequence of observations. The following definitions are equivalent to those introduced by O’Kane and Shell [100].

Definition 1. A *filter* is a 6-tuple $(V, Y, C, \delta, c, v_0)$ in which:

- V is a finite set of *states*,
- Y is a set of possible *observations*, representing the input space of the filter,
- C is a set of *outputs*, sometimes called *colors*, representing the outputs produced by the filter,
- $\delta : V \times Y \rightarrow V \cup \{\perp\}$ is the *transition function* of filter,
- $c : V \rightarrow C$ is a function assigning to each state $v \in V$ a color, and
- $v_0 \in V$ is the initial state.

The set of all possible observation sequences is denoted Y^* . Given an observation sequence $s = s_1s_2 \cdots s_n$, we use $s_{i..j}$ to denote the subsequence $s_i s_{i+1} \cdots s_j$. For each i , the subsequence $s_{i..i}$ of s represents the observation s_i . Filters are readily shown as directed graphs, in which the states are vertices and edges are determined by the transition function. Recall the examples in Figure 1.2b–c.

In this definition, we use symbol \perp to mean null, and if for a state-observation pair (v, y) it holds that $\delta(v, y) = \perp$, then we mean that state v does not have any outgoing transition labeled y . We interpret this to mean that we can be sure that observation y will not, because of some structure in the robot’s environment, occur when the filter is in state v . In the graph view, there would simply be no outgoing edge from v labeled y .

Note that Definition 1 ensures that from any state, for any observation, at most one transition can happen. The next definition makes this idea more precise.

Definition 2. Let $F = (V, Y, C, \delta, c, v_0)$ be a filter, $v \in V$ be a state, and $s = s_1s_2 \cdots s_n \in Y^*$ be an observation sequence where each s_i is a member of Y . We say that s is *trackable* from v if there is a sequence of states q_0, q_1, \dots, q_n such that:

- $q_0 = v$, and
- $\delta(q_i, s_{i+1}) = q_{i+1}$ for all $0 \leq i < n$.

Notice in particular that if s is trackable from some state v , then the corresponding state sequence q_0, q_1, \dots, q_n is unique. Given a state $v \in V$ and an observation sequence $s \in Y^*$ trackable from v , we write $\delta^*(v, s)$ to denote the state reached by tracing s starting from v . If s is not trackable from v , we write $\delta^*(v, s) = \perp$. For the empty string ϵ , we define $\delta^*(v, \epsilon) = v$ for all states v , and define that ϵ is trackable for all states v . For a state $v \in V$, we use S_v to denote the set of all observation sequences that end in v when traced from the initial state. i.e., $S_v = \{s \in Y^* \mid \delta^*(v_0, s) = v\}$

We can now define the language of a filter, which plays a crucial role in filter reduction.

Definition 3. The *language of a state* v , denoted $L(v)$, is the set of all observation sequences trackable from v . The *language of a filter* F , denoted $L(F)$, is the language of its initial state: $L(F) = L(v_0)$.

Before we can speak meaningfully about reduction of filters, we need a definition of filter equivalence with respect to a language.

Definition 4. Let $F_1 = (V_1, Y, C, \delta_1, c_1, v_0)$ and $F_2 = (V_2, Y, C, \delta_2, c_2, w_0)$ be two filters with the same observation space Y and the same color space C . Let $L \subseteq Y^*$ denote a language of observation sequences. We say that F_1 is *equivalent to* F_2 with respect to L , denoted $F_1 \stackrel{L}{=} F_2$, if for any observation sequence $s \in L$:

- $\delta_1^*(v_0, s) \neq \perp$,
- $\delta_2^*(w_0, s) \neq \perp$, and
- $c_1(\delta_1^*(v_0, s)) = c_2(\delta_2^*(w_0, s))$.

This definition says that any observation sequence that is in L is trackable by both F_1 and F_2 , and that both of them produce the same output while tracing that sequence. However, for observation sequences that are not in L , it does not say anything about the outputs generated by the two filters, nor does it require that the observation languages of F_1 and F_2 should be the same. The central problem this work studies is called the *filter minimization problem*.

Problem: Filter minimization (FM) [100]

Input: A filter F .

Output: A filter F^* such that $F \stackrel{L(F)}{=} F^*$ and the number of states in F^* is minimal.

Notice that this problem allows the language $L(F^*)$ of the optimally reduced filter to be a proper superset of the language $L(F)$ of the original filter. This is reasonable because observation sequences in $L(F^*) - L(F)$ will not occur, as we assumed above.

We also study a variant of FM in which we require the reduced filter to have a special property which we define as follows.

Definition 5. Let $F_1 = (V_1, Y, C, \delta_1, c_1, v_0)$ and $F_2 = (V_2, Y, C, \delta_2, c_2, w_0)$ be two filters. Denoted $F_1 \doteq F_2$, we say that F_2 *partitions the state space* of F_1 if for each $v \in V_1$, there is a single state $w \in V_2$ such that for any observation sequence $s \in Y^*$, if $\delta_1^*(v_0, s) = v$, then $\delta_2^*(w_0, s) = w$.

We call this variant the *filter partitioning minimization problem*.

Problem: Filter partitioning minimization (FPM)

Input: A filter F .

Output: A filter F^* such that $F \xrightarrow{L(F)} F^*$, $F \doteq F^*$, and the number of states in F^* is minimal.

FPM requires not only the reduced filter to produce for each observation sequence trackable by the original filter, the same output produced for that observation sequence by the original filter but it also enforces that for each state s of the original filter, only a single state in the reduced filter to play the role of s .

Because much of what follows deals with relations over the set of a filter's states, we will rely on some elements of standard notation for such relations.

For a given filter F , we use I_F to denote the identity relation on the state set V of F , i.e. $I_F = \{(v, v) \mid v \in V\}$. If $R \subseteq V \times V$ is an equivalence relation on V , then it partitions V into a set of equivalence classes. For any $v \in V$, the equivalence class of v in R is denoted $[v]_R$, so that $[v]_R = \{w \in V \mid (v, w) \in R\}$. In particular, for any $v, w \in V$, if $(v, w) \in R$, then $[v]_R = [w]_R$. Finally, the set of all equivalence classes of R is called the quotient of V under R , denoted V/R .

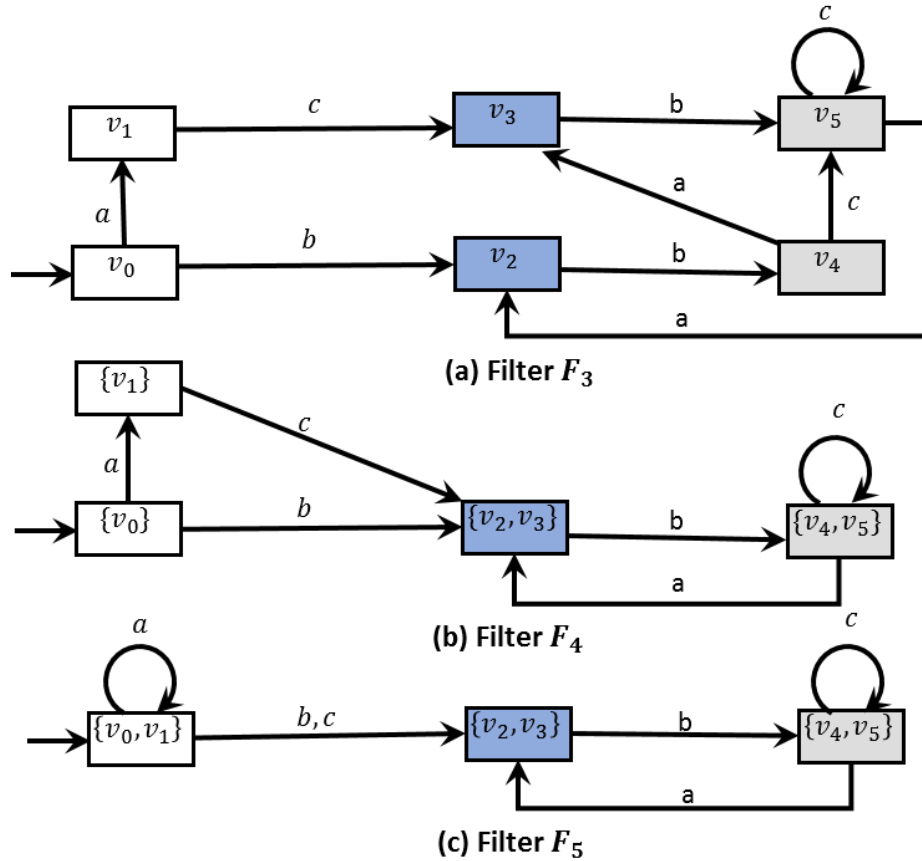


Figure 3.5: **a)** A sample filter F_3 . **b)** A minimal filter F_4 such that $F_3 \stackrel{L(F_3)}{=} F_4$ and $L(F_3) = L(F_4)$. **c)** A minimal filter F_5 such that $F_3 \stackrel{L(F_3)}{=} F_5$.

3.3 FILTER MINIMIZATION VIA QUOTIENT OPERATIONS

In this section, we show how the process of filter reduction can be understood as a quotient operation with respect to certain kinds of relations over the states of the input filter.

Exact solution to FPM

Therefore, we must establish conditions on the relation that guarantee that this merging operation makes sense.

The following definition establishes those conditions.

Definition 6. Let $F = (V, Y, C, \delta, c, v)$ be a filter and let $R \subseteq V \times V$ denote a relation over the states of F . We say that R is a *compatibility relation* for F , if for any $(v, w) \in R$:

1. $c(v) = c(w)$, and
2. for any $y \in Y$, if $\delta(v, y) \neq \perp$ and $\delta(w, y) \neq \perp$, then $(\delta(v, y), \delta(w, y)) \in R$.

Accordingly, two states are said to be *compatible* if they are related by a compatibility relation. To illustrate the notion of compatibility, consider filter F_3 , depicted in Figure 3.5. Some compatibility relations for F_3 are $R_1 = \emptyset$, $R_2 = \{(v_0, v_1)\}$, and $R_3 = \{(v_3, v_2), (v_5, v_4), (v_5, v_5), (v_2, v_3), (v_4, v_5)\}$.¹

Now we can define the notion of a quotient filter with respect to a compatibility equivalence relation.

Definition 7. For a filter $F = (V, Y, C, \delta, c, v_0)$, and a relation $R \subseteq V \times V$ that is both a compatibility relation and an equivalence relation (henceforth, a compatibility equivalence relation), the *quotient of F under R* is the filter $F/R = (V/R, Y, C, \delta', c', [v_0]_R)$, in which

$$\delta'([v]_R, y) = \begin{cases} [\delta(w, y)]_R & \text{if } \exists w \in [v]_R \text{ with } \delta(w, y) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

and $c'([v]_R) = c(v)$.

Note that Definition 6 ensures that every transition in a quotient filter is well-defined. Because R must be a compatibility relation, if two states v and w that share some outgoing observation y are merged, then the resulting states $\delta(v, y)$ and $\delta(w, y)$ must be merged as well. Consider filter F_3 depicted in Figure 3.5. A compatibility

¹Note that the notion of compatibility relation is different from the usual notion of simulation, in that its second condition is weaker than is required of a simulation relation. In fact, two states can be compatible (can exist in a compatibility relation) while neither of them simulates another. We discuss simulation in more detail in Section 3.7.

equivalence relation for this filter is $R = I_{F_3} \cup \{(v_3, v_2), (v_5, v_4), (v_2, v_3), (v_4, v_5)\}$. The quotient of F_3 under this relation, F_3/R , is filter F_4 , depicted in Figure 3.5.

The next lemmas establish that, though this quotient operation may increase the language of the filter, it does not change the behavior, in the sense of Definition 4.

Lemma 1. For any filter $F = (V, Y, C, \delta, c, v_0)$ and any compatibility equivalence relation R for F , $L(F) \subseteq L(F/R)$.

Lemma 2. For any filter $F = (V, Y, C, \delta, c, v_0)$ and any compatibility equivalence relation R for F , $F \xrightarrow{L(F)} F/R$.

Lemma 3. For any filter $F = (V, Y, C, \delta, c, v_0)$ and any compatibility equivalence relation R for F , $F \doteq F/R$.

Proof. Let us prove Lemmas 1, 2 and 3 all together. Assume F/R is defined according to Definition 7. By induction on the length of observation sequences $s \in Y^*$ we show that if $s \in L(F)$ and $\delta^*(v_0, s) = v$ for a $v \in V$, then $\delta'^*([v_0]_R, s) = [v]_R$. This means that for each state $v \in V$, there is a single state $w = [v]_R$ in F/R such that for each $s \in L(F)$, if $\delta^*(v_0, s) = v$, then $\delta'^*([v_0]_R, s) = w$, which by Definition 5 means that $F \doteq F/R$, proving Lemma 3. It also implies that if $s \in L(F)$, then $s \in L(F/R)$, meaning that $L(F) \subseteq L(F/R)$, proving Lemma 1. Furthermore, the conclusion that $\delta'^*([v_0]_R, s) = [v]_R$ will imply that $c(\delta^*(v_0, s)) = c'(\delta'^*([v_0]_R, s))$ given that $c(v) = c'([v]_R)$ by Definition 7. This, coupled with Lemma 1 and Definition 4, proves Lemma 2.

Now, we turn to the induction, which is illustrated in Figure 3.6. The statement holds for the base case, i.e., $s = \epsilon$, since by definition it holds that $\epsilon \in L(F)$, $\delta^*(v_0, \epsilon) = v_0$, and $\delta'^*([v_0]_R, \epsilon) = [v_0]_R$. Assume that the statement holds for all observation sequences s with length k , and let s' be any observation sequence with

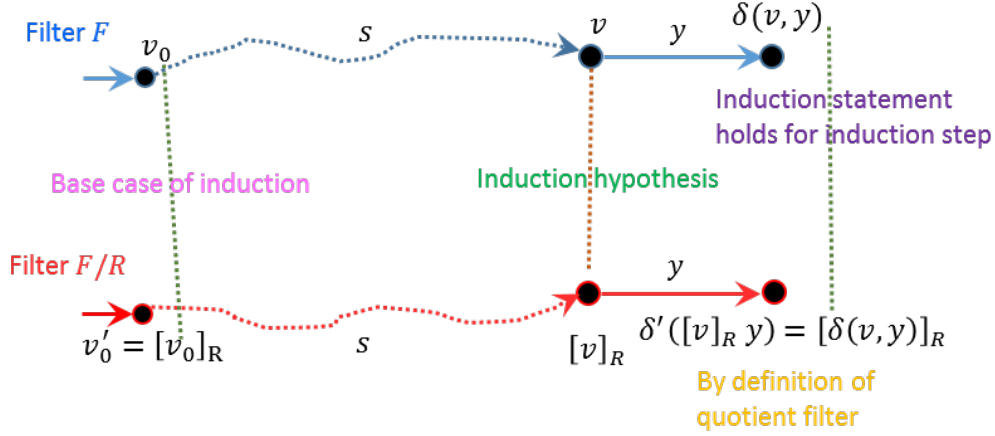


Figure 3.6: It illustrates the induction in the proof of Lemma 1, Lemma 2, and Lemma 3. The induction is on the lengths of observation sequences and it states that for each observation sequence s , if filter F reaches state v by tracing s from its initial state, then filter F/R , the quotient of F under a compatibility equivalence relation R , reaches state $[v]_R$ by tracing the same observation sequence from its own initial state.

length $k+1$. If $s' \in L(F)$, then $s' = sy$ for an observation sequence s with length k and an observation $y \in Y$. Let $\delta^*(v_0, s) = v$ for a state v . By the induction assumption, it holds that $\delta'^*([v_0]_R, s) = [v]_R$. Now, we have that $\delta^*(v_0, s') = \delta(\delta^*(v_0, s), y) = \delta(v, y)$, and that $\delta'^*([v_0]_R, s') = \delta'(\delta'^*([v_0]_R, s), y) = \delta'([v]_R, y)$. Given that $s' \in L(F)$, it holds that $\delta(v, y) \neq \perp$. This by the definition of δ' in Definition 7 implies that $\delta'([v]_R, y) = [\delta(v, y)]_R$. Now, the inductive step is proved since and . \square

Next we prove that the state space of any can be seen as the quotient of the state space of the original filter under some compatibility equivalence relation. Let the input to FPM be a filter $F_1 = (V_1, Y, C, \delta_1, c_1, v_0)$ and let $F_2 = (V_2, Y, C, \delta_2, c_2, w_0)$ be on optimal solution of FPM with input F_1 . Given that $F_1 \doteq F_2$, for each $v \in V_1$ there exists a single state $w \in V_2$ such that for any observation sequence s , if $\delta_1^*(v_0, s) = v$, then $\delta_2^*(w_0, s) = w$. Let $f : V_1 \rightarrow V_2$ denote that one-to-one mapping, i.e., for every observation sequence $s \in L(F_1)$, $\delta_2^*(w_0, s) = f(\delta_1^*(v_0, s))$. We consider the following results.

Lemma 4. Function f is surjective.

Proof. We prove that each state in F_2 is mapped to by at least one state in F_1 via f . For the sake of contradiction, suppose that there exists a state z in F_2 that is not mapped to by f from any state in F_1 . There are two cases.

1. If no observation sequence that ends or passes through z is in $L(F_1)$, then we can construct a new filter F_3 from F_2 by removing state z . Clearly, $F_1 \xrightarrow{L(F_1)} F_3$ and F_3 has fewer states than F_2 . This contradicts the construction that F_2 is minimal.
2. If there exists an observation sequence $s \in L(F_1)$ that ends or passes through z when traced in F_2 , then let $k \leq |s|$ be an integer such that $\delta_2^*(w_0, s_{1..k}) = z$. By the structure of filters, and given that $s \in L(F_1)$, we conclude that $s_{1..|s|-1} \in L(F_1)$, $s_{1..|s|-2} \in L(F_1)$, \dots , and ultimately $s_{1..k} \in L(F_1)$. This means that $z = f(\delta_1^*(v_0, s_{1..k}))$, which is a contradiction.

□

Given such a function f , we define an equivalence relation $R_f \subseteq V \times V$ so that $(v, w) \in R_f$ if and only if $f(v) = f(w)$. Note that there is a one-to-one correspondence between the equivalence classes $[v]_{R_f}$ of R_f and the states of F_2 .

Lemma 5. For any filter F_1 and any with input filter F_1 , the equivalence relation R_f they induce is a compatibility relation.

Proof. First observe that by the construction of R_f , for any $v, w \in V$, if $(v, w) \in R_f$, then v and w are mapped to a single state in F_2 . Let $[v]_{R_f}$ be such a state. To show that R_f is a compatibility relation, we prove that conditions (1) and (2) of Definition 6 hold for any v and w for which $(v, w) \in R_f$. Suppose that condition (1)

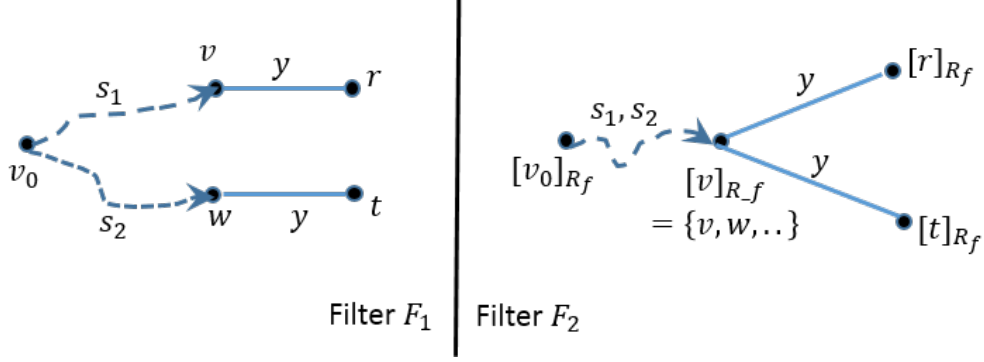


Figure 3.7: An illustration of the proof of Lemma 5. Filter F_2 is a minimal filter for which $F_1 \xrightarrow{L(F_1)} F_2$ holds. The state space of F_2 corresponds to the quotient of the state space of F_1 under the equivalence relation R_f . The assumption for this relation is $(v, w) \in R_f$ but $(r, t) \notin R_f$.

does not hold, that is, $c_1(v) \neq c_1(w)$, which means $c_2([v]_R)$ is different from $c_1(v)$ or $c_1(w)$. Without loss of generality assume that $c_1(v) \neq c_2([v]_{R_f})$. Let $s \in L(F_1)$ be an observation sequence such that $\delta_1^*(v_0, s) = v$. By definition of f , we have that $\delta_2^*(w_0, s) = [v]_{R_f}$. But, $c_1(\delta_1^*(v_0, s)) = c_1(v) \neq c_2([v]_{R_f}) = c_2(\delta_2^*(w_0, s))$, which by Definition 4 contradicts that $F_1 \xrightarrow{L(F_1)} F_3$.

Now suppose that condition (2) does not hold, which means that there exists $y \in Y$, such that $\delta_1(v, y) \neq \perp$ and $\delta_1(w, y) \neq \perp$ but $(\delta_1(v, y), \delta_1(w, y)) \notin R_f$. Let $r = \delta_1(v, y)$ and $t = \delta_1(w, y)$. We argue that if this is the case, then F_2 is not a filter, which is a contradiction. Figure 3.7 illustrates this proof. Let s_1 and s_2 be two observation sequences that end in v and w , respectively, when traced by F_1 . This means that states v and w are in the same equivalence class of R_f , and thus, they are mapped to a single state, such as $[v]_{R_f}$, in F_2 ; hence, both s_1 and s_2 end in $[v]_{R_f}$ when traced by F_2 . Consider also that observation sequences s_1y and s_2y end in r and t , respectively, when traced by F_1 . Observe that from state $[v]_{R_f}$ there should be two outgoing edges with the same label y , one of which goes to $[r]_{R_f}$ and another goes to $[t]_{R_f}$. Because F_2 is a filter, the only way to reach r by tracing s_1y from the initial state is to have an edge labeled by y , that goes from $[v]_{R_f}$ to $[r]_{R_f}$. We can use the same argument to prove that there should be an outgoing edge labeled by y

that connects $[v]_{R_f}$ to $[t]_{R_f}$. This implies that F_2 has two edges labeled y from $[v]_{R_f}$, a contradiction. \square

In particular, since R_f is both an equivalence relation and a compatibility relation for F_1 , it is meaningful to consider the quotient filter F_1/R_f . Moreover, F_1/R_f is structurally identical to the minimal filter F_2 . Of course, in the context of filter partitioning minimization, F_2 is unknown, so we cannot expect to compute F_1/R_f directly.

Nevertheless, the impact of Lemma 5 is that we can view the problem of filter minimization as equivalent to the problem of identifying a suitable compatibility equivalence relation with which to construct a quotient filter —there always exists some such relation for which the quotient leads to the minimal filter. This directly implies the following result.

Theorem 1. Let F be a filter and let R be a compatibility equivalence relation for F with minimum number of partitions. The filter $F^* = F/R$ is a minimal filter for which $F \xrightarrow{L(F)} F^*$ and $F \overset{\circ}{=} F^*$ holds.

Proof. By Lemma 2 and that R is a compatibility equivalence relation, $F \xrightarrow{L(F)} F^*$. If F/R is not minimal, that is if there is another filter F_2 with fewer number of states than F/R , then by Lemma 5, the relation R would not be a compatibility equivalence relation with minimum number of equivalence classes. \square

Exact solution to FM

Zhang and Shell [172] proposed an algorithm for computing an exact solution to FM. Their algorithm first constructs the *compatibility graph* of filter, which is an undirected graph whose vertices are the states of filter and its edges connect pairs of distinct compatible states. Then, it forms a SAT formulation that aims to find

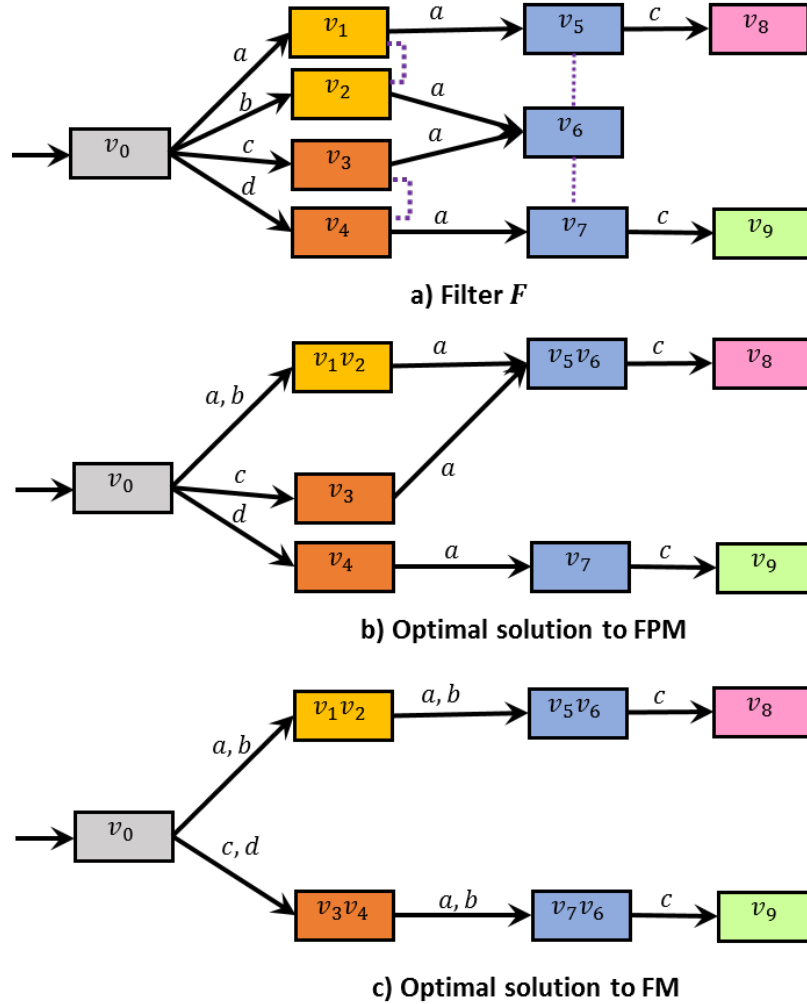


Figure 3.8: **a)** A sample filter F . **b)** A minimal solution of FPM for filter F . **c)** The minimal solution of FM for filter F .

a clique covering of the graph with minimum number of classes subject to a list of zipper constraints where each constraint enforces that if a set of states are merged, then the set of states they go by an observation must also be merged. An exact solution to FPM is computed by finding over the same graph, a clique partitioning rather than a clique covering. The difference between a partitioning and a covering is that a state is assigned to one and only one class of the partitioning while a state can be shared between two or several classes of the covering.

To see how this differentiates between an optimal solution to FM and an optimal

solution to FPM, we use a filter similar to that Zhang and Shell [172] used to demonstrate an optimal solution to FM. This filter, named F , is shown in Figure 3.8a. In this filter, those states that are compatible are connected by dashed lines. In this filter, states v_1 and v_2 are compatible, v_3 and v_4 are compatible, and v_6 is compatible with both v_5 and v_7 . Note that v_5 and v_7 are not compatible with each other. FPM with input F has two optimal solutions, in one of which, v_6 is merged with v_5 , and in the other one, v_6 is merged with v_7 . Figure 3.8b shows the one in which v_6 is merged with v_5 .

FM for the same filter has a single optimal solution, which is shown in Figure 3.8c. Note that because an optimal solution to FPM is computed by finding a partitioning rather than a covering, in a solution to FPM state v_6 is merged either with v_5 or with v_7 , while because an optimal solution to FM is computed by finding a covering, in the optimal solution to FM, state v_6 is split between two compatibility classes $\{v_5, v_6\}$ and $\{v_6, v_7\}$.

We now formalize an exact solution of FM via the notions of relation and covering, but before that we first consider the following definitions.

Given a filter $F = (V, Y, C, \delta, c, v_0)$, a *compatibility class* over V is a non-empty set $L \subseteq V$ in which each pair of states are compatible. A set of compatibility classes $\mathbf{M} = \{L_1, L_2, \dots, L_k\}$ over V is closed if for every $y \in Y$ and $i \in \{1, \dots, k\}$, there exists a $j \in \{1, \dots, k\}$ such that $\bigcup_{v \in L_i: \delta(v, y) \neq \perp} \delta(v, y) \subseteq L_j$. We use $\Delta_{\mathbf{M}}(L_i, y)$ to represent one of such compatibility classes L_j . A closed covering for V is a closed set of compatibility classes $\mathbf{M} = \{L_1, L_2, \dots, L_k\}$ such that for each $v \in V$, there exists at least an integer $i \in \{1, \dots, k\}$ such that $v \in L_i$. Note that a closed covering for V always forms a *tolerance* relation, a reflexive and symmetric relation, which is formed by relating for each compatibility class of the covering, all pairs of states within that class and also relating each state with itself.

Given a closed covering of the state space of the filter, we can form a quotient

filter as follows.

Definition 8. For a filter $F = (V, Y, C, \delta, c, v_0)$, and a closed covering $\mathbf{M} = \{L_1, L_2, \dots, L_k\}$ of V , the *quotient of F under \mathbf{M}* is the filter $F/\mathbf{M} = (\mathbf{M}, Y, C, \delta', c', L_0)$, in which L_0 is a compatibility class such that $v_0 \in L_0$, and for each $L \in \mathbf{M}$, $\delta'(L, y) = \Delta_{\mathbf{M}}(L, y)$ if $\Delta_{\mathbf{M}}(L, y) \neq \emptyset$ and otherwise $\delta'(L, y) = \perp$, and $c'(L) = c(v)$, where v is a state within L .

Accordingly, an optimal solution to FM is obtained by computing a closed covering with minimum number of classes.

Lemma 6. Let F be a filter with state space V and let \mathbf{M} be a closed covering of V with minimum number of compatibility classes. The filter $F^* = F/\mathbf{M}$ is a minimal filter for which $F \xrightarrow{L(F)} F^*$ holds.

Proof. The proof, which we omit, combines similar results and proofs of Lemma 1, Lemma 2, and Lemma 5. □

Consider that any compatibility equivalence relation over the state space of the filter is a closed covering of the state space, and therefore, any feasible solution to FPM is also a feasible solution to FM. As result, any heuristic or greedy algorithm for FPM can be used to compute a feasible solution to FM. The next section studies how big the difference between the sizes of optimal solutions to FPM and FM can be.

3.4 MAXIMUM DIFFERENCE BETWEEN THE SIZES OF OPTIMAL SOLUTIONS TO FM AND FPM

In this section, we show that there exist filters for which the difference between the sizes of optimal solutions to FM and FPM can be in the order of the size of the filter.

Theorem 2. For any $n \geq 10$, there exists a filter F with n states such that the difference between the sizes of an optimal solution to FPM with input F and an optimal solution to FM with input F is $\lfloor \frac{n-6}{4} \rfloor$.

Proof. Let n be an integer such that $n \geq 10$. We first consider the case where $n - 6$ is divisible by 4, that is, where n is one of the numbers 10, 14, 18, 22, 26, \dots . For that n , we construct the filter F , which has n states, as shown in Figure 3.9a. This filter generalizes the filter we used in Figure 3.8a. In fact, the number 10 in this theorem comes from the fact that the filter in Figure 3.8a has 10 states and that the filter in Figure 3.8a is the smallest filter within the family of filters described by Figure 3.9a. To color the states of F , we have used $\frac{n-6}{2} + 4$ distinct colors: 1 color for v_0 ; 1 color for v_{n-1} ; 1 color for v_{n-2} ; 1 color for v_{n-5}, v_{n-4} , and v_{n-3} ; and the remaining $\frac{n-6}{2}$ colors for v_1 through v_{n-6} . Note that the states in the second column, states v_1 through v_{n-6} , appeared in pairs such that the two states of each pair have the same color. Observe that in filter F , for each color, except color blue, all states with that color are compatible.

The states that have color blue are v_{n-3}, v_{n-4} , and v_{n-5} . States v_{n-3} and v_{n-5} are not compatible with each other, but state v_{n-4} is compatible with each of them.

Each pair of the states that appear in the second column have the same color and the two states within each of those pairs are compatible only with themselves, that is, for each $i \in \{1, 3, 5, \dots, n-7\}$, v_i is compatible with v_{i+1} and only with v_{i+1} .

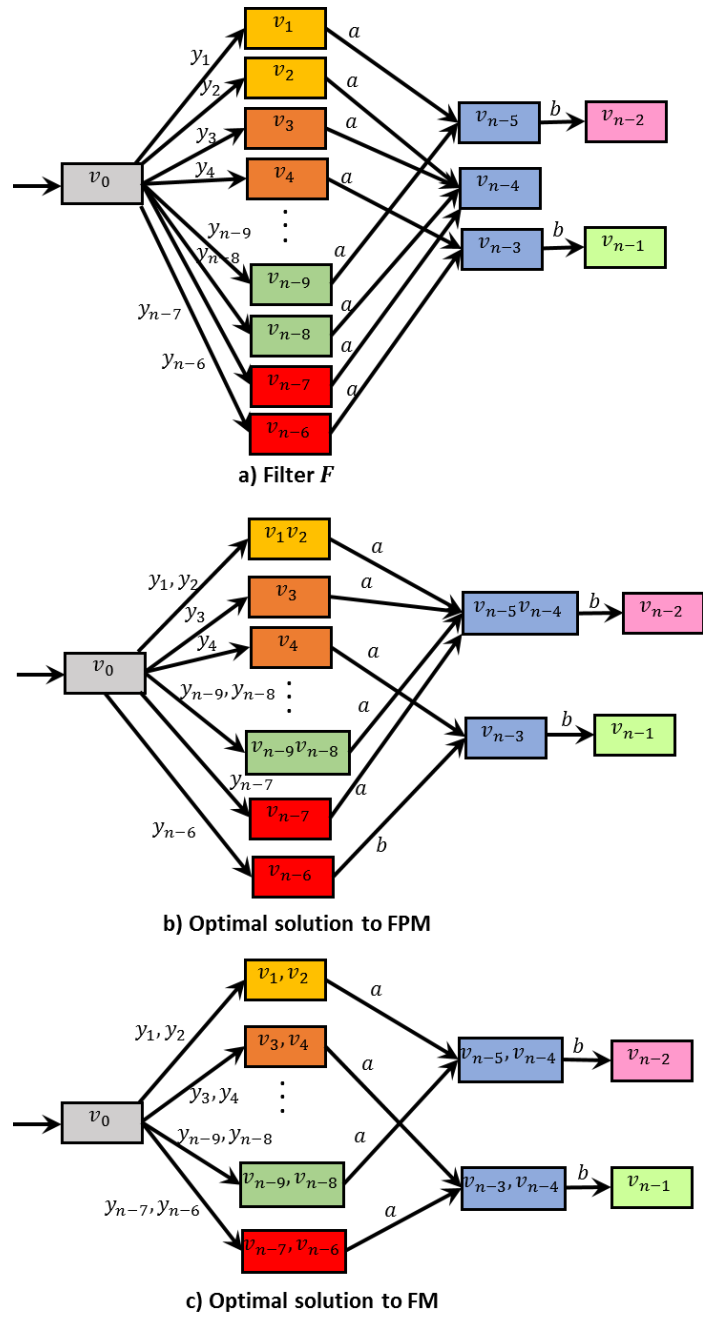


Figure 3.9: An illustration of the proof of Theorem 2. **a)** A sample filter F with n states. **b)** An optimal solution of FPM with input filter F . **c)** An optimal solution of FM with input filter F .

Half of those pairs have outgoing edges to state v_{n-4} and v_{n-5} , and the other half have outgoing edges to v_{n-3} and v_{n-5} . In an optimal solution to FPM, state v_{n-4} can be merged with either v_{n-3} or v_{n-5} . Part b of Figure 3.9 shows an optimal solution to FPM. In that solution, state v_{n-4} is merged with v_{n-5} , and as a result, of the pairs in the middle column, only those who have outgoing edges to v_{n-4} and v_{n-5} are merged. Accordingly, only half of the pairs in the middle column, or more precisely, $\frac{n-6}{4}$ states, are merged in the optimal solution to FPM. The FM with input filter F in this figure has only one optimal solution, which is shown in part c of this figure. Consider that since an optimal solution of FM is obtained by finding a covering of the state space rather than a partitioning of it, the states with color blue are split into two compatibility classes $\{v_{n-4}, v_{n-5}\}$ and $\{v_{n-4}, v_{n-3}\}$. State v_{n-4} is shared between the two classes. Because of this, all pairs in the middle column are merged, and thus, the optimal solution to FPM has $\frac{n-6}{4}$ states more than the optimal solution to FM.

Now consider the case where $n \geq 10$ but $n - 6$ is not divisible by 4. In this case, we can always choose integers n_1 and j such that $n = n_1 + j$, $n_1 \geq 10$, $n_1 - 6$ is divisible by 4, and $j \in \{1, 2, 3\}$. For example, if $n = 13$, then $n_1 = 10$ and $j = 3$. As another example, if $n = 28$, then $n_1 = 26$ and $j = 2$. For this case, we make a filter F_1 with n_1 states such that F_1 has n_1 states exactly in the form of filter F in Figure 3.9a and j extra states to each of which there is one and only one transition labeled y_{n_1-6+j} from state v_0 to that state. We opt that none of those states to have outgoing transitions. Additionally, we color those j extra states with j distinct colors that are not used to color other states. None of these extra j states are compatible with any other state. Accordingly, neither in the optimal solution to FM nor in an optimal solution to FPM, those states are merged. Accordingly, for each filter F_1 , the difference between the optimal solutions to FPM and FM is $\frac{n_1-6}{4}$. For this case, it holds that $\lfloor \frac{n-6}{4} \rfloor = \frac{n_1-6}{4}$, and as a result, the theorem is proved. \square

According to this theorem, a feasible solution to FPM, even an optimal one, does

not necessarily give a good solution for FM. An impact of this is that FM, which because of being an NP-hard problem we can hope to find in a reasonable amount of time only feasible solutions of it for large filters, requires to be treated by heuristic and greedy algorithms or other kind of techniques that are designed specifically for FM, and those who are proposed for FPM may not compute high-quality solutions for FM.

3.5 BISIMULATION AND FILTER REDUCTION

In this section, we show that the well-known notion of bisimulation does not always yield optimal solutions to FM and FPM.

In fact, one apparently reasonable hypothesis is that the notion of bisimulation may be useful for We show that although the bisimilarity relation is indeed a compatibility equivalence relation, and hence, can be used to compute feasible solutions to FM and FPM; it does not, in general, induce minimal filters. We begin by adapting the standard notion of bisimulation to filters.

Definition 9. Let $F_1 = (V_1, Y, C, \delta_1, c_1, v_0)$ and $F_2 = (V_2, Y, C, \delta_2, c_2, w_0)$ be two (not necessarily distinct) filters. A relation $R \subseteq V_1 \times V_2$ is said to be a *bisimulation relation* for (F_1, F_2) if for any $(v, w) \in R$:

1. $c_1(v) = c_2(w)$,
2. for any $y \in Y$, if $\delta_1(v, y) \neq \perp$, then $\delta_2(w, y) \neq \perp$ and $(\delta_1(v, y), \delta_2(w, y)) \in R$
3. for any $y \in Y$, if $\delta_2(w, y) \neq \perp$, then $\delta_1(v, y) \neq \perp$ and $(\delta_1(v, y), \delta_2(w, y)) \in R$

We say that state v in filter F_1 is *bisimilar* to state w in filter F_2 if there exists a bisimulation relation R for (F_1, F_2) such that $(v, w) \in R$ or equivalently if $v \sim_{(F_1, F_2)} w$, where relation $\sim_{(F_1, F_2)}$, which itself is a bisimulation relation for (F_1, F_2) , is the union

of all bisimulation relations for (F_1, F_2) . The notion of bisimulation can also be lifted to the filters themselves, according to which F_1 and F_2 are bisimilar, denoted by $F_1 \simeq F_2$, if there exists a bisimulation relation R for (F_1, F_2) such that $(v_0, w_0) \in R$.

We use bisimulation between two filters in the next section; in this section, we are concerned only about bisimulation relations that relate states within a single filter, that is, where in Definition 9, $F_1 = F_2 = F = (V, Y, C, \delta, c, v_0)$. Observe that any union of bisimulation relations for a filter is itself a bisimulation relation. The union of *all* bisimulation relations for F , denoted \sim_F , is called the *bisimilarity relation* for F . Recall filter F_3 , depicted in Figure 3.5. For this filter, $\sim_{F_3} = I_{F_3} \cup \{(v_2, v_3), (v_3, v_2), (v_4, v_5), (v_5, v_4)\}$.

Such bisimilarity relations are of interest in part because they are suitable for constructing quotient filters.

Lemma 7. The bisimilarity relation of every filter is both a compatibility relation and an equivalence relation.

Proof. It is well known that the bisimilarity is an equivalence relation (see, for example, Lemma 7.8 of Baier, Katoen, and Larsen [5] for a proof). Also, by Definition 9, any bisimulation relation—including the bisimilarity relation—is a compatibility relation in the sense of Definition 6. \square

Because the bisimilarity relation of a given filter represents, in a certain sense, a coarsest partitioning of the states into ‘mergeable’ subsets, intuition might suggest that a quotient with the bisimilarity relation might perhaps produce an optimally reduced filter, in the sense of the FM or the FPM problem. The next result debunks this misconception.

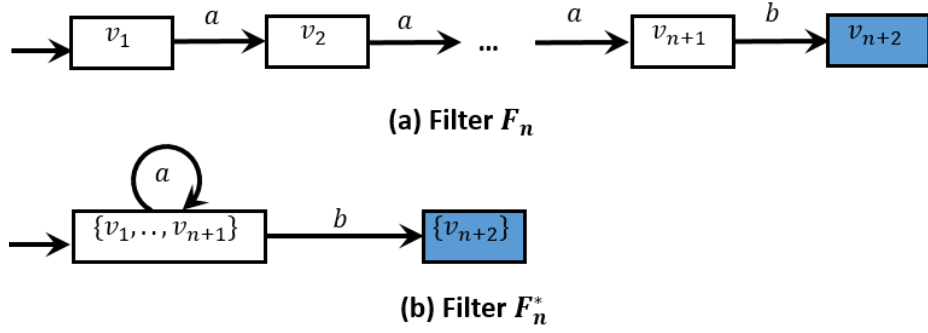


Figure 3.10: **a)** The construction of filter F_n , mentioned in Theorem 3. The quotient of this filter under \sim_{F_n} does not reduce its size. **b)** Filters F_n^* and F_n^{**} , which are identical, are respectively the optimal solutions to FM and FPM with input F_n . State v_{n+2} in filter F_n and state $\{v_{n+2}\}$ in filters F_n^* and F_n^{**} have color 2; all other states in both filters have color 1.

Theorem 3. For any integer $n \geq 1$, there exists a filter F_n with $n + 2$ states, such that F_n / \sim_{F_n} is larger than the optimal solution F_n^* to FM by n states. The same filter F_n / \sim_{F_n} is larger than the optimal solution F_n^{**} to FPM by n states.

Proof. For a given n , we construct a filter F_n with $n + 2$ states, for which F_n / \sim_{F_n} also has $n + 2$ states. Figure 3.10a shows the construction. In particular, note that for any pair of distinct states (v, w) , we have $v \not\sim_{F_n} w$; this is because if $v \sim_{F_n} w$, then they must have the same color, meaning that there must exist $1 \leq i \neq j \leq n + 1$ such that $v = v_i$ and $w = v_j$, and if this is the case then by the definition of bisimulation relation, we must have that $v_{i+1} \sim_{F_n} v_{j+1}$, $v_{i+2} \sim_{F_n} v_{j+2}$, ..., and ultimately $v_{i+k} \sim_{F_n} v_{n+2}$, which is a contradiction. Therefore $\sim_{F_n} = I_{F_n}$, and F_n / \sim_{F_n} is structurally identical to F_n — no two states will be merged. In contrast, for any n , each of the optimally reduced F_n^* has exactly two states, as shown in Figure 3.10b. \square

In particular, Theorem 3 implies that bisimulation-quotienting does not always induce optimal, and in fact, that the difference in size between the optimally reduced and the bisimilarity-quotient filter cannot be bounded.

Note that by Definition 9, for two states to be bisimilar, not only they must agree on their outputs but also for each observation, either both of them must have an outgoing transition labeled with that observation or none of them must have an outgoing transition for that observation. Accordingly, bisimulation imposes a strong condition for two states to be merged, and this, results to the fact that bisimulation does not yield enough reduction for some filters. Yet, we can still use bisimulation to make an extent of reduction, but for real-world applications in robotics, we may not be satisfied with the amount of reduction we achieve with bisimulation. Both the examples in this theorem and Figure 1.2 show that for some filters, bisimulation does not reduce the size of the filter at all while the minimal filter is much smaller than the original filter. Intuitively, if the underlying system or the problem of interest does not enjoy a big deal of symmetry and indistinguishability, then bisimulation does not offer much reduction. To illustrate these notions of symmetry and indistinguishability, consider an environment similar to that in Figure 1.2 but in which the four landmarks are on the four corners of a square and the robot could not distinguish between some landmarks. It is usually for the filters of this kind of situations happens that some states become bisimilar, and thus, bisimulation can help to reduce the size of the filter. These notions also arise in problems similar to that in Figure 3.4, where the shape of the environment is symmetric and in which when a robot crosses a beam, the system knows which beam sensor it was except that it cannot distinguish between some beams.

3.6 STRONG FILTER MINIMIZATION

In this section, we introduce a variant of filter minimization problem for which the bisimilarity relation always produces an optimal solution.

Section 3.5 showed that, although quotient with the bisimilarity relation produces

an equivalent filter, that filter may not necessarily be minimal. In this section, we provide some insight into why that happens, by showing that this kind of bisimilarity quotient instead solves a variant of the filter minimization problem, in which the language of the reduced filter must be identical to the language of the original filter, rather than merely a superset of it. Specifically, this section shows that bisimilarity-quotienting solves the following problem.

Problem: Strong Filter Minimization (SFM)

Input: A filter F .

Output: A filter F^* such that $F \xrightarrow{L(F)} F^*$, $L(F) = L(F^*)$, and the number of states in F^* is minimal.

Now we can state the main result of this section.

Theorem 4. For any filter F , the bisimilarity quotient F / \sim_F is a solution to the SFM problem for F .

Proof. It suffices to prove only that SFM is equivalent to the problem of finding a minimal filter F^* such that $F \simeq F^*$. The rest is due to the well-known fact about bisimulation minimization, that the quotient of a structure under its bisimilarity relation is a smallest structure that is bisimilar to the original structure. For a proof of this result in the context of transition systems, see [5]. The only difference here is that filters are concerned with observation sequences with finite-length rather than observation sequences with infinite-length.

Therefore, we need to prove that $F \simeq F^*$ if and only if $F \xrightarrow{L(F)} F^*$ and $L(F) = L(F^*)$. Let $F = (V, Y, C, \delta, c, v_0)$ and $F^* = (V', Y, C, \delta', c', v'_0)$. For the direction \Rightarrow , assume by contradiction that $F \simeq F^*$, that is, $v_0 \sim_{(F, F^*)} v'_0$, but either $L(F) \neq L(F^*)$ or for an observation sequence $s \in L(F) \cap L(F^*)$, $c(\delta^*(v_0, s)) \neq c'(\delta'^*(v'_0, s))$.

Figure 3.11 shows the proof.

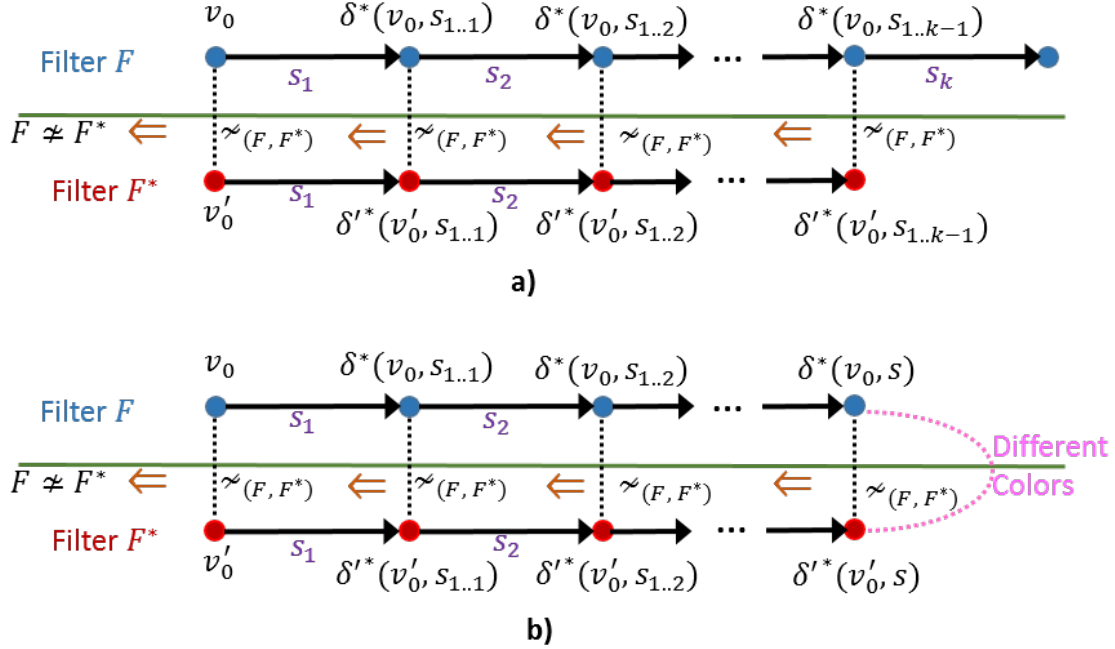


Figure 3.11: Part of the proof of Theorem 4, which shows by contradiction that if $F \simeq F^*$ then $F \stackrel{L(F)}{=} F^*$ and $L(F) = L(F^*)$. **a)** The contradiction assumption that $L(F) \neq L(F^*)$. Without loss of generality, we assume s to be an observation sequence in $L(F)$ but not in $L(F^*)$. Integer k is assumed to be the smallest integer such that $s_{1..k} \notin L(F^*)$ but $s_{1..k-1} \in L(F^*)$. Given that state $\delta^*(v_0, s_{1..k-1})$ has a transition for observation s_k but state $\delta'^*(v'_0, s_{1..k-1})$ does not have a transition for that observation, by the second condition of Definition 9, those two states are not bisimilar, i.e., $\delta^*(v_0, s_{1..k-1}) \not\sim_{(F, F^*)} \delta'^*(v'_0, s_{1..k-1})$, and this, via a series of implications ($\delta^*(v_0, s_{1..k-2}) \not\sim_{(F, F^*)} \delta'^*(v'_0, s_{1..k-2})$, $\delta^*(v_0, s_{1..k-3}) \not\sim_{(F, F^*)} \delta'^*(v'_0, s_{1..k-3})$, ...) implies that v_0 is not bisimilar with v'_0 , meaning that F and F^* are not bisimilar, which is a contradiction. A similar argument applies for when we assume that there is an observation sequence s such that $s \in L(F^*)$ but $s \notin L(F)$. **b)** The contradiction assumption that for an observation sequence $s \in L(F) \cap L(F^*)$, $c(\delta^*(v_0, s)) \neq c(\delta'^*(v'_0, s))$. Given that those two states $\delta^*(v_0, s)$ and $\delta'^*(v'_0, s)$ do not have the same color, by the first condition of Definition 9, they are not bisimilar, i.e., $\delta^*(v_0, s) \not\sim_{(F, F^*)} \delta'^*(v'_0, s)$ and this, via a series of implications ($\delta^*(v_0, s_{1..|s|-1}) \not\sim_{(F, F^*)} \delta'^*(v'_0, s_{1..|s|-1})$, $\delta^*(v_0, s_{1..|s|-2}) \not\sim_{(F, F^*)} \delta'^*(v'_0, s_{1..|s|-2})$, ...), implies that $v_0 \not\sim_{(F, F^*)} v'_0$, meaning that F is not bisimilar to F^* , which is a contradiction.

For the former case, assume, without loss of generality, that s be an observation sequence such that $s \in L(F)$ but $s \notin L(F^*)$. Let integer k , where $1 \leq k < |s|$, be the smallest index such that $\delta'^*(v'_0, s_{1..k}) = \perp$ and let $y = s_{k..k}$. Clearly, $\delta(\delta^*(v_0, s_{1..k-1}), y) \neq \perp$ but $\delta'(\delta'^*(v'_0, s_{1..k-1}), y) = \perp$. This by the second condition

of Definition 9 implies that $\delta^*(v_0, s_{1..k-1}) \not\sim_{(F, F^*)} \delta'^*(v'_0, s_{1..k-1})$, which, in turn, by the same condition of that definition implies that $\delta_1^*(v_0, s_{1..k-2}) \not\sim_{(F, F^*)} \delta_2^*(v'_0, s_{1..k-2})$. Applying the same condition $k-3$ more times implies that $v_0 \not\sim_{(F, F^*)} v'_0$, which means that $F \not\sim F^*$, another contradiction.

For the later case, given that $c(\delta^*(v_0, s)) \neq c'(\delta'^*(v'_0, s))$, by the first condition of Definition 9 it holds that $\delta^*(v_0, s) \not\sim_{(F, F^*)} \delta'^*(v'_0, s)$, which by the second condition of the same definition implies that $\delta^*(v_0, s_{1..|s|-1}) \not\sim_{(F, F^*)} \delta'^*(v'_0, s_{1..|s|-1})$, which in turn implies that $\delta^*(v_0, s_{1..|s|-2}) \not\sim_{(F, F^*)} \delta'^*(v'_0, s_{1..|s|-2})$, $\delta^*(v_0, s_{1..|s|-3}) \not\sim_{(F, F^*)} \delta'^*(v'_0, s_{1..|s|-3})$, ..., and finally $v_0 \not\sim_{(F, F^*)} v'_0$, which means that $F \not\sim F^*$ and is a contradiction.

For the direction \Leftarrow , we need to prove that if $F \stackrel{L(F)}{=} F^*$ and $L(F) = L(F^*)$, then $F \simeq F^*$. Consider that from $L(F) = L(F^*)$ and $F \stackrel{L(F)}{=} F^*$, it is implied that for all $s \in L(F)$, $c(\delta^*(v_0, s)) = c'(\delta'^*(v'_0, s))$. Accordingly, we consider the relation $R = \bigcup_{s \in L(F)} \{(\delta^*(v_0, s), \delta'^*(v'_0, s))\}$. Each tuple of states $\delta^*(v_0, s)$ and $\delta'^*(v'_0, s)$ related by this relation have the same color, thereby satisfying the first condition of Definition 9. Also because $F \stackrel{L(F)}{=} F^*$ and $L(F) = L(F^*)$, for each observation y , if one of these two states has an outgoing transition labeled y , then the other one also has an outgoing transition labeled y . Then the states $\delta(\delta^*(v_0, s), y)$ and $\delta'(\delta'^*(v'_0, s), y)$ are likewise related by R , satisfying the second and the third conditions of Definition 9. Hence, R is a bisimulation relation for (F, F^*) in sense of Definition 9. Also, given that $(v_0, v'_0) \in R$, it holds that $v_0 \sim_{(F, F^*)} v'_0$, which means that $F \simeq F^*$. \square

Consider also that since the bisimilarity relation is a compatibility equivalence relation, by Lemma 3 the optimal solution to SFM always partitions the state space of the original filter, that is, $F \doteq F / \sim_F$.

Corollary 1. SFM can be solved in polynomial time.

Proof. Beyond Theorem 4, we need only to show that given a filter $F = (V, Y, C, \delta, c, v_0)$ both (a) the bisimilarity relation \sim_F and (b) the quotient of a filter and a relation, can be computed in polynomial time.

A simple efficient algorithm for constructing the bisimilarity relation starts with assigning the set $\{(v, w) \in V \times V \mid c(v) = c(w) \wedge \forall y \in Y, (\delta(v, y) = \delta(w, y) = \perp \vee c(\delta(v, y)) = c(\delta(w, y)))\}$ as the initial value to a variable R . Then, in each iteration of a loop, all members of R that fail to satisfy all three conditions of Definition 9 are removed from R . This loop continues until no additional members of R can be removed; at that time, we have $R = \sim_F$. Clearly, the time complexity of this algorithm is $O(|V|^4 \times |Y|)$. This relation has at most $|V|^2$ members, hence, the filter F / \sim_F is constructed in $O(|V|^4 \times |Y|)$ time. \square

As an example of this theorem, consider again filter F_3 from in Figure 3.5. The quotient of this filter under \sim_{F_3} is filter F_4 , depicted in the same figure. The language of F_3 is equal to the language of F_4 . Filter F_5 , depicted in the same figure, represents the smallest filter who is equivalent to F_3 with respect to the language of F_3 . In this case, we have $L(F_3) \subset L(F_5)$.

Knowing now that making the quotient of a filter under bisimilarity relation does not always optimally reduce the size of that filter, in the next section, we are interested in realizing the connection between filter reduction and a weaker notion of bisimulation called simulation.

3.7 SIMULATION AND FILTER REDUCTION

In this section, we prove that the FM problem is equivalent to the problem of finding a minimal filter that simulates the original filter.

To do so, we introduce a definition of simulation relation between states of two filters, adapting the standard notion for transition systems.

Definition 10. Let $F_1 = (V_1, Y, C, \delta_1, c_1, v_0)$ and $F_2 = (V_2, Y, C, \delta_2, c_2, w_0)$ be two filters. A relation $R \subseteq V_1 \times V_2$ is said to be a *simulation relation* for (F_1, F_2) if for any $(v, w) \in R$:

1. $c_1(v) = c_2(w)$, and
2. for all $y \in Y$, if $\delta_1(v, y) \neq \perp$, then $\delta_2(w, y) \neq \perp$ and $(\delta_1(v, y), \delta_2(w, y)) \in R$

We say that state w *simulates* state v if there exists a simulation relation R for (F_1, F_2) such that $(v, w) \in R$. The union of all simulation relations for (F_1, F_2) , which is called the *similarity relation* for (F_1, F_2) and denoted $\prec_{(F_1, F_2)}$, is itself a simulation relation. It also can be computed in time polynomial to the size of F_1 and F_2 .

Just as with bisimulation, the notion of simulation, which is defined thus far between states of two filters, can be lifted to filters themselves. We write $F_1 \preceq F_2$, indicating that filter F_2 simulates filter F_1 , if there exists a simulation relation R for (F_1, F_2) such that $(v_0, w_0) \in R$. Equivalently, $F_1 \preceq F_2$ if $v_0 \prec_{(F_1, F_2)} w_0$.

To illustrate this notion of simulation, consider filters F_6 and F_7 , depicted in Figure 3.12. Some simulation relations for (F_6, F_7) are:

- $R_1 = \{(v_1, w_0)\}$,
- $R_2 = \{(v_1, w_0), (v_3, w_3)\}$,
- $R_3 = \{(v_1, w_0), (v_3, w_3), (v_2, w_2), (v_5, w_4), (v_4, w_4)\}$, and
- $R_4 = R_3 \cup \{(v_0, w_0)\}$.

Note that $\prec_{(F_6, F_7)} = R_4$.

The following theorem establishes a strong connection between simulation (in the sense of Definition 10) and filter equivalence (in the sense of Definition 4).

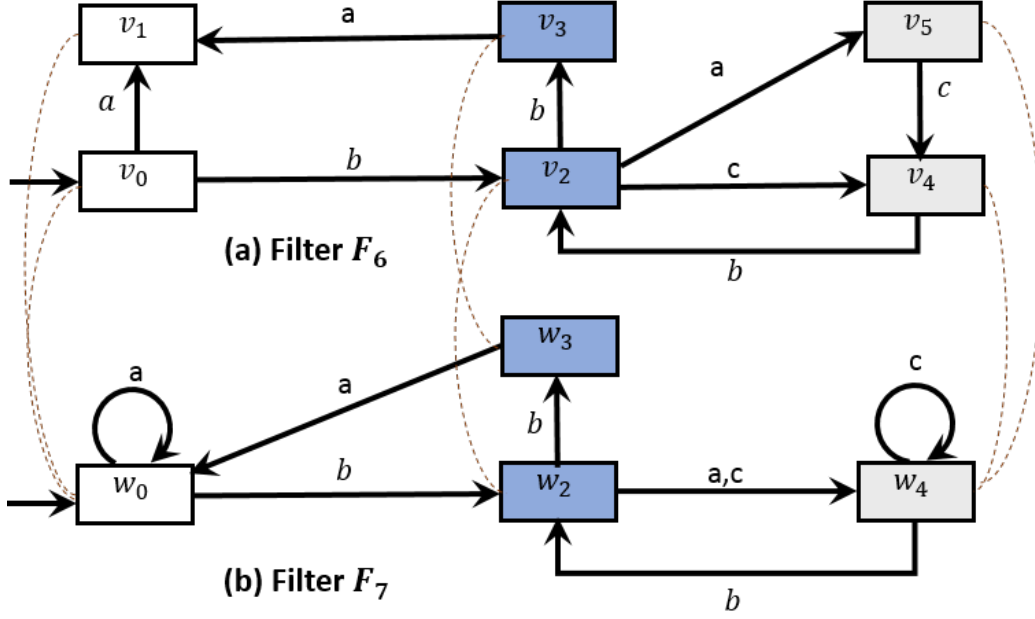


Figure 3.12: a) A sample filter F_6 b) An example of a filter that simulates F_6 . For both filters, states in the left column have color 1, states in the middle column have color 2, and states in the right column have color 3. The similarity relation for (F_6, F_7) has been depicted by dashed lines.

Theorem 5. For any filters F_1 and F_2 , $F_1 \preceq F_2$ if and only if $F_1 \xrightarrow{L(F_1)} F_2$.

Proof. We first prove that if $F_1 \preceq F_2$ then $F_1 \xrightarrow{L(F_1)} F_2$. Assume for a contradiction that the statement $F_1 \preceq F_2$ holds but the statement $F_1 \xrightarrow{L(F_1)} F_2$ does not hold. By Definition 4, the statement $F_1 \xrightarrow{L(F_1)} F_2$ does not hold in two cases: either (1) where $L(F_1) \not\subseteq L(F_2)$ or (2) where there exists an observation sequence $s \in L(F_1) \cap L(F_2)$ such that $c_1(\delta_1^*(v_0, s)) \neq c_2(\delta_2^*(w_0, s))$.

For the former case, assume that s be an observation sequence such that $s \in L(F_1)$ but $s \notin L(F_2)$. Let integer k , where $1 \leq k < |s|$, be the smallest index such that $\delta_2^*(w_0, s_{1..k}) = \perp$ and let $y = s_{k..k}$. Clearly, $\delta_1(\delta_1^*(v_0, s_{1..k-1}), y) \neq \perp$ but $\delta_2(\delta_2^*(w_0, s_{1..k-1}), y) = \perp$. This by the second condition of Definition 10 implies that $\delta_1^*(v_0, s_{1..k-1}) \not\prec_{(F_1, F_2)} \delta_2^*(w_0, s_{1..k-1})$. This by the same condition implies that

$\delta_1^*(v_0, s_{1..k-2}) \not\prec_{(F_1, F_2)} \delta_2^*(w_0, s_{1..k-2})$. Again by applying the same condition $k - 3$ more times, we conclude that $v_0 \not\prec_{(F_1, F_2)} w_0$. And, this contradicts that $F_1 \preceq F_2$.

For the later case, given that $c_1(\delta_1^*(v_0, s)) \neq c_2(\delta_2^*(w_0, s))$, by the first condition of Definition 10 it holds that $\delta_1^*(v_0, s) \not\prec_{(F_1, F_2)} \delta_2^*(w_0, s)$. This by the second condition of the same definition implies that $\delta_1^*(v_0, s_{1..|s|-1}) \not\prec_{(F_1, F_2)} \delta_2^*(w_0, s_{1..|s|-1})$. Again by applying the second condition, we concludes that $\delta_1^*(v_0, s_{1..|s|-2}) \not\prec_{(F_1, F_2)} \delta_2^*(w_0, s_{1..|s|-2})$, $\delta_1^*(v_0, s_{1..|s|-3}) \not\prec_{(F_1, F_2)} \delta_2^*(w_0, s_{1..|s|-3})$, ..., and finally $v_0 \not\prec_{(F_1, F_2)} w_0$, which contradicts that $F_1 \preceq F_2$.

We now prove that if $F_1 \stackrel{L(F_1)}{=} F_2$ then $F_1 \preceq F_2$. For the sake of contradiction assume that $F_1 \stackrel{L(F_1)}{=} F_2$ but $F_1 \not\prec_{(F_1, F_2)} F_2$, that is, $v_0 \not\prec_{(F_1, F_2)} w_0$. It is easy to observe that since $v_0 \not\prec_{(F_1, F_2)} w_0$, there must exist an observation sequence $s \in L(v_0) \cap L(w_0)$ such that $\delta_1^*(v_0, s) \not\prec_{(F_1, F_2)} \delta_2^*(w_0, s)$. This by Definition 10 means that either (1) $c_1(\delta_1^*(v_0, s)) \neq c_2(\delta_2^*(w_0, s))$ or (2) for an observation $y \in Y$, $\delta_1(\delta_1^*(v_0, s), y) \neq \perp$ while $\delta_2(\delta_2^*(w_0, s), y) = \perp$, which means that $sy \in L(F_1)$ while $sy \notin L(F_2)$. If any of these two cases holds, then it means that the statement $F_1 \stackrel{L(F_1)}{=} F_2$ does not hold, which is a contradiction. \square

To see an impact of Theorem 5, we first consider the following .

Problem:

Input: A filter F .

Output: A filter F^* such that $F \preceq F^*$ and the number of states in F^* is minimum.

Problem: Minimal partitioning simulation filter (MPSF)

Input: A filter F .

Output: A filter F^* such that $F \preceq F^*$, $F \doteq F^*$ and the number of states in F^* is minimum.

Next we establish the following result.

Corollary 2. Given a filter F , any optimal (feasible) solution to FM with input F is an optimal (feasible) solution to MSF with the same input, and vice versa. Also, any optimal (feasible) solution to FPM with input F is an optimal (feasible) solution to MPSF with the same input, and vice versa.

Proof. The result follows from Theorem 5 and the of the filter minimization problem and the filter partitioning minimization problem. \square

As a result, the filter minimization problem can be thought of finding a minimal filter simulating the original filter.

In the context of transition systems, the problem of computing a minimal transition system that simulates a given transition system is a trivial problem since transition systems are, in general, “nondeterministic” and, therefore, the minimal solution has always $|C|$ states—one state for each “color”—between any pair of which there is a transition labeled y for any “nondeterministic” (action) y . That approach, however, cannot be used here since combinatorial filters are “deterministic” while the constructed structure (the one with $|C|$ states) is “nondeterministic”; in other words, the final structure does not fit into Definition 1.

3.8 THE UNION OF ALL COMPATIBILITY RELATIONS AND THE MERGEABILITY RELATION

In this section, we introduce two special compatibility relations which are used to identify a taxonomy of filters that are minimizable in polynomial time.

By the discussions of the Section 3.3, to solve FPM for given filter F —one without any unreachable states, of course— one can make the quotient filter under some compatibility equivalence relation, and to solve FM for F , one need to make a quotient filter under a closed covering of the state space of F . Section 3.5 proved that

Algorithm 1: UNIONOFALLCOMPRELATIONS

Input : A filter $F = (V, Y, C, \delta, c, v_0)$

Output: The union of all compatibility relations for (F_1, F_2)

```
1  $R \leftarrow \emptyset$ 
2 forall  $(v, w) \in V \times V$  do
3   if  $c(v) \neq c(w)$  then
4     continue
5    $add \leftarrow true$ 
6   forall  $y \in Y$  do
7     if  $\delta(v, y) \neq \perp$  and  $\delta(w, y) \neq \perp$  then
8       if  $c(\delta(v, y)) \neq c(\delta(w, y))$  then
9          $add \leftarrow false$ 
10  if  $add = true$  then
11     $R \leftarrow R \cup \{(v, w)\}$ 
12  $updated \leftarrow true$ 
13 while  $updated = true$  do
14    $updated \leftarrow false$ 
15   forall  $(v, w) \in R$  do
16     forall  $y \in Y$  do
17       if  $\delta(v, y) \neq \perp$  and  $\delta(w, y) \neq \perp$  then
18         if  $(\delta(v, y), \delta(w, y)) \notin R$  then
19            $R = R / \{(v, w)\}$ 
20            $updated \leftarrow true$ 
21 return  $R$ 
```

the bisimilarity relation \sim_F is not always the appropriate relation for this job, and Section 3.7 showed that any minimal filter equivalent to the original filter is always a minimal filter that simulates the original filter.

Another intuitive possibility would be to use the union of all compatibility relations, analogous to the definition of the bisimilarity relation as the union of all bisimulation relations. As an example, this relation for filter F_3 depicted in Figure 3.5, is $I_{F_3} \cup \{(v_0, v_1), (v_1, v_0), (v_2, v_3), (v_3, v_2), (v_4, v_5), (v_5, v_4)\}$. For a given filter F , we write λ_F to denote this the union of all compatibility relations for F . Trivially, λ_F is itself a compatibility relation for F .

In addition, we can compute λ_F in time polynomial in the size of F . See Algorithm 1 for a simple approach to doing so. The intuition is to begin with a relation

containing state pairs that are compatible for observation strings of length at most one, and then to iteratively delete state pairs that violate Definition 6 for successively longer strings. The time complexity of this algorithm is $O(|V|^4|Y|)$, where V and Y are, respectively, the state space and the observation space of the input filter.

The next lemma shows that, unfortunately, λ_F may not be suitable for solving , because for some filters it is not an equivalence relation. (Recall Definition 7, under which quotient filters are well-defined only for compatibility equivalence relations.)

Lemma 8. For any filter $F = (V, Y, C, \delta, c, v)$, the relation λ_F is reflexive and symmetric. However, there exist filters F for which λ_F is not transitive.

Proof. For the first claim, consider that in sense of Definition 6, the identity relation $I_F = \{(v, v) \mid v \in V\}$ is a compatibility relation for F . By definition of λ_F , it is a superset of I_F , and therefore λ_F is reflexive. To prove that λ_F is symmetric, one need to show that if $v \lambda_F w$, then $w \lambda_F v$. Suppose that $v \lambda_F w$. This means that there exists a compatibility relation R for F such that $(v, w) \in R$. By the symmetry of conditions (1) and (2) of Definition 6 with respect to v and w , if R is a compatibility relation for F , then so is R^{-1} . The relation R^{-1} contains (w, v) , and so does λ_F given the definition of λ_F .

For the second claim, to observe that λ_F may not be transitive, let F be the filter depicted in Figure 3.13. For this filter, we have $\lambda_F = I_F \cup \{(v_1, v_2), (v_2, v_1), (v_2, v_3), (v_3, v_2)\}$. This relation is not transitive since $v_1 \lambda_F v_2$ and $v_2 \lambda_F v_3$ hold but $v_1 \lambda_F v_3$ does not. \square

In an important sense, Lemma 8 should not be a surprise. Since the is NP-hard [100, 125], and F/λ_F can be computed in polynomial time, if Lemma 8 were false, that would imply that $P = NP$.

However, any filter F with state space V for which λ_F is indeed an equivalence relation, then F/λ_F is guaranteed to be an optimal solution to FPM with input F ,

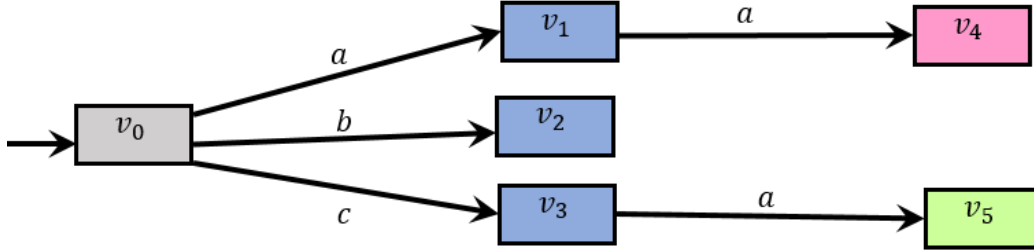


Figure 3.13: A filter for which the union of all compatibility relations is not an equivalence relation. State v_0 has color 1, states in the middle column have color 2, state v_4 has color 3, and state v_5 has color 4.

and because in this case λ_F is a minimal closed covering of V , we conclude that F/λ_F is an optimal solution for FM too. Moreover, given any filter F , it takes polynomial time to check whether λ_F is an equivalence relation or not. This implies that solving FM and FPM for any filter for which λ_F is an equivalence relation takes polynomial time in size of F . This fact gives a roadmap to recognize some classes of filters for which the filter minimization problem and the filter partitioning minimization problem are in P , specifically by looking for classes of filters for which the union of all compatibility relations can be proven to be an equivalence relation. Section 3.9 uses this fact to identify a several of these kind of classes.

A question here is does there exist any other efficiently computable relation other than the union of all compatibility relations, that if it has a special property, then FM or FPM for the given input filter is solvable in polynomial time.

In the sequel, we answer that question, but first consider the following definition.

Definition 11. Let v and w be two states in a filter F . We say that v is *mergeable* with w if there exists a compatibility equivalence relation for F such that $(v, w) \in R$, and we say that v is *not mergeable* with w otherwise.

The idea of this definition is that by Definition 7, making quotient filters is well-defined only under compatibility equivalence relations. Accordingly, if v and w cannot

be related by any compatibility equivalence relation F , then they are not collapsed into a single state in any correctly reduced filter.

Clearly, any pair of mergeable states are compatible, but the reverse does not necessarily hold. For an example of a filter in which a pair of compatible states are not mergeable, consider Figure 3.14. The union of all compatibility equivalence relations for the filter F_8 in this figure is $\lambda_{F_8} = I_{F_8} \cup \{(2, 3), (3, 2), (0, 1), (1, 0), (3, 4), (4, 3)\}$, which has been depicted in part (b) of that figure by the graph CG_{F_8} . Notice that the mentioned graph does not need to have loops and it does not need to be a directed graph given that we implicitly know that the union of all compatibility relations for any filter is reflexive and symmetric. Although states 2 and 3 in this filter are compatible, we argue that they are not mergeable. To do so, we need to show that no compatibility relation containing $(2, 3)$ is an equivalence relation. Let R be any compatibility relation that contains $(2, 3)$. Due to the assumption that R is a compatibility relation and contains $(2, 3)$, by the second condition of Definition 6, relation R must contain $(0, 1)$. Given the same condition and that R contains $(0, 1)$, it must contain $(3, 4)$ too. But, relation R cannot be transitive since it has $(2, 3)$ and $(3, 4)$ but cannot contain $(2, 4)$ due to that $2 \not\sim_{F_8} 4$. Thus, R is not an equivalence relation.

This example shows that in the quest for an optimal solution to via a compatibility equivalence relation with minimum number of partitions we can immediately throw away unmergeable pairs since we know that such pairs cannot be related by any compatibility equivalence relation. As a result, to find a compatibility equivalence relation with minimum number of partitions, we can search among subsets of a relation stronger than λ_F . That relation is defined as follows:

Definition 12. The *mergeability relation* for filter F , denoted \bowtie_F , is the union of all compatibility equivalence relations for F .

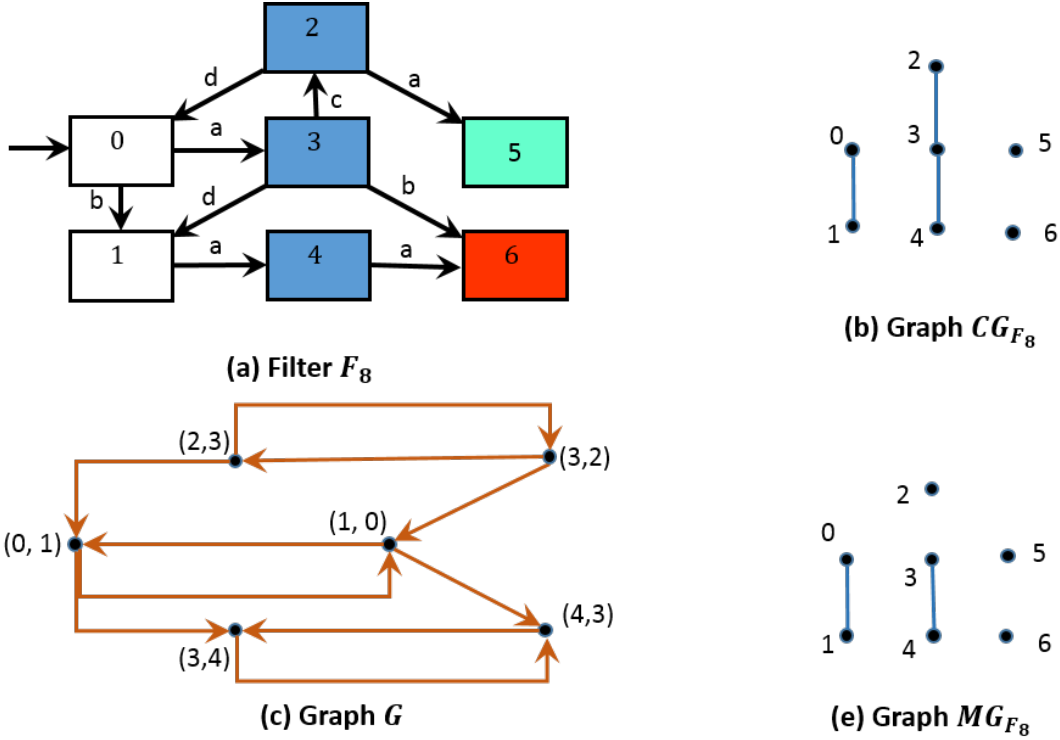


Figure 3.14: **a)** A filter for which the union of all compatibility relations does not coincide with its mergeability relation **b)** The graph of the union of all compatibility relations for filter F_8 **c)** The helper graph created for filter F_8 by Algorithm 2. **d)** The graph of the mergeability relation for filter F_8 . In this example, states 2 and 3 are compatible, but they are not mergeable. States 0 and 1 have color white. States in the middle column have color blue. State 5 has color light-green, and state 6 has color red.

Equivalently, relation \bowtie_F contains all state pairs (v, w) such that v is mergeable with w . Given this, v is mergeable with w in the sense of Definition 11 if and only if $v \bowtie_F w$.

A point worth mentioning about Definition 12 is that \bowtie_F is a superset of what we are looking for—a compatibility equivalence relation with minimum number of partitions—to solve FPM. Moreover, relation \bowtie_F is a compatibility relation given that it is the union of some compatibility relations. Unfortunately, for some filters, it is not an equivalence relation, and hence, may not be always used for making quotient filters.

Lemma 9. For any given filter F , relation \bowtie_F is reflexive and symmetric. However, there exist filters F for which \bowtie_F is not transitive.

Proof. For reflexivity, observe that for any filter F , relation $I_F = \{(v, v) \mid v \in V\}$ is an equivalence relation and is also a compatibility relation for F in the sense of Definition 6. Thus, by definition of \bowtie_F , it holds that $I_F \subseteq \bowtie_F$. For symmetricity, observe that by the definition of \bowtie_F , for any v and w such that $v \bowtie_F w$, there exists a compatibility equivalence relation R for F such that $(v, w) \in R$. Given that R is an equivalence relation, we have that $(w, v) \in R$. Now, since $R \subseteq \bowtie_F$, it holds that $w \bowtie_F v$.

With respect to the second claim, observe that the mergeability relation of the filter F in Figure 3.13 is $\bowtie_F = I_F \cup \{(v_1, v_2), (v_2, v_1), (v_2, v_3), (v_3, v_2)\}$, which is clearly not transitive. \square

An important point about the mergeability relation is that if for a filter F the relation \bowtie_F is an equivalence relation, then F / \bowtie_F is a minimal filter that is equivalent to F and also partitions the state space of F . This is due to the fact that in this case, the relation \bowtie_F is the coarsest compatibility equivalence relation for F . More importantly, there exist filters F for which λ_F is not an equivalence relation while \bowtie_F is an equivalence relation. An example of this kind of filters is filter F_8 , depicted in Figure 3.14.

A question remaining here is whether we can compute the mergeability relation, which is the union of all compatibility equivalence relations, in time polynomial in the size of the filter. The question is relevant because if the answer is yes, then is solvable in polynomial time for the class of filters F for which \bowtie_F is an equivalence relation. Note that an algorithm that constructs the mergeability relation by constructing and unioning all compatibility equivalence relations seems likely to require superpolynomial time, since otherwise one could solve in polynomial time merely by

making the quotient operation under one of those constructed compatibility equivalence relations—one with the minimum number of equivalence classes.

We answer this question in positive by providing Algorithm 2. This algorithm computes the mergeability relation by removing all unmergeable pairs from λ_F , i.e., it sets $\bowtie_F = \lambda_F - N_F$, where N_F is the set all unmergeable pairs that are compatible.

To compute N_F , we check any pair of compatible states (v, w) —only those in which $v \neq w$, of course—to see if they can be related by a compatibility equivalence relation for F or not. Specifically, we try to construct a compatibility equivalence relation with minimum number of members that contains (v, w) . If we fail in constructing such a relation, then it means that $v \not\bowtie_F w$, and thus, we put (v, w) in N_F . Lines 2-12 of this algorithm construct as a helper, a directed graph $G = (V', E')$, used for the construction of those compatibility equivalence relations. Each vertex of this graph is a pair of states $(v, w) \in V^2$ such that $v \lambda_F w$ and $v \neq w$. Each edge $((v, w), (x, z))$ of this graph indicates that any relation R containing (v, w) needs to contain (x, z) too in order to be a symmetric compatibility relation. Lines 6-8 of the algorithm add those edges that enforce the second condition of compatibility relation (See Definition 6). Lines 9-11 add edges enforcing the symmetricity of the constructed relation. The reader can check why for no $v \in V$ we need to add a vertex (v, v) to that graph, and that why the enforcement of condition 2 of Definition 6 is made only between nodes (v, w) and (x, z) where $(v, w) \neq (x, z)$.

To illustrate this kind of graph, see filter F_8 in Figure 3.14, the helper graph for which has been drawn in part c of the same figure.

Lines 17-24 of this algorithm use this graph to check for each pair of distinct compatible states (v, w) if v is mergeable with w or not. This is done by constructing relation $R_{v,w}$, whose initial value is the union set of I and $DFS(G, (v, w))$, where the procedure call $DFS(G, (v, w))$ performs the standard depth first search algorithm and returns as a relation, the union of all vertices in G that are reachable from vertex

Algorithm 2: MERGEABILITYRELATION

Input : A filter $F = (V, Y, C, \delta, c, v_0)$
Output: The mergeability relation for F

```
1  $\lambda \leftarrow \text{UNIONOFALLCOMPRELATIONS}(F)$ 
2  $V' \leftarrow \emptyset, E' \leftarrow \emptyset$ 
3 forall  $(v, w) \in \lambda$  do
4   | if  $v \neq w$  then
5   |   |  $V' \leftarrow V' \cup \{(v, w)\}$ 
6 forall distinct vertices  $(v, w), (x, z) \in V'$  do
7   | if  $\delta(v, y) = x$  and  $\delta(w, y) = z$  for a  $y \in Y$  then
8   |   |  $E' \leftarrow E' \cup \{(v, w), (x, z)\}$ 
9 forall  $(v, w) \in V'$  do
10  | if  $((v, w), (w, v)) \notin E'$  then
11  |   |  $E' \leftarrow E' \cup \{(v, w), (w, v)\}$ 
12 Create a helper graph  $G := (V', E')$ 
13  $I \leftarrow \emptyset$ 
14 forall  $v \in V$  do
15  |  $I \leftarrow I \cup \{(v, v)\}$ 
16  $N \leftarrow \emptyset$ 
17 forall  $(v, w) \in V'$  do
18  |  $R_{v,w} \leftarrow \text{DFS}(G, (v, w)) \cup I$ 
19  | while  $R_{v,w}$  contains some tuples  $(a, b)$  and  $(b, c)$  such that  $(a, c) \notin R_{v,w}$ 
20  |   | do
21  |     | if  $(a, c) \notin \lambda$  then
22  |       |  $N \leftarrow N \cup \{(v, w)\}$ 
23  |       | break
24  |     | else
25  |       |  $R_{v,w} \leftarrow R_{v,w} \cup \text{DFS}(G, (a, c))$ 
26  $\bowtie \leftarrow \lambda - N$ 
27 return  $\bowtie$ 
```

(v, w) . Notice that the procedure call $\text{DFS}(G, (v, w))$ returns all tuples $(x, z) \in V'$ that must be contained in any symmetric compatibility relation that relates v to w . Lines 19-24 of this algorithm check if $R_{v,w}$ is an equivalence relation or not, and if not, check if it can be extended to an equivalence relation being a compatibility relation too. If the while loop broke in line 22, it means that we could not add a tuple (a, c) to $R_{v,w}$ to resolve the intransitivity of (a, b) and (b, c) and thus $R_{v,w}$ cannot be a compatibility equivalence relation. But, if the while loop did not break in that line,

it means that when the while loop is exited, relation $R_{v,w}$, which by that time is transitive, is the smallest compatibility equivalence relation that relates v to w . At line 25, the relation \bowtie_F is computed by performing a set subtraction.

The time complexity of this algorithm is $O(|V|^4|Y|+|V|^8)$. Notice that computing λ and G each takes $O(|V|^4|Y|)$. At any time of the algorithm, relation $R_{v,w}$ has a size of $O(|V|^2)$. The for-loop in line 17 is performed $O(|V|^2)$, in each iteration of which it takes $O(|V|^2)$ to compute $R_{v,w}$ in line 18 and it takes $O(|V|^6)$ time to run lines 19-24. Note that Algorithm 2 is presented in a simple form for clarity. The time complexity can be improved by using some simple optimizations. For example, if the algorithm did not break in line 22, then we can mark all pairs in $R_{v,w}$ as mergeable and we do not need to do the while-loop of line 17 on them, and if the algorithm did break in line 22, then we mark all pairs in $R_{v,w}$ as unmergeable and again we do not run the while-loop in line 17 for any pair in $R_{v,w}$.

Notice that for filters F for which λ_F is an equivalence relation, it holds that $\bowtie_F = \lambda_F$. This is because for this kind of filters, we have that (1) $\lambda_F \subseteq \bowtie_F$, which is due to the definition of \bowtie_F and that λ_F is a compatibility equivalence relation for F , and (2) $\bowtie_F \subseteq \lambda_F$, which holds for any filter given the definition of λ_F and that \bowtie_F is a compatibility relation.

In the next section, we identify several classes of filters for which filter minimization or filter partitioning minimization is solvable in polynomial time.

3.9 SPECIAL CLASSES OF FILTERS

In this section, we identify several classes of filters for which FM can be solved in polynomial time. To do so, we identify where the union of all compatibility relations or the mergeability relation becomes an equivalence relation.

One such special class of filters consists of filters is one we call *observation-at-*

most-once-in-a-color filters. An observation-at-most-once-in-a-color filter is a filter in which for any observation, from all states with the same color, there is at most one outgoing edge labeled by that observation. Such filters are a generalization of the class that Saberifar *et al.* [125] called *once-appearing-observations* filters. The difference between once-appearing-observations and observation-at-most-once-in-a-color filter is that in the former each observation appears only once while in the latter an observation can appear more than one time in the filter, but only once from the states of each color. This kind of filters might arise in applications where the observations are produced by distinct and identifiable sensors, and for each group of “related” situations or locations, a distinct sensory data is observable from at most a single situation or location of that group. In particular, this can happen in environments similar to those in Figure 3.4 but with different shapes. The following theorem proves that solving the filter minimization problem and the filter partitioning minimization problem for this class takes polynomial time in size of the input filter.

<p>Problem: Observation-at-most-once-in-a-color Filter Minimization (OBS-AT-MOST-ONCE-IN-A-COL-FM)</p>
--

<p><i>Input:</i> An observation-at-most-once-in-a-color filter F.</p>
--

<p><i>Output:</i> A filter F^* such that $F \xrightarrow{L(F)} F^*$, and the number of states in F^* is minimal.</p>

Theorem 6. OBS-AT-MOST-ONCE-IN-A-COL-FM $\in P$.

Proof. According to the discussion above, if we prove that for any observation-at-most-once-in-a-color filter $F = (V, Y, C, \delta, c, v_0)$ the relation λ_F is an equivalence relation, then the proof is complete. It is easy to observe that since in F no distinct states with the same color shares an outgoing edge labeled with the same observation, we have $\lambda_F = \{(v, w) \mid c(v) = c(w)\}$. This relation is clearly an equivalence relation.

□

A similar result holds for filter partitioning minimization of the same family of filters.

Problem: Observation-at-most-once-in-a-color Filter Partitioning Minimization (OBS-AT-MOST-ONCE-IN-A-COL-FPM)

Input: An observation-at-most-once-in-a-color filter F .

Output: A filter F^* such that $F \xrightarrow{L(F)} F^*$, $F \doteq F^*$, and the number of states in F^* is minimal.

Theorem 7. OBS-AT-MOST-ONCE-IN-A-COL-FPM $\in P$.

Proof. It follows from the fact that FM and FPM share the same optimal solution for a filter for which the union of all compatibility relations is an equivalence relation. \square

The second class consists of filters which we call *at-most-two-comp-states-in-a-col*—a filter that for each color, at most two states with that color are compatible. An example of filters that fall into this class are the class of filters in which for each color, at most two states with that color exist.

Problem: At-most-two-comp-states-in-a-col Filter Minimization (AT-MOST-TWO-COMP-STATES-IN-A-COL-FM)

Input: An at-most-two-comp-states-in-a-col filter F .

Output: A filter F^* such that $F \xrightarrow{L(F)} F^*$, and the number of states in F^* is minimal.

Theorem 8. AT-MOST-TWO-COMP-STATES-IN-A-COL-FM $\in P$.

Proof. It follows from the fact that the union of all compatibility relations for any at-most-two-comp-states-in-a-col filter F is always transitive, and that for any filter F , that relation is reflexive and symmetric by Lemma 8. \square

The same result holds for filter partitioning minimization of the same family of filters.

Problem: At-most-two-comp-states-in-a-col Filter Partitioning Minimization (AT-MOST-TWO-COMP-STATES-IN-A-COL-FPM)

Input: An at-most-two-comp-states-in-a-col filter F .

Output: A filter F^* such that $F \stackrel{L(F)}{=} F^*$, $F \doteq F^*$, and the number of states in F^* is minimal.

Theorem 9. AT-MOST-TWO-COMP-STATES-IN-A-COL-FPM $\in P$

A similar class consists of filters which we call *at-most-two-merg-states-in-a-col*—a filter that for each color, at most two states with that color are mergeable. Note that this class is different than at-most-two-comp-states-in-a-col filters because a filter that has more than two compatible states per color may not be a at-most-two-comp-states-in-a-col, but it could be an at-most-two-merg-states-in-a-col filter.

Problem: At-most-two-merg-states-in-a-col Filter Partitioning Minimization (AT-MOST-TWO-MERG-STATES-IN-A-COL-FPM)

Input: An at-most-two-merg-states-in-a-col filter F .

Output: A filter F^* such that $F \stackrel{L(F)}{=} F^*$, $F \doteq F^*$, and the number of states in F^* is minimal.

Theorem 10. AT-MOST-TWO-MERG-STATES-IN-A-COL-FPM $\in P$.

Proof. It follows from the fact that the mergeability relation for any at-most-two-merg-states-in-a-col filter F is always transitive, and that for any filter F , that relation is reflexive and symmetric by Lemma 9. \square

At-most-two-comp-states-in-a-col and at-most-two-merg-states-in-a-col filters arise, for example, in applications where the task of interest might need a filter with a high number of colors (outputs) and each color is assigned to only a few states. This usually happens in localization and state estimate settings, where each state of the filter

represents an estimation of the robot's position and only those states who estimate close positions are assigned the same color.

Another class consists of filters which we call *mergeability-is-bisimilarity*— a filter for which the mergeability relation coincides with the bisimilarity relation.

Problem: Mergeability-is-bisimilarity Filter Partitioning Minimization (MERG-IS-BISIM-FPM)

Input: A mergeability-is-bisimilarity filter F .

Output: A filter F^* such that $F \xrightarrow{L(F)} F^*$, $F \doteq F^*$, and the number of states in F^* is minimal.

Theorem 11. $\text{MERG-IS-BISIM-FPM} \in P$.

Proof. For any filter F in this class, $\bowtie_F = \sim_F$. By Lemma 7, the relation \sim_F is an equivalence relation. □

A subclass of class mergeability-is-bisimilarity filters consists of filters *largest-compatibility-is-bisimilarity*— a filter for which the union of all compatibility relations coincides with the bisimilarity relation. In fact, this subclass consists of filters F for which, $\bowtie_F = \wedge_F = \sim_F$.

Both FM and FPM for this class of filters are solvable in time polynomial to the size of the filter.

These two kinds of filters, mergeability-is-bisimilarity and largest-compatibility-is-bisimilarity filters, arise in applications where the only feature of the system that can help reduce the size of the filter is due to some symmetry or indistinguishability in the environment or the underlying problem. This usually happens in tracking problems.

A subclass of largest-compatibility-is-bisimilarity filters are filters Saberifar *et al.* [125] called *no-missing-edges*– filters for which, from any state, for any observation, there is an outgoing edge labeled by that observation. This kind of filters can

be generalized to a class we call *color-no-missing-edges* filters. A filter is color-no-missing-edges if for any two states v and w for which $c(v) = c(w)$, for any observation y , if $\delta(v, y) \neq \perp$ then $\delta(w, y) \neq \perp$. An application in which color-no-missing-edges arise is navigation in grid environments, where from all cells of a row or column, the same set of observations are observable, and from each of them, the same set of actions are available.

Problem: Color-no-missing-edges Filter Minimization (COL-NO-MIS-EDG-FM)

Input: A color-no-missing-edges filter F .

Output: A filter F^* such that $F \stackrel{L(F)}{=} F^*$, and the number of states in F^* is minimal.

Theorem 12. COL-NO-MIS-EDG-FM $\in P$.

Proof. By the definition of color-no-missing-edges for each observation y , and for any two states v and w that share the same color, we have that either $\delta(v, y) = \delta(w, y) = \perp$ or $(\delta(v, y) \neq \perp) \wedge (\delta(w, y) \neq \perp)$. In this case, the three conditions of Definition 9, taking $F_1 = F_2 = F$, are identical to the conditions of Definition 6. Therefore, $\sphericalangle_F = \sim_F$, which by Lemma 7, relation \sphericalangle_F is an equivalence relation. □

A similar result holds for filter partitioning minimization of that kind of filters.

Problem: Color-no-missing-edges Filter Partitioning Minimization (COL-NO-MIS-EDG-FPM)

Input: A color-no-missing-edges filter F .

Output: A filter F^* such that $F \stackrel{L(F)}{=} F^*$, $F \doteq F^*$, and the number of states in F^* is minimal.

Theorem 13. COL-NO-MIS-EDG-FPM $\in P$.

CHAPTER 4

INTEGER LINEAR PROGRAMMING FORMULATIONS OF FM AND FPM PROBLEMS

In this chapter, we offer an alternative approach to filter minimization that improves upon the heuristic proposed by O’Kane and Shell. The basic approach is to reduce FM and FPM to integer linear programming (ILP). We consider three different integer linear programming formulations for the filter partitioning minimization problem. None of these three formulations is fully superior to one another, and each might be useful for minimizing certain types of filters. We also consider an ILP formulation of the filter minimization problem.

The material in this chapter is based on our work Rahmani and O’Kane [108].

The organization of this chapter is as follows. In Section 4.1, we present the three ILP formulations for FPM. In Section 4.2, we present experimental results, which show that the ILP formulation outperforms the algorithm of O’Kane and Shell [100], for both optimal and feasible solutions of the filter partitioning minimization problem. In Section 4.3, we discuss an ILP formulation of FM.

4.1 ILP FORMULATIONS OF THE FPM PROBLEM

The idea of the ILP formulations we propose in this chapter stems from Chapter 3, in which we proved that an optimal filter can be formed by taking the *quotient* of the original filter under some equivalence relation over the state space of the original filter. Assuming that one has the correct relation, for each equivalence class of that

relation, all of the states in that class are merged to form a single state in the reduced filter.

In Chapter 3, we provide conditions on the relation under which the quotient operation produces a well-defined filter. Specifically, the relation must be a *compatibility relation*. See Definition 6.

To illustrate the notion of compatibility again, consider filter F_1 in Figure 4.1. Some compatibility relations for this filter are $R_1 = \emptyset$, $R_2 = I_{F_1}$, $R_3 = \{(4, 5), (7, 7)\}$, $R_4 = \{(2, 3), (0, 1), (3, 4)\}$, and $R_5 = I_{F_1} \cup \{(2, 3), (3, 2), (0, 1), (1, 0), (3, 4), (4, 3), (4, 5), (5, 4)\}$, where I_{F_1} denotes the identity relation on the state space of F_1 . Observe that $\wedge_{F_1} = R_5$.

According to Definition 7, given a relation that is both a compatibility relation and an equivalence relation, we can form a quotient filter, which merges equivalent states. Notice that Definition 6 ensures that if two states v and w , both of which have an outgoing edge labeled by an observation y , are merged then their ‘ y -successors’ — $\delta(v, y)$ and $\delta(w, y)$ — must also be merged.

According to Lemma 1 and Lemma 3 in Chapter 3, for any filter $F = (V, Y, C, \delta, c, v_0)$, and any compatibility equivalence relation R for F , we have $F \xrightarrow{L(F)} F/R$ and that $F \cong F/R$. As a result, if the relation R in Lemma 2 has the minimum number of equivalence classes, then it is guaranteed that F/R is a minimal filter equivalent to F . Therefore, the FPM problem is reduced to the following problem:

Problem: Minimum-partition compatibility equivalence relation (MPCER)

Input: A filter F .

Output: A compatibility equivalence relation for F with a minimum number of equivalence classes.

Note that, due to Lemma 2, any feasible solution to the MPCER problem identifies a feasible solution to the FPM problem. Thus, even if we can find only a feasible

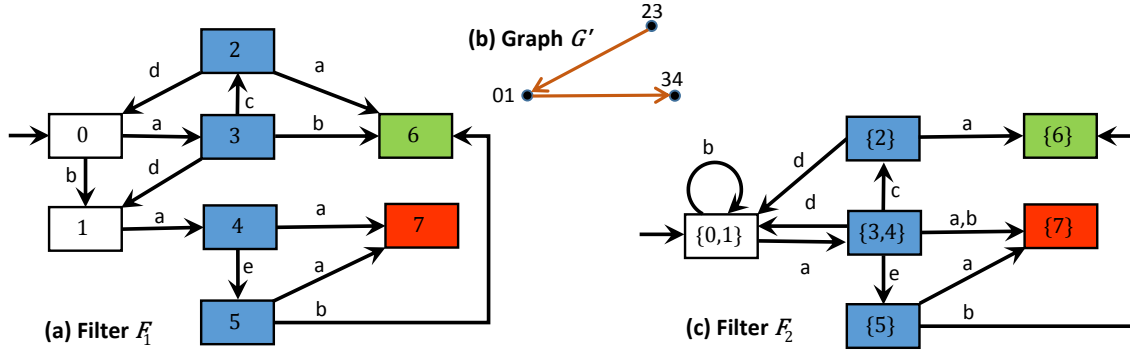


Figure 4.1: **a)** Filter F_1 . States in the left column have color 1, states in the middle column have color 2, state 6 has color 3, and state 7 has color 4. **b)** The compatibility enforcement graph of filter F_1 . **c)** A minimal filter equivalent to F_1 .

(rather than optimal) solution to MPCER, we can still use that feasible solution to find a feasible solution to the corresponding FPM instance. The ILP models introduced below are constructed by leveraging this connection between FPM and MPCER.

Assignment-based ILP

Our first formulation of FPM as an ILP is inspired by the classical ILP for the graph coloring problem [60, 89]. To describe this formulation, we first describe how to cast the problem as a mathematical program that happens to contain some nonlinear constraints. Then we show how to linearize those constraints to form an ILP and describe some simple optimizations that reduce the complexity of the program.

Nonlinear optimization formulation

In this approach, each state of F is assigned to a label from a set of $n = |V|$ labels $1, \dots, n$. These labels form an equivalence relation on V by relating pair of states that are assigned to the same label. Considering this, in the formulation, for any state v and an integer $1 \leq j \leq n$, a binary variable x_{vj} is introduced. This variable receives value 1 if state v is assigned to label j , and it receives 0 otherwise. Furthermore,

n binary variables p_1, p_2, \dots, p_n are introduced, where for each integer $1 \leq j \leq n$, variable p_j receives value 1 if label j is used for some state in the assignment, or it receives 0 otherwise. Accordingly, the MPCER problem with input F can be solved via the following (nonlinear) mathematical programming model.

<p>Minimize:</p> $\sum_{j=1}^n p_j \tag{4.1}$ <p>Subject to:</p> <ul style="list-style-type: none"> • For all $v \in V$, $\sum_{j=1}^n x_{vj} = 1. \tag{4.2}$ <ul style="list-style-type: none"> • For all $j \in \{1, \dots, n\}$ and all $v, w \in V$ such that $v \not\sim_F w$, $x_{vj} + x_{wj} \leq p_j. \tag{4.3}$ <ul style="list-style-type: none"> • For all $v, w \in V$ and all $y \in Y$ such that $\delta(v, y) \neq \perp$ and $\delta(w, y) \neq \perp$, $\sum_{j=1}^n x_{vj} x_{wj} \leq \sum_{k=1}^n x_{\delta(v,y)k} x_{\delta(w,y)k}. \tag{4.4}$ <ul style="list-style-type: none"> • For all $v \in V$ and all $j \in \{1, \dots, n\}$, $p_j \in \{0, 1\} \text{ and } x_{vj} \in \{0, 1\}. \tag{4.5}$

The objective function of this model minimizes the number of labels that are used. That is, it minimizes the size of the partition specified by the assignment. Observe that constraints of type (4.2) ensure that each state of the filter is assigned to one and only one label.

Thus, to see that an optimal solution to this program would yield an optimally reduced filter, we need only to verify that the relation induced by the x_{vi} variables is indeed a compatibility equivalence relation. We write R to denote this relation,

defined as

$$R = \{(v, w) \mid x_{v,j} = x_{w,j} = 1 \text{ for some } 1 \leq j \leq n\}. \quad (4.6)$$

The fact that R is an equivalence relation follows directly from this definition.

Constraints of types (4.3) and (4.4) together guarantee that R must be a compatibility relation in the sense of Definition 6:

- By the constraints of type (4.3), if two states v and w are not compatible, then to any label j , at most one of the states v or w is assigned. Therefore, it prevents states x and y from being related by R . Thus, $R \subseteq \wedge_F$. But, relation \wedge_F does not relate states of different colors, and thus, relation R satisfies the first part of Definition 6.
- The constraints of type (4.4) ensure that if $x_{vj}x_{wj} = 1$ for a j —that is, if both $x_{vj} = 1$ and $x_{wj} = 1$ —then $x_{\delta(v,y)k}x_{\delta(w,y)k} = 1$ for some k . This means that if v and w are merged (related by R), then $\delta(v, y)$ and $\delta(w, y)$ must also be merged. Thus, relation R satisfies the second part of Definition 6.

We conclude that a solution to this mathematical program does indeed induce a compatibility equivalence relation R with a minimal number of equivalence classes. Therefore, that relation can be used to solve MPCER, and therefore, FPM.

Linearizing the constraints

This model has all the properties to be an integer linear program except that constraints of type (4.4) are not linear. To linearize these constraints, we introduce for each pair of states $v, w \in V$, two binary variables α_{vw} and β_{vw} , the values of which are constrained as follows:

- For all $v, w \in V$ and all $j \in \{1, \dots, n\}$,

$$\alpha_{vw} \geq x_{vj} + x_{wj} - 1, \quad (4.7)$$

$$\beta_{vw} \geq x_{vj} - x_{wj}, \quad (4.8)$$

$$\alpha_{vw} + \beta_{vw} = 1, \text{ and} \quad (4.9)$$

$$\alpha_{vw}, \beta_{vw} \in \{0, 1\}. \quad (4.10)$$

According to these constraints, the variable α_{vw} becomes 1 only when $x_{vj} = x_{wj} = 1$ for a j . In this case, β_{vw} receives 0. Consider that if for no j it holds that $x_{vj} = x_{wj} = 1$, then no constraint of type (4.7) enforces α_{vw} to receive value 0 since in this case the only restrictions on α_{vw} are $\alpha_{vw} \geq 0$ and $\alpha_{vw} \geq -1$, not preventing α_{vw} from receiving 1. In this case, however, it is guaranteed that β_{vw} gets value 1 by a constraint of type (4.8) because for a j it holds that $x_{vj} = 1$ and $x_{wj} = 0$. Subsequently, constraint (4.9) makes α_{vw} receive 0.

Knowing that for each state pair v, w , the variable α_{vw} takes value 1 when and only when v and w are chosen to be merged, we can replace the inequality $\sum_{j=1}^n x_{vj}x_{wj} \leq \sum_{k=1}^n x_{\delta(v,y)k}x_{\delta(w,y)k}$ with the following inequality:

$$\alpha_{vw} \leq \alpha_{\delta(v,y)\delta(w,y)} \quad (4.11)$$

After these changes, the original formulation becomes an ILP.

Optimizing the ILP

Though we now have a correct ILP for FPM, there are several straightforward changes that can make that program more efficiently solvable.

First, observe that the current formulation introduces two variables α_{vw} and β_{vw} for each pair of states v and w , whether v and w can be merged or not. However, if we know that two states can never be merged, or even if merged, they do not enforce

via the second condition of Definition 6 any other pairs to be merged, then we do not need to introduce this kind of extra variables for them, and by doing so, we may help the solver to eliminate a considerable amount of computations.

To identify pairs (v, w) for which we require variables α_{vw} and β_{vw} , we first construct an auxiliary graph, denoted by $G' = (V', E')$, which we call *the compatibility enforcement graph* for F . To construct that graph, we set $V' = \{vw \mid v \neq w \text{ and } v \wedge_F w\}$. Then, for each pair of distinct vertices $vw, rz \in V'$, if for some $y \in Y$ it holds that $\delta(v, y) = r$ and $\delta(w, y) = z$, then we add edge (vw, rz) to V' . Finally, we remove the isolated vertices from V' .

The vertices of this graph, V' , are the pairs (v, w) for which we require variables α_{vw} and β_{vw} . An edge (vw, rz) of this graph means that in making a smaller filter, if states v is merged with state w , then state r must also be merged with z . More precisely, if $(v, w) \in R$, then it must hold that $(r, z) \in R$.

To illustrate, consider again filter F_1 in Figure 4.1. The compatibility enforcement graph of this filter is shown in Figure 4.1b. Notice that although states 4 and 5 are compatible, the graph does not have a vertex 45 since even if they are merged they do not enforce any other pair of states to be merged through the second condition of Definition 6.

Second, we add two additional types of constraints (4.21 and 4.22 below) are intended to reduce symmetry, as suggested by Méndez-Díaz and Zabala [60]. The final assignment-based ILP, combining each of these elements appears below.

<p>Minimize:</p>	$\sum_{j=1}^n p_j \tag{4.12}$
<p>Subject to:</p> <ul style="list-style-type: none"> • For all $v \in V$, 	$\sum_{j=1}^n x_{vj} = 1 \tag{4.13}$

- For all $v, w \in V$ such that $vw \in E'$ and for all $j \in \{1, \dots, n\}$,

$$x_{vj} + x_{wj} \leq p_j. \quad (4.14)$$

- For all $v, w \in V$ such that $vw \in V'$ and for all $j \in \{1, \dots, n\}$,

$$\alpha_{vw} \geq x_{vj} + x_{wj} - 1, \quad (4.15)$$

$$\beta_{vw} \geq x_{vj} - x_{wj}. \quad (4.16)$$

- For all $v, w \in V$ such that $vw \in V'$,

$$\alpha_{vw} + \beta_{vw} = 1. \quad (4.17)$$

- For all $v, w, r, z \in V$ such that $(vw, rz) \in E'$,

$$\alpha_{vw} \leq \alpha_{rz}. \quad (4.18)$$

- For all $v \in V$ and for all $j \in \{1, \dots, n\}$,

$$x_{vj}, p_j \in \{0, 1\} \quad (4.19)$$

- For all $v, w \in V$ such that $vw \in V'$

$$\alpha_{vw}, \beta_{vw} \in \{0, 1\} \quad (4.20)$$

- For all $j \in \{1, \dots, n\}$

$$p_j \leq \sum_{v \in V} x_{vj}. \quad (4.21)$$

- For all $j \in \{2, \dots, n\}$

$$p_j \leq p_{j-1}. \quad (4.22)$$

This formulation has $|V|^2 + |V| + 2|V'|$ variables— $|V|^2$ variables for x 's, $|V|$ variables for p 's, $|V'|$ variables for α 's, and $|V'|$ variables for β 's.

Representative ILP

Our second approach uses ideas based on those of Campêlo *et al.* [17,18]. Observe that to make an equivalence relation on the state space of a filter F , we can choose among the states, a set of distinct representatives so that each equivalence class be represented by a representative and then assign each state of the filter to a single representative (in the case that a state is chosen to be a representative, then it can be assigned only to itself). Also consider that one necessary condition for that equivalence relation to be a compatibility relation is that a state cannot be assigned to a representative with which is not compatible. More precisely, any state $v \in V$ can be represented only by those states that are in $S(v)$ where $S(v) = \{u \mid (u, v) \in \lambda_F\}$. Observe that v itself is in $S(v)$. Given these, in the representative ILP formulation of FPM problem, for any state v and any state $u \in S(v)$, a binary variable x_{uv} is defined. This variable receives 1 if v is represented by u , and receives 0 otherwise. Moreover, similar to the assignment-based formulation, for each state pair $v, w \in V$ such that $vw \in V'$, two binary variables α_{vw} and β_{vw} are introduced. If v and w are assigned to the same representative, then variable α_{vw} receives value 1, while β_{vw} receives value 0. Otherwise, α_{vw} receives value 0 and β_{vw} receives value 1.

These three kinds of variables participate in the representative ILP formulation of the MPCER problem as follows:

<p>Minimize:</p> $\sum_{u \in V} x_{uu} \tag{4.23}$ <p>Subject to:</p> <ul style="list-style-type: none"> • For all $v \in V$, $\sum_{u \in V} x_{uv} = 1. \tag{4.24}$
--

- For all $v, w \in V$ such that $u \not\sim_F w$ and for all $u \in S(v) \cap S(w)$,

$$x_{uv} + x_{uw} \leq x_{uu}. \quad (4.25)$$

- For all $v, w \in V$ such that $vw \in V'$ and for all $u \in S(v) \cap S(w)$,

$$\alpha_{vw} \geq x_{uv} + x_{uw} - 1, \quad (4.26)$$

$$\beta_{vw} \geq x_{uv} - x_{uw}. \quad (4.27)$$

- For all $v, w \in V$ such that $vw \in V'$,

$$\alpha_{vw} + \beta_{vw} = 1. \quad (4.28)$$

- For all $v, w, r, z \in V$ such that $(vw, rz) \in E'$,

$$\alpha_{vw} \leq \alpha_{rz}. \quad (4.29)$$

- For all $v \in V$ and for all $u \in S(v)$,

$$x_{uv} \in \{0, 1\} \quad (4.30)$$

- For all $v, w \in V$ such that $vw \in V'$

$$\alpha_{vw}, \beta_{vw} \in \{0, 1\} \quad (4.31)$$

The objective function of this formulation minimizes the number of representatives and thus the number of equivalence classes induced by the solution. While constraints of type (4.24) all together ensure all states are assigned to representatives and that any state is assigned to exactly one representative, any two *incompatible* states are prevented to be assigned to a single representative by a constraint of type (4.25). The constraints involving α_{vw} and β_{vw} have a similar meaning they had in their corresponding constraints of the assignment-based ILP. Given the x 's part of a

solution to this problem, the equivalence relation R induced by x is as follows:

$$R = \{(v, w) \mid \exists u \in S(v) \cap S(w) \text{ s.t. } x_{uv} = x_{uw} = 1\}. \quad (4.32)$$

Partial-ordering based ILP

Our third formulation is similar to the approach of Jabrayilov and Mutzel [60], who proposed a partial-ordering based ILP formulation of the graph coloring problem. This approach is similar to the assignment-based approach in that an equivalence relation over V is constructed by relating pairs of states that are assigned the same label among a set of n labels $1, \dots, n$. The difference, however, is that in this approach, states are assigned labels indirectly (rather than directly) via making a partial order on a set containing all states and all labels. It is assumed that the sequence of labels is linearly ordered, and accordingly, to make that said partial order, one can specify for each given state $v \in V$ and label $j \in \{1, \dots, n\}$ that if v is greater than j , denoted $v \succ j$, or v is smaller than j , denoted $v \prec j$. Subsequently, for each state v and label $j \in \{1, \dots, n\}$, two variables g_{jv} and l_{vj} are introduced. Variable g_{jv} receives value 1 if it is assumed that $v \succ j$, and receives value 0 otherwise. In contrast, variable l_{vj} receives value 1 if it is assumed that $v \prec j$, and otherwise it receives 0. Based on these two kind of variables, state v is assigned label j if and only if $g_{j,v} = l_{v,j} = 0$, that is, when v is neither greater nor smaller than j . To see the connection between these two kinds of variables and the variables of assignment-based approach, one can think of $x_{vj} = 1 - (g_{j,v} + l_{v,j})$. Also, in this formulation an arbitrary state $q \in V$ is chosen to assign the largest label used in the assignment.

Finally, the partial-ordering base ILP for the MPCER problem is as follows:

Minimize:	$1 + \sum_{j=1}^n g_{j,q} \quad (4.33)$
Subject to:	

- For all $v \in V$,

$$l_{v,1} = 0, \quad (4.34)$$

$$g_{n,v} = 0. \quad (4.35)$$

- For all $v \in V$ and $j \in \{1, \dots, n-1\}$,

$$g_{j,v} - g_{j+1,v} \geq 0, \quad (4.36)$$

$$g_{j,v} + l_{v,j+1} = 1, \quad (4.37)$$

$$g_{j,q} - g_{j,v} \geq 0. \quad (4.38)$$

- For all $v, w \in V$ such that $v \not\sim w$ and for all $j \in \{1, \dots, n\}$,

$$g_{j,v} + l_{v,j} + g_{j,w} + l_{w,j} \geq 1. \quad (4.39)$$

- For all $v, w \in V$ such that $vw \in V'$ and for all $j \in \{1, \dots, n\}$,

$$\alpha_{vw} \geq 1 - g_{j,v} - l_{v,j} - g_{j,w} - l_{w,j}, \quad (4.40)$$

$$\beta_{vw} \geq -g_{j,v} - l_{v,j} + g_{j,w} + l_{w,j}. \quad (4.41)$$

- For all $v, w \in V$ such that $vw \in V'$,

$$\alpha_{vw} + \beta_{vw} = 1. \quad (4.42)$$

- For all $v, w, r, z \in V$ such that $(vw, rz) \in E'$,

$$\alpha_{vw} \leq \alpha_{rz}. \quad (4.43)$$

- For all $v \in V$ and for all $j \in \{1, \dots, n\}$,

$$g_{v,j}, l_{j,v} \in \{0, 1\} \quad (4.44)$$

- For all $v, w \in V$ such that $vw \in V'$,

$$\alpha_{vw}, \beta_{vw} \in \{0, 1\} \tag{4.45}$$

Types (4.34) and (4.35) of constraints ensure that no state is less than label 1 and that no state is greater than label n , respectively. Constraints of type (4.36) and (4.37) all together make sure that each state is assigned a label.

Due to constraints of type (4.36), if a state v is greater than a label $j + 1$, then it must also be greater than label j . By constraint of type (4.37) it is not possible for a state to be greater than a label j and at the same time to be smaller than label $j + 1$. Constraints of type (4.39) prevent incompatible states from being assigned a single label. Those constraints that involve α_{vw} and β_{vw} play the same role as in the previous two ILP formulation. The relation induced by a solution to this 0-1 model is as follows:

$$R = \{(v, w) \mid \exists j \text{ s.t. } g_{j,v} = l_{v,j} = g_{j,w} = l_{w,j} = 0\}. \tag{4.46}$$

4.2 EXPERIMENTAL RESULTS FOR THE FPM PROBLEM

Next, we present some experimental results evaluating the performance of these three formulations. The implementation is in Java, using Cplex to solve ILPs, executed on an Ubuntu 16.04 computer with a 3.6GHz processor.

Experimental filters

We conducted tests using three kinds of filters. For comparison purposes, we choose two of them to be ones on which previous work performed experiments.

The first kind consists of naïve filters for a family of the *single-agent-donut* problem (Figure 4.2) by varying the number n of regions. This family of filters was originally studied in [98].

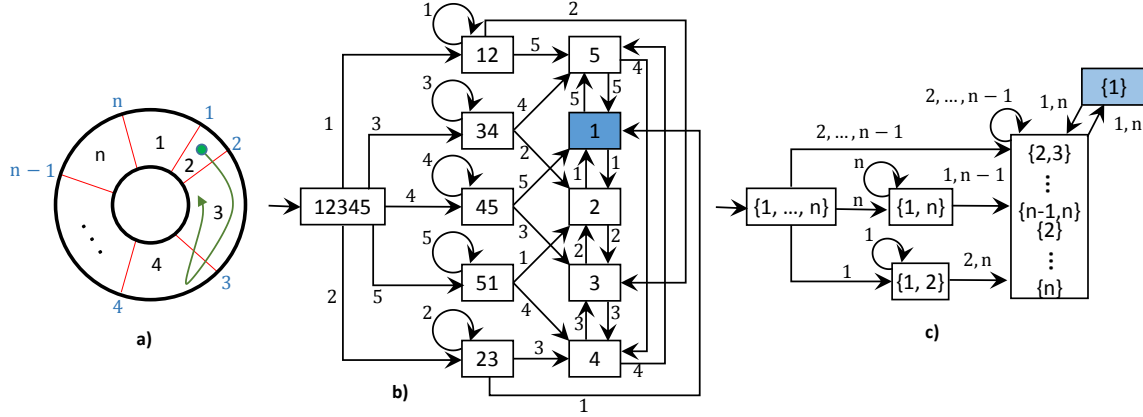


Figure 4.2: **a)** A donut-shaped environment, in which an agent moves within n regions separated by n beam sensors. When the robot crosses a beam, the system knows which sensor it was, but it does not know the direction of that crossing. The task of the system is to determine at any time whether the agent is definitely in region 1 or not. **b)** A naïve filter which is used to accomplish the task for an instance problem with $n = 5$ regions. **c)** The smallest filter the system can use to accomplish its task.

The first part of this figure shows n beam sensors, numbered $1, \dots, n$, that partition a donut-shaped environment into n regions. In this environment, an agent moves in an unpredictable but continuous path. When the agent passes a beam sensor, the system can identify which one of the n beam sensors it was, but cannot detect if the crossing was clockwise or otherwise. The task is to make an alarm when it is completely sure that the agent is in region 1. For this problem, the system can use a naïve filter, each state of which indicates a set of possible regions in which the agent can be. Accordingly, the initial state is the set of all regions $1, 2, \dots, n$. For each state, we can draw an outgoing edge whose label is a beam the system senses, and that edge goes to another state whose set of regions are obtained by filtering in the source state, the set of regions the robot can be.

For any $n \geq 3$, the naïve filter that solves an instance of the single-agent-donut problem with n regions has $2n + 1$ states. To illustrate, see Figure 4.2b, which shows the naïve filter for the case of $n = 5$. For any $n \geq 3$, the optimal filter has only 5 states, regardless of n . Figure 4.2c shows this optimal filter.

We also consider a family of *two-agent-donut* problems, which are similar to single-

agent-donut problems, but instead of a single agent, there are two agents in the environment. This kind of problem was introduced by Tovar *et al.* [146]. The goal is to determine, at any time, if the two agents are in the same region or not. When an agent crosses a beam, the system can only detect which beam sensors it was, but it can detect neither the direction of crossing nor the identity of that agent.

The third family consists of *randomly generated filters*.

Algorithms to be compared

We compare 6 algorithms. Notice that in the assignment-based ILP and the partial-ordering based ILP we set $n = |V|$. In those two ILPs, the value of $|V|$ is an upper bound for the number of labels used, or more precisely, for the number of states in the reduced filter. Clearly, that upper bound works for any filter. However, for input filters for which we know an upper bound $h < |V|$ on the number of states in the optimally reduced filter, then we can use that upper bound and set $n = h$. By so doing, the number of variables and constraints in the model may be considerably reduced. One way to obtain this sort of h is to compute a feasible solution for the FPM problem using a heuristic, efficient algorithm and then set h to the number of states of the reduced filter. Clearly, the number of states of an optimally reduced filter will be less than or equal to the number of states of a reduced filter computed by that heuristic algorithm. In our experiments, we consider whether applying this kind of bound helps or not.

We considered the following algorithms:

1. BFC: the heuristic algorithm of O’Kane and Shell [100] where conflict graphs are colored by a brute force algorithm.
2. ASGB ILP: the assignment-based ILP (with $n = |V|$).

3. ASGB ILP+H: the assignment-based ILP with $n = h$, where h is a value obtained by the heuristic algorithm of O’Kane and Shell, with conflicts colored greedily.
4. PORB ILP: the partial-ordering based ILP (with $n = |V|$).
5. PORB ILP+H: a variation on PORB ILP using the $n = h$ bound, analogous to ASGB ILP+H.
6. REP ILP: The representative ILP.

Computing optimal solutions

In this section, we compare the performance of the six algorithms in computing optimally reduced filters.

Naïve filters of single-agent-donut instances

The chart in Figure 4.3-top depicts the result of this experiment, which was performed on the naïve filters of instances of single-agent-donut problem where the number of regions varied from 3 to 50 regions. BFC outperforms the ILP based algorithms in computing optimal filters for very small filters, but is rapidly outperformed by the ILP-based algorithms as the original filter sizes grow. We also observed in this experiment that the use of a value obtained by a heuristic algorithm as an upper bound helped ASGB ILP+H and PORB ILP+H to perform better than their corresponding versions without seeds. REP ILP did not perform well for this kind of filters.

Naïve filters of two-agents-donut instances

Figure 4.3-bottom shows the results of this experiment. In a time limit of eight hours, BFC could compute optimal filters only for instances with up to seven regions. In the same time limit, ILP based algorithms could make optimal filters for one

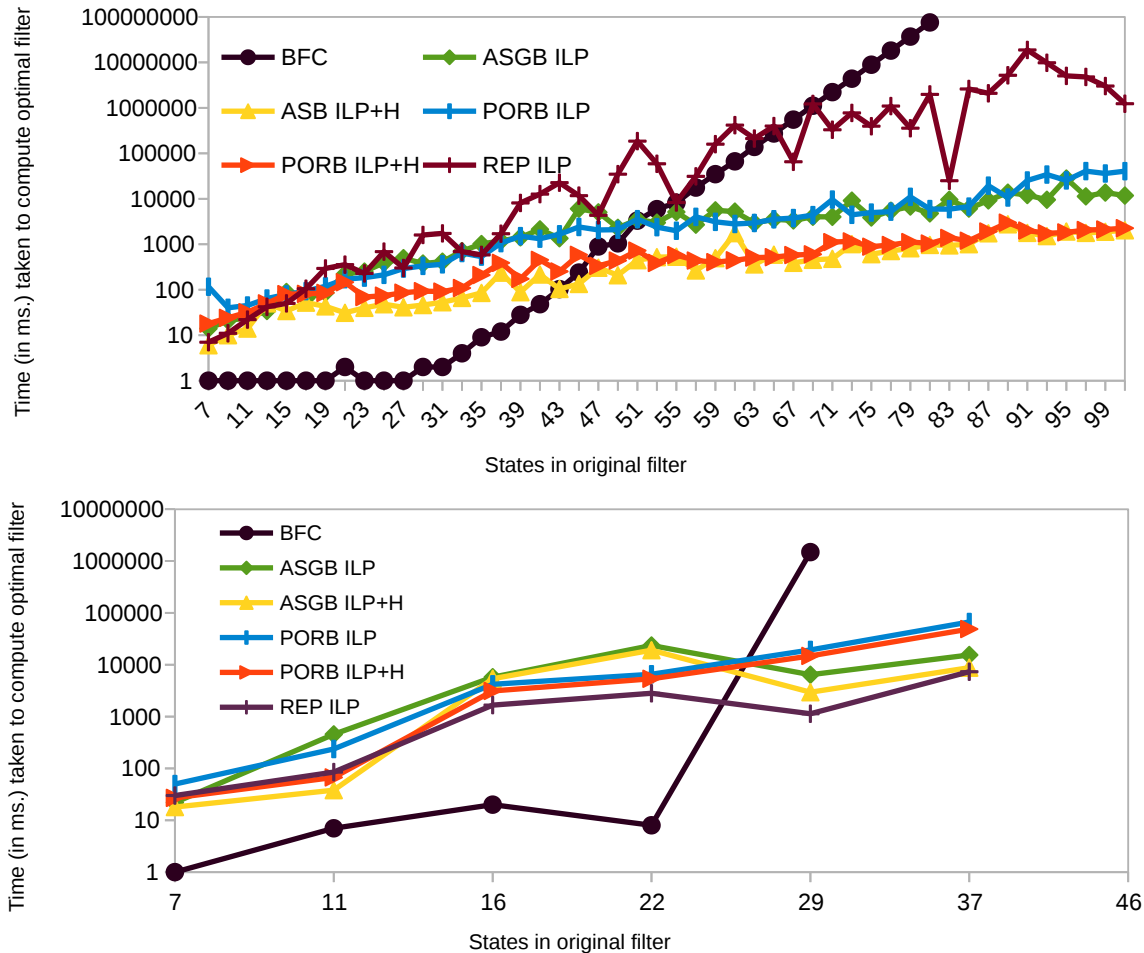


Figure 4.3: The performance of the algorithms in computing optimal filters for: **(top)** naïve filters of single-agent-donut instances, **(bottom)** naïve filters of two-agent-donut instances. Notice that the vertical axis in both charts is in logarithmic scale. For each filter size, the experiment was performed for only one filter of that size.

more instance—an instance with eight regions. Consider that although ILP based algorithm could solve only one more instance, the naïve filter for the instance with 8 regions (which has 37 states) has 8 states more than the naïve filter for 7 regions (which has 29 states).

Random filters with linear number of compatible pairs

In this experiment, we generated filters for which the number of distinct compatible pairs was less than $3|V|$, and the compatibility enforcement graph had both number

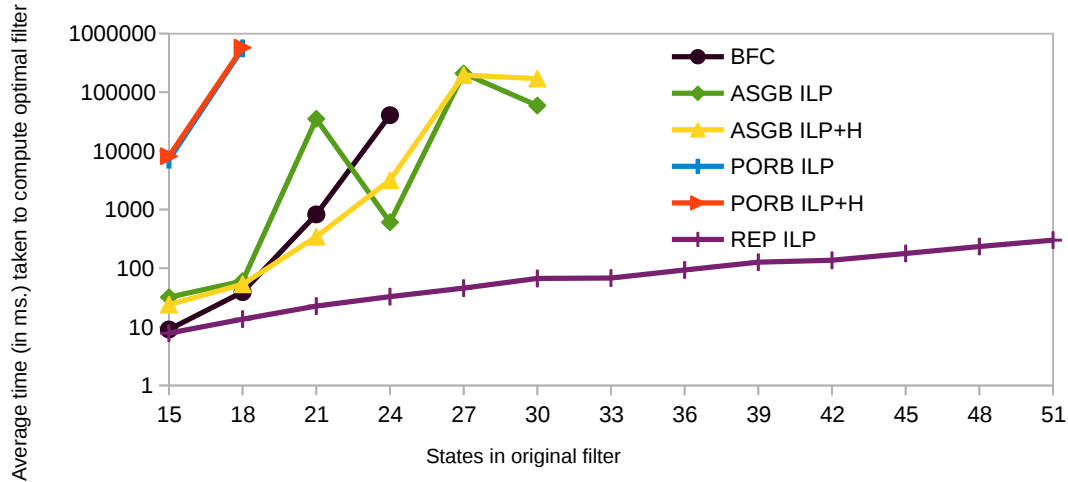


Figure 4.4: The performance of the algorithms in computing optimal filters for random filters for which the number of compatible pairs was linear to the number of states of the filter. For each of the filter sizes, the plotted time is the average time to minimize the filter over 10 randomly generated filters of that size. If an algorithm could not find an optimal solution for a filter on a given size in half an hour, then it was considered as a failure, and as a result, the average time is not plotted. Note again that the vertical axis of the chart is in logarithmic scale.

of vertices and edges around $|V|$. The observation space and color space upon which the filters are constructed has sizes of five and three, respectively. For each state v and an observation y , on the basis of a probability of $1/2$ it was chosen if v must have an outgoing edge labeled y or not, and if yes, then the destination of that outgoing edge was chosen randomly among the states of the filter. Figure 4.4 compares the performance of the algorithms for this kind of filters. One notable result about this experiment is that the representative ILP considerably outperformed the other algorithms. This is because the REP ILP introduces a variables x_{uv} only when u and v are compatible. More precisely, the number of variables in the REP ILP model was linear to the number of states of the filter, while the number of variables of the other ILPs was quadratic to the number of states of the filter.

Computing smaller filters

This section considers experiments that use ILP models for finding smaller (rather than optimal) filters where optimal filters cannot be computed due to lack of time. Especially, we are interested in trade-off between the time a solver spends and the quality of a solution returned. Figure 4.5 presents the results of our experiments on naïve filters of two instances of two-agents-donut problem, which are considered difficult to minimize. Notice that each algorithm had 10 minutes to find best solutions it could. The heuristic algorithm with greedy-random coloring was performed as many times as it could during 10 minutes. As an example, this algorithm was performed 34749 rounds to find its best solution on the naïve filter of two-agent-donut with 15 regions. Results shows that although the heuristic algorithm can find good feasible solutions very quickly, we can wait a fairly small amount of time for the ILPs to obtain better solutions than those obtained by the heuristic algorithm.

Discussion

In this section, we discuss a few observations we made. Our experiments show that none of the three proposed ILP formulations is fully superior to one another and that each one is useful for certain kind of filters. In particular, we observed,

1. from the experiment on random filters (Section 4.2 and Figure 4.4), that the representative ILP formulation is superior to the assignment-based ILP and partial-ordering ILP for filters whose union of all compatibility relations is not dense;
2. from the experiment on optimal filters for single-agent-donut problems (Section 4.2 and Figure 4.3), that the assignment-based ILP and partial-ordering ILP are superior to the representative ILP formulation for filters whose union of all compatibility relations is dense;

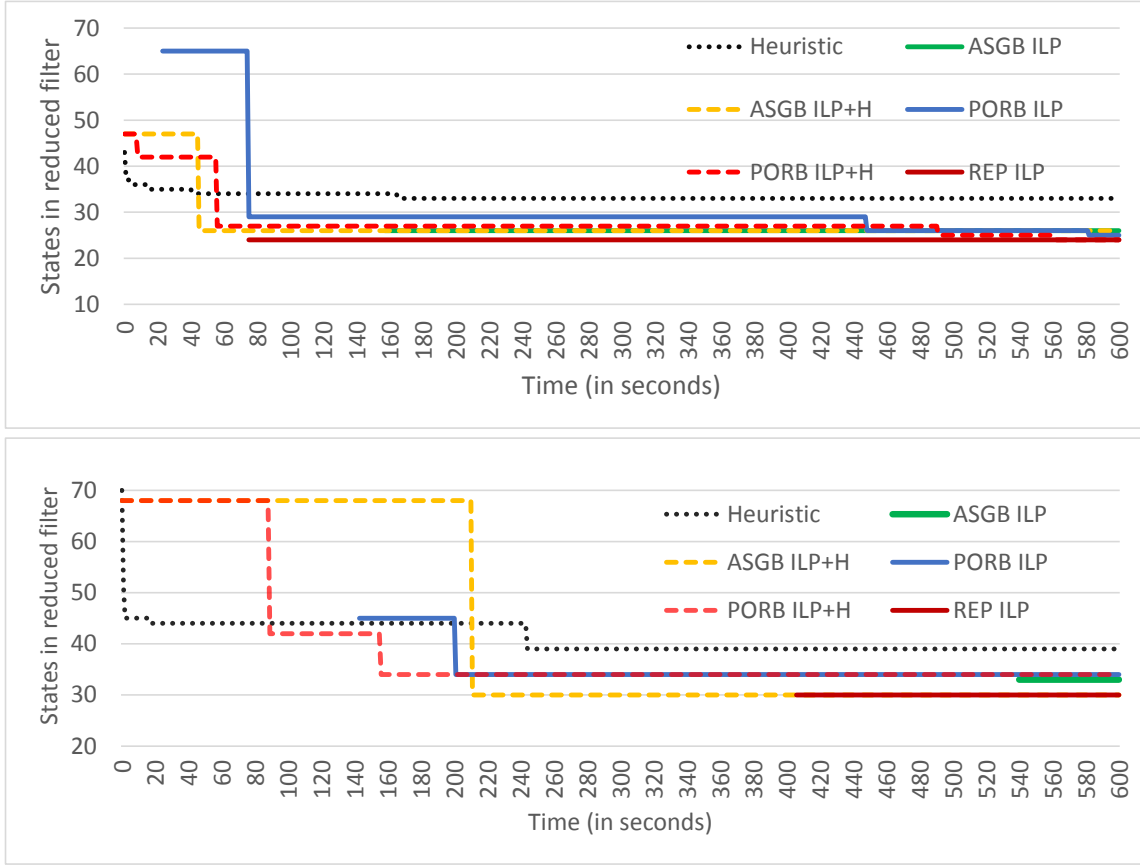


Figure 4.5: The trade-off chart of running time vs quality of solution of reduced filters found by the algorithms on **(top)** the naïve filter of two-agents-donut problem with 13 regions **(bottom)** the naïve filter of two-agents-donut problem with 15 regions. The former filter has 92 states, and the later has 121 states.

3. from the experiment on random filters (Section 4.2 and Figure 4.4), that the assignment-based ILP is superior to the partial-ordering ILP for filters whose union of all compatibility relations is not dense; and
4. from the experiment on computing feasible solutions (Section 4.2 and Figure 4.5), that for filters that are hard to minimize, the partial-ordering ILP finds feasible solutions faster than the assignment-based ILP.

The first two observations are supported by the fact that the number of variables in the representative ILP model is linear to the number of compatible pairs of states, while the number of variables in the other two ILP models are always quadratic to

the number of states of the filter, regardless of the number of compatible pairs of states.

To summarize, our results suggest several general rules of thumb for selecting the most appropriate ILP model.

- For filters that have a large number of colors relative to the number of states, the representative ILP model is likely to outperform the other two ILP models. Such filters might arise, for example, in state estimate settings, where the filter should produce different outputs for each of its internal states. Likewise, if the number of observations is large relative to the number of states, the representative ILP model also seems to outperform the other models. This would be the case, for example, in robotic perception applications in which large amounts of partially redundant data are available.
- For filters whose number of colors and observations are relatively small compared to the number of states of the filter—notably, in cases where the system is using sparse information to reason about long-term dynamics of its environment, such as in some tracking problems—the assignment-based and partial-ordering ILP models tend to fare better. In such cases, our results suggest to use the assignment-based ILP if the number of states of the filters is small, otherwise use the partial-ordering ILP.

4.3 ILP FORMULATION OF THE FM PROBLEM

In this section, we present an ILP formulation of FM. The material presented in this section resulted from a recent study [171] co-authored with Zhang, O’Kane, and Shell.

Consider Lemma 6 in Chapter 3 again. According to this formula, if \mathbf{M} is a closed covering for F , then F/\mathbf{M} is guaranteed to be a feasible to FM. Also, any optimal

solution to FM can be computed by making the quotient of the original filter under a closed covering with minimum number of compatibility classes. As a result, the filter minimization problem is reduced to the following problem:

Problem: Minimal closed covering problem (MCCP)

Input: A filter F .

Output: A closed covering for F with a minimum number of compatibility classes.

The ILP formulation we provide solves MCCP. Note also because any feasible solution to MCCP yields a feasible solution to FM, this ILP formulation is used not only for finding optimal solutions of FM but also for sub-optimal solutions.

The formulation we present is very similar to the Assignment-based formulation of FPM. We again first cast FM as mathematical problem, which contains nonlinear constraints, and then linearize those nonlinear constraints.

Nonlinear optimization formulation

In this approach we assign the states of F into $n = |V|$ compatibility classes, and the objective becomes to minimize the number of nonempty compatibility classes. To indicate whether each compatibility classes is empty or not, we use n binary variables p_1, p_2, \dots, p_n , where for each integer $1 \leq j \leq n$, variable p_j receives value 1 if at least a state of F is assigned to compatibility class L_j , or it receives 0 otherwise. For any state v and an integer $1 \leq j \leq n$, a binary variable x_{vj} is introduced. This variable receives value 1 if state v is assigned to class j , and it receives 0 otherwise. Also, for each state $v \in V$ and observation $y \in Y$, a binary variable t_{vy} is introduced, which receives value 1 if v has an outgoing transition for y , and it receives 0 otherwise. Note that in this formulation, unlike the assignment-based formulation for FPM, a state can be assigned more than one label (compatibility class). Accordingly, the MCCP problem with input F can be solved via the following (nonlinear) mathematical programming model.

Minimize:

$$\sum_{j=1}^n p_j \quad (4.47)$$

Subject to:

$$\sum_{j=1}^n x_{jv_0} \geq 1 \quad (4.48)$$

- For all $j \in \{1, \dots, n\}$ and all $v, w \in V$ such that $v \not\sim_F w$,

$$x_{vj} + x_{wj} \leq p_j. \quad (4.49)$$

- For all $j \in \{1, \dots, n\}$ and all $y \in Y$,

$$\sum_{i=1}^n \prod_{v \in V} (2 - x_{vj} - t_{vy} + x_{\delta(v,y)i}) \geq 1 \quad (4.50)$$

- For all $v \in V$ and all $j \in \{1, \dots, n\}$,

$$p_j \in \{0, 1\} \text{ and } x_{vj} \in \{0, 1\}. \quad (4.51)$$

The objective (4.47) is to minimize the number of non-empty compatibility classes in \mathbf{M} . Constraint (4.48) requires that the initial state of F be contained in at least one compatibility class. This constraint with constraints of type (4.50) all together guarantee that every state of F that are reachable from F are covered by the covering.

Constraints (4.50) ensure that the covering is closed. Formally for each compatibility class $\forall j \in \{1, 2, \dots, |V|\}$ and observation $y \in Y$:

$$\exists i \in \{1, 2, \dots, |V|\}, \text{ s.t., } \forall v \in V, \underbrace{\left((x_{vj} t_{vy} = 1) \implies (x_{\delta(v,y)i} = 1) \right)}_{\text{all } y\text{-children of } L_j \text{ are contained in } L_i}.$$

Intuitively, for any compatibility class L_j and observation y , there must exist a compatibility class L_i that contains all state that are reachable by y from a state within L_j . Once a solution to the ILP is computed, for each j for which $p_j = 1$, we form a compatibility class and assign to that class, all the states v 's for which $x_{vj} = 1$. Those compatibility classes form a closed covering of the filter. Then,

we make a reduced filter by quotienting the original filter under the obtained closed covering.

Linearizing the constraints

To convert the programming model into an integer linear programming, one need to linearize constraints of type (4.50).

To linearize them for each $i, j \in \{1, 2, \dots, n\}$ and $y \in Y$, a binary variable a_{ij}^y is introduced. The value of this variable determines whether in the output filter there is a transition labeled y from state made for L_i to state made for L_j .

If $a_{ij}^y = 1$, then the value of term $\prod_{v \in V} (1 - x_{vj} + 1 - t_{vy} + x_{\delta(v,y)i})$ 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. Formally, for all $i, j \in \{1, \dots, n\}$, $v \in V$, and $y \in Y$,

$$a_{ij}^y + x_{vj} + t_{vy} - x_{\delta(v,y)i} \leq 2.$$

Then to express constraints (4.50), we need for all $i \in \{1, \dots, n\}$ and $y \in Y$,

$$\sum_{j=1}^{|V|} a_{ij}^y \geq 1.$$

Accordingly, the programming model becomes the following integer linear programming model, to which constraints (4.57) are added to reduce symmetry as suggested by Méndez-Díaz and Paula [89].

Minimize:	$\sum_{j=1}^n p_j$	(4.52)
Subject to:	$\sum_{j=1}^n x_{v_0j} \geq 1$	(4.53)
<ul style="list-style-type: none"> • For all $j \in \{1, \dots, n\}$ and all $v, w \in V$ such that $v \not\ll_F w$, 	$x_{vj} + x_{wj} \leq p_j.$	(4.54)

- For all $i, j \in \{1, \dots, n\}$, $v \in V$, and $y \in Y$,

$$a_{ij}^y + x_{vj} + t_{vy} - x_{\delta(v,y)i} \leq 2 \quad (4.55)$$

- For all $i \in \{1, \dots, n\}$ and $y \in Y$,

$$\sum_{j=1}^n a_{ij}^y \geq 1 \quad (4.56)$$

- For all $j \in \{2, \dots, n\}$,

$$p_j \leq p_{j-1} \quad (4.57)$$

- For all $v \in V$ and all $j \in \{1, \dots, n\}$,

$$p_j \in \{0, 1\} \text{ and } x_{vj} \in \{0, 1\}. \quad (4.58)$$

Note that compared to the assignment-based ILP formulation of FPM, this ILP formulation is more difficult to solve because it has more constraints and each constraints of types (4.56) and (4.55) have more terms comparing to those constraints in the assignment-based ILP formulation of FPM that ensured if two states are merged, then the states to which they go by an observation are also merged.

CHAPTER 5

PLANNING TO CHRONICLE

In this chapter we study the problem of using autonomous robots to record a sequence of events that happen unpredictably in an environment. The material of this chapter is based of our work Rahmani, Shell, and O’Kane [111], which appeared in WAFR 2020, and its extension [113], accepted to appear in the International Journal of Robotics Research.

The organization of this chapter is as follows: In Section 5.1, we introduce our modeling and present our problem statement, in Section 5.2, we present our algorithm, in Section 5.3, we present a theoretical result, in Section 5.4, we present two algorithms that are more efficient than the general algorithm for two special inputs of the problem, in Section 5.5, we present a greedy algorithm which we use later to compare the quality of solutions computed by our algorithm, in Section 5.6, we present several language ‘mutators’, and in Section 5.7, we present our case studies.

5.1 THE PROBLEM

To begin, we introduce the problem formalization, starting with the most basic elements of the model.

Events and observations

The essential objects of interest are *events*, that is, atomic occurrences situated at specific times and places. We treat each event as a letter drawn from a finite alphabet

E , a set which contains all possible events. Any finite sequence of events, in particular a *story* ξ the robot wants to record from the events that occur in the system, is a word in E^* . (The set of finite sequences is written here using the Kleene star.)

We model the occurrence of events using a structure defined as follows.

Definition 13. [event model] An *event model* $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$ is a tuple in which

- S , a nonempty finite set, is the *state space* of the model;
- $\mathbf{P} : S \times S \rightarrow [0, 1]$ is the *transition probability function* of the model, such that for each state $s \in S$, $\sum_{s' \in S} \mathbf{P}(s, s') = 1$;
- $s_0 \in S$ is the *initial state*;
- E is the set of all possible events;
- $g : S \times E \rightarrow [0, 1]$ is an *event ‘going-on’* function defined so that for state s and event e , value $g(s, e)$ is the probability that event e happens at state s . We assume that for each $e \in E$, $g(s_0, e) = 0$.

An execution of the model starts from the initial state s_0 and then, at each time step k , the system makes a transition from state s_k to state s_{k+1} , the latter being chosen randomly based on \mathbf{P} from those states for which $\mathbf{P}(s_k, \cdot) > 0$. This execution specifies a path $s_0 s_1 \dots$. For every time step k , when the system enters state s_k , some (possibly empty) set of events occurs simultaneously, each event $e \in E$ occurring with probability $g(s_k, e)$, independently from other events.

We are interested in scenarios in which a robot is tasked with recording certain sequences of events. We model the state of the event model as only partially observable to the robot. That is, the current state s_k of the event model is hidden from the robot, but the system instead emits an output observable to the robot at each time step. The next definition formalizes the idea.

Definition 14. [observation model] For a given event model $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$, an *observation model* $\mathcal{B} = (Y, h)$ is a pair in which

- Y is a nonempty set of *observations* or outputs;
- $h : S \times Y \rightarrow [0, 1]$ is the *emission probability function* of the model, such that for each state $s \in S$, $\sum_{y \in Y} h(s, y) = 1$.

At each time step, when the system enters a state s_k , it emits an output y_k , drawn according to $h(s_k, \cdot)$. The emitted output y_k is observable to the robot. We consider, as important special cases, two particular types of observation models.

Definition 15. Given an event model $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$ with observation model $\mathcal{B} = (Y, h)$, we say that \mathcal{B} makes \mathcal{M} *fully observable* if (1) $Y = S$, and (2) $h(s, y) = 1$ if and only if $s = y$.

At the other extreme, another special event model is one in which the emitted outputs do not help at all to reduce uncertainty.

Definition 16. Given an event model $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$ with observation model $\mathcal{B} = (Y, h)$, then \mathcal{B} causes the event model to be *fully hidden* if the observation space Y is a .

Story specifications, belief states, and policies

As the system evolves along a path $s_0 s_1 s_2 \dots$, the robot attempts to record some of the events that actually occur in the world to form a story $\xi \in E^*$. We specify the desired story using a deterministic finite automaton (DFA) $\mathcal{D} = (Q, E, \delta, q_0, F)$, where Q is its state space, E is its alphabet, $\delta : Q \times E \rightarrow Q$ is its transition function, q_0 its initial state, and $F \subseteq Q$ is the set of all final (accepting) states of the automaton. In other words, we want the robot to make a story ξ in the language of \mathcal{D} , denoted $\mathcal{L}(\mathcal{D})$,

which is the set of all strings in E^* that when are tracked from q_0 , the automaton reaches an accepting state.

The semantics of event capture are as follows. At each step $k \geq 0$, the robot chooses one event e from E to attempt to record in the next step, $k + 1$. If any of the actual events that do happen at step $k + 1$ (an event e can happen at s_{k+1} only if $g(s_{k+1}, e) > 0$) match the robot's prediction, then the robot successfully records this event; otherwise, it records nothing. The robot is aware of the success or failure of each of its attempts. The robot stops making guesses and observations once it has recorded a desired story—a story in $\mathcal{L}(\mathcal{D})$.

To estimate the current state, the robot maintains, at each time step k , a belief state $b_k : S \rightarrow [0, 1]$, in which $\sum_{s \in S} b_k(s) = 1$.

The robot's predictions are governed by a policy $\pi : \Delta(S) \times Q \rightarrow E$ that depends on the belief state and the state of the DFA. At time step $k + 1$, the robot may append a recorded event to ξ_k via the following formula:

$$\xi_{k+1} = \begin{cases} \xi_k \pi(b_k, q_k) & \pi(b_k, q_k) \text{ happened at } s_{k+1} \\ \xi_k & \pi(b_k, q_k) \text{ did not happen at } s_{k+1}. \end{cases} \quad (5.1)$$

The initial condition is that $\xi_0 = \epsilon$, in which ϵ is the empty string. The robot changes the value of variable q_k only when the guessed event actually happened:

$$q_{k+1} = \begin{cases} \delta(q_k, \pi(b_k, q_k)) & \pi(b_k, q_k) \text{ happened at } s_{k+1} \\ q_k & \pi(b_k, q_k) \text{ did not happen at } s_{k+1}. \end{cases} \quad (5.2)$$

The robot stops when $q_k \in F$.

Optimal recording problems

The robot's goal is to record a story (or video) as quickly as possible. We consider this problem in three different settings: a general setting without any restriction on the

observation model, a setting in which the observation model makes the event model fully observable, and a final one in which the event model becomes fully hidden.

Definition 17. [correct policy] For a given event set E , event model $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$ with observation model $\mathcal{B} = (Y, h)$, DFA $\mathcal{D} = (Q, E, \delta, q_0, F)$, and policy $\pi : \Delta(S) \times Q \rightarrow E$, we say π is a *correct policy* if it ensures, with probability 1, that a story within $\mathcal{L}(\mathcal{D})$ will eventually be captured.

First, the general setting.

Problem: Recording Time Minimization (RTM)

Input: An event set E , an event model $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$ with observation model $\mathcal{B} = (Y, h)$, and a DFA $\mathcal{D} = (Q, E, \delta, q_0, F)$.

Output: A correct policy that minimizes the expected number of steps k until $\xi_k \in \mathcal{L}(\mathcal{D})$; or ‘NO SOLUTION’ if no correct policy exists.

Note that k is not necessarily the length of the resulting story ξ_k , but rather is the number of steps the system runs to capture that story. In fact, since the robot captures at most one event in each time step, $|\xi_k| \leq k$.

The second setting constrains the system to be fully observable.

Problem: RTM with Fully Observable Model (RTM/FOM)

Input: An event set E , an event model $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$, and a DFA $\mathcal{D} = (Q, E, \delta, q_0, F)$.

Output: A correct policy that, under observation model $\mathcal{B}_{obs}(\mathcal{M})$, minimizes the expected number of steps k until $\xi_k \in \mathcal{L}(\mathcal{D})$; or ‘NO SOLUTION’ if no correct policy exists.

The third setting assumes a fully hidden event model state.

Problem: RTM with Fully Hidden Model (RTM/FHM)

Input: An event set E , an event model $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$, and a DFA $\mathcal{D} = (Q, E, \delta, q_0, F)$.

Output: A correct policy that, under observation model $\mathcal{B}_{hid}(\mathcal{M})$, minimizes the expected number of steps k until $\xi_k \in \mathcal{L}(\mathcal{D})$; or ‘NO SOLUTION’ if no correct policy exists.

5.2 ALGORITHM DESCRIPTION

Next we give an algorithm for RTM, which also solves RTM/FOM and RTM/FHM, essentially special cases of RTM.

The Goal POMDP

The first step of the algorithm constructs a specific partially observable Markov decision process (POMDP), which we term the Goal POMDP, as follows:

Definition 18. [Goal POMDP] For an event model $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$ with observation model $\mathcal{B} = (Y, h)$, and a DFA $\mathcal{D} = (Q, E, \delta, q_0, F)$, the associated *Goal POMDP* is a tuple $\mathcal{P}_{(\mathcal{M}, \mathcal{B}, \mathcal{D})} = (X, A, b_0, \mathbf{T}, X_G, Z, \mathbf{O}, c)$, in which

1. $X = S \times Q$ is the state space;
2. $A = E$ is the action space;
3. $b_0 \in \Delta(X)$ is the initial belief state, in which $b_0(x) = 1$ if and only if $x = (s_0, q_0)$;
4. $\mathbf{T} : X \times A \times X \rightarrow [0, 1]$ is the transition probability function such that, assuming $\mathbb{1}_{\{A\}}(\cdot)$ to be set A 's indicator function, for each $e \in E$ and $(s, q), (s', q') \in X$,

$$\mathbf{T}((s, q), e, (s', q')) = \begin{cases} \mathbf{P}(s, s') \cdot g(s', e) & \text{if } q \notin F, q' = \delta(q, e), \\ & \text{and } q' \neq q \quad (4.a) \\ \\ \mathbf{P}(s, s') \cdot g(s', e) \cdot \\ \mathbb{1}_{\{q'\}}(\delta(q, e)) + & \text{if } q \notin F, \\ \mathbf{P}(s, s') \cdot (1 - g(s', e)) & \text{and } q' = q \quad (4.b) \\ \\ 1 & \text{if } q \in F, q' = q, \\ & \text{and } s = s' \quad (4.d) \\ \\ 0 & \text{otherwise;} \end{cases}$$

5. $X_G = S \times F$ is the set of goal states;
6. $Z = (\{\text{True}, \text{False}\} \times Y) \cup \{\perp\}$, in which \perp is used for the observation that the robot has completed recording a desired story, is the set of observations;
7. $\mathbf{O} : A \times X \times Z \rightarrow [0, 1]$ is the observation probability function such that for each $e \in E$, $s \in S$, $q \in Q$, and $y \in Y$:
 - (a) $\mathbf{O}(e, (s, q), (\text{True}, y)) = h(s, y) \cdot g(s, e)$ if $q \notin F$,
 - (b) $\mathbf{O}(e, (s, q), (\text{False}, y)) = h(s, y) \cdot (1 - g(s, e))$ if $q \notin F$,
 - (c) $\mathbf{O}(e, (s, q), \perp) = 1$ if $q \in F$;
8. $c : X \times A \rightarrow \mathbb{R}_{\geq 0}$ is the cost function such that for each $x \in X$ and $a \in A$, $c(x, a) = 1$ if $x \notin X_G$, and $c(x, a) = 0$ otherwise.

Figure 5.1 illustrates this construction for an elementary example. Each state of this POMDP is a pair (s, q) indicating the situation where, under an execution of the system, the current state of the event model is s and the current state of the DFA is q . For each $x, x' \in X$ and $a \in A$, $\mathbf{T}(x, a, x')$ gives the probability of transitioning from state x to state x' under performance of action a . In the context of our event model, each transition corresponds to a situation where the robot chooses an event e to observe and the event model makes a transition from a state s to s' . If e happens at s' and $\delta(q, e) \neq q$, then the robot records e and then changes the current state of the DFA to $\delta(q, e)$; otherwise, it does nothing and the DFA remains in state q . These correspond to cases (4.a) and (4.b) above, respectively. Note that the term $\mathbf{P}(s, s') \cdot g(s', e) \cdot \mathbb{1}_{\{q'\}}(\delta(q, e))$ in case (4.b) corresponds to the situation where $\delta(q, e) = q$ and the predicted event e happens at s' , while term $\mathbf{P}(s, s') \cdot (1 - g(s', e))$ corresponds to the situation where the predicted event e , regardless of whether $\delta(q, e) = q$ or not, does not happen at s' . Case (4.c) makes all the goal states of the POMDP *absorbing* states. The goal states of the POMDP are those in which the robot has recorded a story, i.e., the current state of the specification DFA is accepting.

For each $a \in A$, $x \in X$, and $z \in Z$, the function $\mathbf{O}(a, x, z)$ is an observation model, its value being the probability of observing z given that the system has entered state x via action a . The POMDP has a special observation, \perp , which is observed only when a goal state is reached. Any other observation is a pair (r, y) where $r \in \{\text{True}, \text{False}\}$ discloses whether the robot's prediction was correct—the event did happen—or not, and y indicates the sensed observation the robot made (as per \mathcal{B}). Rules 7a–7b ensure that the first element of the observation pair informs the robot whether its prediction was correct. To see this, if the robot has predicted e to occur, the event model has entered state s such that e has happened at s , and the robot has made an observation y , then the probability of observing (True, y) by entering to state (s, q) via action e is equal $h(s, y)g(s, e)$ (case 7a). If event e has not happened at s , then the robot's

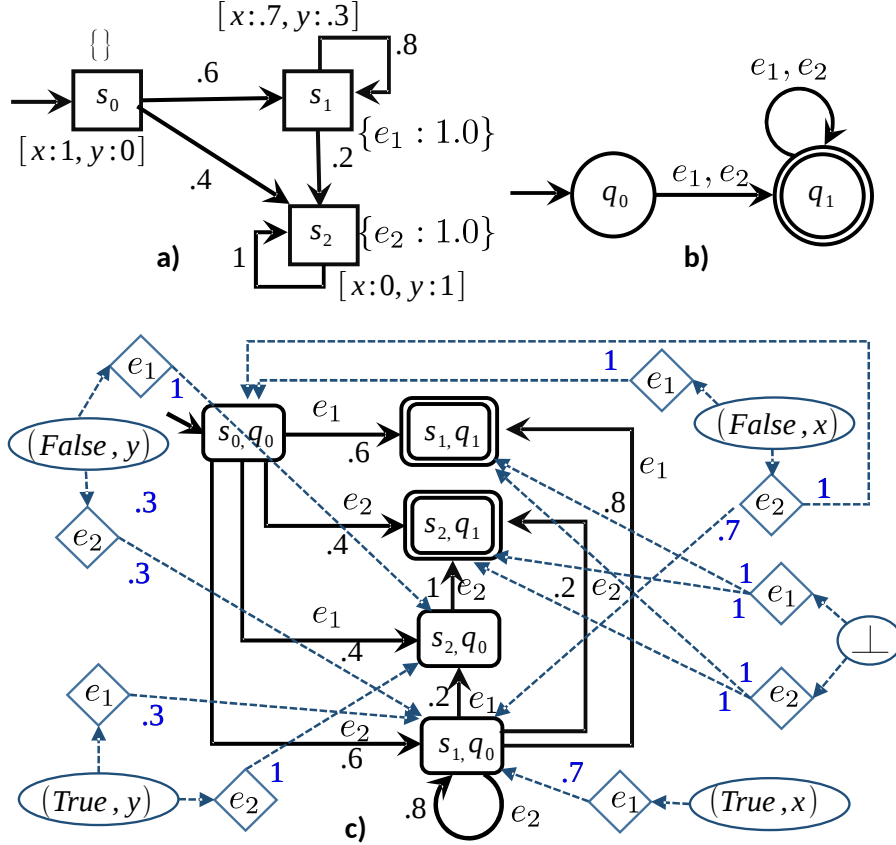


Figure 5.1: **a)** An event model \mathcal{M} with its observation model \mathcal{B} . **b)** A DFA \mathcal{D} , specifying event sequences that contain at least one event. **c)** The Goal POMDP $\mathcal{P}_{(\mathcal{M}, \mathcal{B}; \mathcal{D})}$, constructed by Definition 18. (Self-loop transitions of the goal states have been omitted to try reduce visual clutter.)

prediction has to be wrong, and thus, the probability of observing $(False, y)$ in state (s, q) when it is reached via action e is $h(s, y)(1 - g(s, a))$ (expressed in case 7b). Case 7c indicates the observation that the robot has completed recording of a story in $\mathcal{L}(\mathcal{D})$.

After making the product automaton in Definition 18, our algorithm first checks whether or not there is a policy that assures a desired story will be captured. Then, if there exists such a policy, it computes a policy minimizing the expected number of steps to record such a policy. We first focus on computing such a policy, assuming such a policy exists, in Section 5.2 (and Section 5.2 as well for special inputs of the

problem), and then we discuss how our algorithm can check if such a policy exists or not in Section 5.2.

Solving the Goal POMDP

$$b_z^a(x) = Pr(x|z, a, b) = \frac{\mathbf{O}(a, x, z) \sum_{x' \in X} \mathbf{T}(x', a, x) b(x')}{Pr(z|a, b)}, \quad (5.3)$$

in which

$$Pr(z|a, b) = \sum_{x \in X} \mathbf{O}(a, x, z) \sum_{x' \in X} \mathbf{T}(x', a, x) b(x'). \quad (5.4)$$

For this belief MDP, the cost of each action a at belief state b is

$$c'(b, a) = \sum_{x \in X} b(x) c(x, a).$$

In our case, $c'(b, a) = 1$ if b is a not a goal belief state, and otherwise $c'(b, a) = 0$. An optimal policy $\pi^* : X \rightarrow A$ for this MDP is formulated as a solution to the Bellman recurrences

$$V'^*(b) = \min_{a \in A} (c'(b, a) + \sum_{z \in Z} Pr(z|a, b) V'^*(b_z^a)), \quad (5.5)$$

$$\pi'^*(b) = \arg \min_{a \in A} (c'(b, a) + \sum_{z \in Z} Pr(z|a, b) V'^*(b_z^a)). \quad (5.6)$$

Any standard technique may be used to solve these recurrences. For surveys on methods, see [12, 120, 131]. Note that, in general, solutions to POMDPs are approximate solutions because it is intractable to provide exact solutions for POMDPs. An optimal policy computed via these recurrences prescribes, for any belief state reachable from b_0 , an optimal action to execute. Hence, the robot executes, at each step, the action given by the optimal policy, and then updates its belief state via (5.3). One can show, via induction, that at each step i , there is a unique $q_i \in Q$ such that belief state b_i has outcomes only for (but probably not all) $x_j = (s_j, q_i) \in X, j = 1, 2, \dots, |S|$. As such, function $\beta : \Delta(X) \rightarrow \Delta(S) \times Q$ maps each b_i of those belief states to a tuple (d, q_i) , where for each $s \in S, d(s) = b((s, q_i))$. Subsequently, the optimal policy π^* computed for $\mathcal{P}_{(\mathcal{M}, \mathcal{B}; \mathcal{D})}$ can be mapped to an optimal solution $\pi^* : \Delta(S) \times Q \rightarrow A$ to RTM, by interpreting $\pi^*(\beta(b_i)) = \pi'^*(b_i)$, for each reachable belief state $b_i \in \Delta(X)$.

Solving RTM/FOM via a Goal MDP

The previous construction can be used to solve RTM/FOM instances too. But, since the event model fed into a RTM/FOM is fully observable, it seems rather more sensible, especially in terms of solution tractability, to construct a Goal MDP. To do so, for the event model and the DFA in Definition 18, the Goal MDP $\mathcal{M} = (X, A, b_0, \mathbf{T}, X_G, c)$ embedded in the POMDP \mathcal{P} in that definition is extracted and then an optimal policy for \mathcal{M} is solved. An optimal policy $\pi^{''*}$ for the MDP is a function over $X = S \times Q$, which is computed via the Bellman equations

$$V^{''*}(x) = \min_{a \in A} (c(x, a) + \sum_{x' \in X} V^{''*}(x') \mathbf{T}(x, a, x')), \quad (5.7)$$

$$\pi^{''*}(x) = \arg \min_{a \in A} (c(x, a) + \sum_{x' \in X} V^{''*}(x') \mathbf{T}(x, a, x')). \quad (5.8)$$

This section provided an algorithm to solve RTM and the two variants of it, RTM/FOM and RTM/FHM, assuming that there exists a policy that can guarantee that a desired story will be captured. However, for some input DFAs and event models to the problem, no such policy exists. Therefore, before using the algorithm in this section, one might want to check whether such a policy exists or not. The next section discusses how to answer that decision problem.

Deciding if a Policy Exists

In this section, we discuss how to check if there exists a policy that guarantees, for any execution of the event model, a story within the language of the DFA will be captured.

We first consider the RTM/FOM problem, recalling that we can compute a policy for the Goal MDP underlying the Goal POMDP in Definition 18 rather than the Goal POMDP itself. To provide an answer to the decision question, we can check

whether or not there exists a policy for the MDP that guarantees that the goal states are reachable with probability 1. To find one, we check if there exists any policy that avoids the MDP’s *dead-end* states, namely those states from which no goal state is reachable. If no policy can avoid these dead-ends, then no policy will ensure that a desired story will be captured.

To illustrate, consider Figure 5.2a, which shows a simple DFA specifying all stories that start with e_1 . Figure 5.2b shows a simple event model in which e_1 happens at s_1 and e_2 occurs at s_2 . Figure 5.2c shows the Goal MDP obtained from the product of this DFA and event model. In this example, no policy guarantees that a desired story will be captured with probability 1 because no policy assures that the goal states of the MDP are almost surely reachable. Should a policy choose event e_1 to capture in the first time step, then the probability that a desired story will be captured by this policy is 0.6, which happens when the event model enters state s_1 in the next time step. Similarly, if a policy chooses event e_2 in the same configuration, then the probability that a desired story is captured is again 0.6 if the policy chooses event e_1 when the DFA is in state q_0 and the event model is in state s_1 . Observe that the Goal MDP constructed for this DFA and event model, which is shown in Figure 5.2c, has dead-end states (q_0, s_2) and (q_2, s_2) that are unavoidable.

Where there exists no policy that can guarantee a story will be captured, several strategies can be used to compute a reasonable policy that, nevertheless, may still be useful in practice. Generally, these maintain some balance between maximizing the probability of reaching a goal state and minimizing the expected number of steps to reach these goal states. See the discussion of [69] for several such options.

An alternative strategy, and one specific to our context, would be to alter the specification by expanding the language described by the DFA. One seeks to modify the DFA to obtain a language which will ensure the existence of some policy guaranteeing a story be captured almost surely. One desires, naturally, that the altered

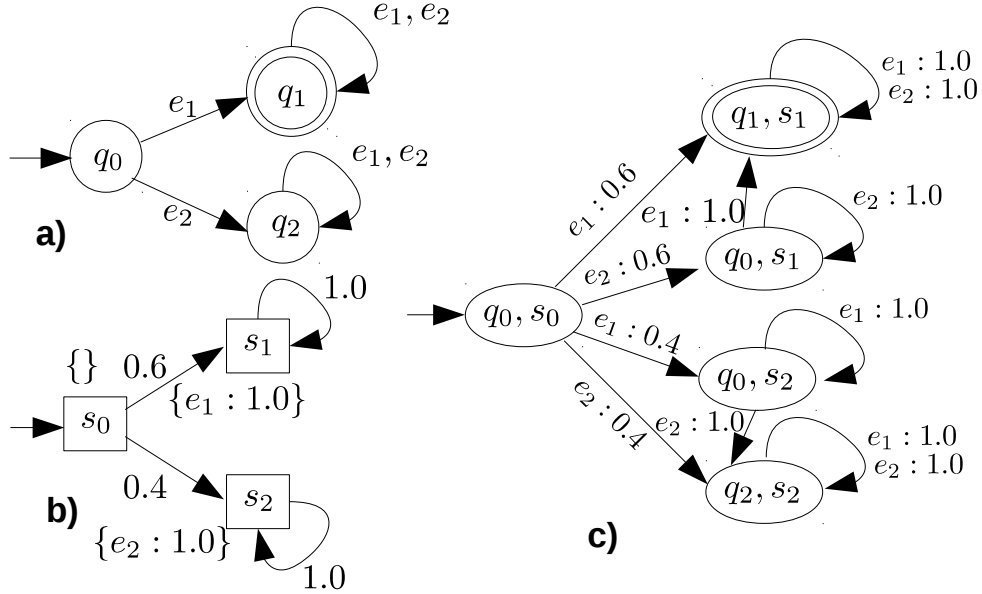


Figure 5.2: **a)** A DFA specifying all stories that contain at least an event over the event set $\{e_1, e_2\}$. **b)** A sample event model, in which event e_1 happens at s_1 and event e_2 happens at s_2 . **c)** The Goal MDP obtained from the product of the DFA and the event model in parts (a) and (b). component.

DFA be “close” to the original in terms of some metric for the distance between two DFAs. In Section 5.6, we examine one such metric, the Levenshtein distance, and discuss how, given a DFA, to construct another that is within a desired Levenshtein distance.

Note, however, that the existence of dead-ends does not mean that no policy ensures that the robot is able to capture a desired story under any execution of the event model. If all the dead-ends are avoidable and some goal states are almost surely reachable, then there will exist such a policy. For illustration consider the example in Figure 5.3. In this example, the MDP, part (c) of the figure, has only a single dead-end state (q_2, s_1) . But this dead-end is avoidable because the policy can avoid taking action b in state (q_0, s_0) . Doing so, the probability of reaching this dead-end becomes zero, ensuring that all executions under this policy always end in goals. For

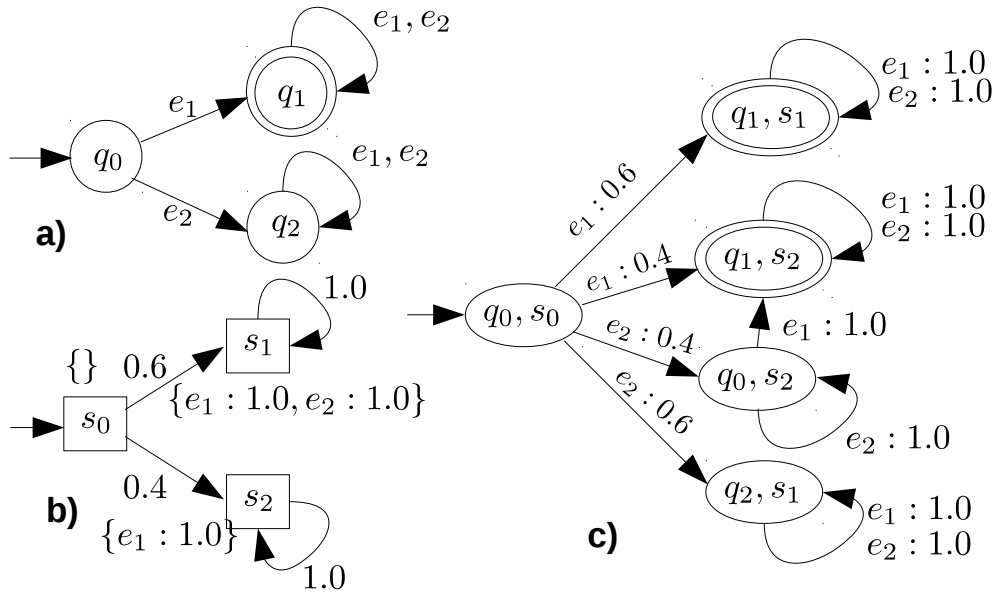


Figure 5.3: **a)** A DFA specifying all stories that contain at least an event over the event set $\{e_1, e_2\}$. **b)** A sample event model, in which event e_1 happens at s_1 and s_2 while event e_2 happens at s_1 . **c)** The Goal MDP obtained from the product of the DFA and the event model in parts (a) and (b). component.

the standard value iteration method to work, we first preprocess the MDP to identify dead-end states. The value of each dead-end is infinite as no goal state from that dead-end is reachable. For more discussions about MDPs with dead-ends, including algorithms to detect dead-ends and strategies to deal with them, we refer the reader to [11, 66, 69, 70, 82]. In our breadth-first-like implementation, we search backward from the goal states, finding all states that are reachable from those states, avoiding dead-ends found so-far.

For RTM and RTM/FOM, to determine whether there exists any policy guaranteeing that a desired story will be captured, we need to ascertain, for the Goal POMDP in Definition 18, whether there exists a policy whose execution ends in the goal states with probability 1 or not. One must checked whether any belief state in the reachable part of the (infinite) belief MDP has an unavoidable dead-end belief

state. In this case, we treat a belief state to be a dead-end if it arises from a dead-end state. This belief MDP has an infinite state space and we cannot deal with it directly, but instead one forms the finite support-belief MDP [62], whose states are the support for belief states. Note that, in general, the size of this support-belief MDP is exponential in the size of the MDP underlying the POMDP.

5.3 REPRESENTATION-INVARIANCE OF EXPECTED TIME

The event selected by the policy π^* at each step depends, in part, on the current state of the specification DFA. Because a single regular language may be represented with a variety of distinct DFAs with different sets of states—and thus, their optimal policies cannot be identical—one might wonder whether the expected execution time achieved by their computed policies depends on the specific DFA, rather than on the language. The question is particularly relevant in light of the language mutators we examine in Section 5.6. Here, we show that the expected number of steps required to capture a story within a given event model does indeed depend only on the language specified by the DFA, and not on the particular representation of that language.

For a DFA $\mathcal{D} = (Q, E, \delta, q_0, F)$, we define a function $f : Q \rightarrow \{0, 1\}$ such that for each $q \in Q$, $f(q) = 1$ if $q \in F$, and otherwise, $f(q) = 0$. Now consider the well-known notion of bisimulation, defined as follows:

Definition 19. [bisimulation] Given DFAs $\mathcal{D} = (Q, E, \delta, q_0, F)$ and $\mathcal{D}' = (Q', E, \delta', q'_0, F')$, a relation $R \subseteq Q \times Q'$ is a *bisimulation relation* for $(\mathcal{D}, \mathcal{D}')$ if for any $(q, q') \in R$:

- (1) $f(q) = f'(q')$; (2) for any $e \in E$, $(\delta(q, e), \delta'(q', e)) \in R$.

Bisimulation implies language equivalence and vice versa.

Proposition 1. [122] For two DFAs $\mathcal{D} = (Q, E, \delta, q_0, F)$, $\mathcal{D}' = (Q', E, \delta', q'_0, F')$, we have $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{D}')$ iff $(q_0, q'_0) \in R$ for a bisimulation relation R for $(\mathcal{D}, \mathcal{D}')$.

Bisimulation is preserved for any reachable pairs.

Proposition 2. If (q, q') are related by a bisimulation relation R for $(\mathcal{D}, \mathcal{D}')$, then for any $r \in E^*$, $(\delta^*(q, r), \delta'^*(q', r)) \in R$.

We now define a notion of equivalence for a pair of belief states.

Definition 20. [equivalence of belief states] Given an event model $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$, an observation model $\mathcal{B} = (Y, h)$ for \mathcal{M} , DFAs $\mathcal{D} = (Q, E, \delta, q_0, F)$ and $\mathcal{D}' = (Q', E, \delta', q'_0, F')$ such that $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{D}')$, let $\mathcal{P}_{(\mathcal{M}, \mathcal{B}; \mathcal{D})} = (X, A, b_0, \mathbf{T}, X_G, Z, \mathbf{O}, c)$ and $\mathcal{P}'_{(\mathcal{M}, \mathcal{B}; \mathcal{D}')} = (X', A, b'_0, \mathbf{T}', X'_G, Z, \mathbf{O}', c')$. For two reachable belief states $b \in \Delta(X)$ and $b' \in \Delta(X')$, with $\beta(b) = (d, q)$ and $\beta'(b') = (d', q')$, we say that b' is *equivalent* to b , denoted $b \equiv b'$, if (1) (q, q') are related by a bisimulation relation for $(\mathcal{D}, \mathcal{D}')$ and that (2) $d = d'$, i.e. for each $s \in S$, $d(s) = d'(s)$.

Equivalence is preserved for updated belief states.

Lemma 10. Given the structures in Definition 20, let $b \in \Delta(X)$ and $b' \in \Delta(X')$ be two reachable belief states such that $b \equiv b'$. For any action $a \in A$ and observation $z \in Z$, it holds that $b_z^a \equiv b'_z{}^a$ and that $Pr(z|a, b) = Pr(z|a, b')$.

Proof. Let $b_2 = b_z^a$ and $b'_2 = b'_z{}^a$, and assume $\beta(b_2) = (d_2, q_2)$ and $\beta'(b'_2) = (d'_2, q'_2)$. The case where b and b' are both goal belief states, that is, where $f(q) = f'(q') = 1$ is immediately implied from the fact that the goal states of the POMDPs from in Definition 18 are absorbing, i.g., $b_2 = b$ and $b'_2 = b'$. Therefore, we consider the case where $f(q) = f'(q') = 0$. Let b_a and b'_a be respectively the belief states resulted from doing action a (but before any observation) at belief states b and b' . These belief states are computed by the following formulas:

$$b_a(t) = \sum_{x \in X} \mathbf{T}(x, a, t)b(x), \quad (5.9)$$

and

$$b'_a(t') = \sum_{x' \in X'} \mathbf{T}'(x', a, t') b'(x'). \quad (5.10)$$

Given that $\beta(b) = (d, q)$, belief state b can have outcomes only for those states that are among $x_1 = (s_1, q), x_2 = (s_2, q), \dots, x_n = (s_n, q)$, and similarly, b' can have outcomes only for states that are among $x'_1 = (s_1, q'), x'_2 = (s_2, q'), \dots, x'_n = (s_n, q')$, where $n = |S|$. Also, because it is assumed that $d = d'$, for each integer $1 \leq j \leq n$, $b((s_j, q)) = b'((s_j, q'))$, or in other words, $b(x_j) = b'(x'_j)$. Now, with (5.9) and given the construction of the transition probability function \mathbf{T} in Definition 18, the only states for which b_a can have outcomes are among $t_1 = (s_1, q), t_2 = (s_2, q) \dots, t_n = (s_n, q)$ and $t_{n+1} = (s_1, \delta(q, a)), t_{n+2} = (s_2, \delta(q, a)), \dots, t_{2n} = (s_n, \delta(q, a))$. Similarly, the only states for which b'_a can have outcomes are among $t'_1 = (s_1, q'), t'_2 = (s_2, q') \dots, t'_n = (s_n, q')$ and $t'_{n+1} = (s_1, \delta'(q', a)), t'_{n+2} = (s_2, \delta'(q', a)), \dots, t'_{2n} = (s_n, \delta'(q', a))$.

Now, we claim that for each integer $1 \leq k \leq 2n$, $b_a(t_k) = b'_a(t'_k)$. To prove this, consider that by assumption, q and q' are related by a bisimulation relation for $(\mathcal{D}, \mathcal{D}')$, and this, by Definition 9, means that $q \in F$ iff $q' \in F'$. As a result, by the construction of the transition probability function in Definition 18, for each pair of integers $1 \leq i, j \leq n$, $\mathbf{T}((s_j, q), a, (s_i, q)) = \mathbf{T}'((s_j, q'), a, (s_i, q'))$ and $\mathbf{T}((s_j, q), a, (s_i, \delta(q, a))) = \mathbf{T}'((s_j, q'), a, (s_i, \delta'(q', a)))$, which together mean that for integers $1 \leq j \leq n$ and $1 \leq k \leq 2n$, $\mathbf{T}(x_j, a, t_k) = \mathbf{T}'(x'_j, a, t'_k)$. We use this, the assumption that $b(x_j) = b'(x'_j)$ for all $1 \leq j \leq n$, (5.9) and (5.10) to prove our claim as follows:

$$\begin{aligned} b_a(t_k) &= \sum_{1 \leq j \leq n} \mathbf{T}(x_j, a, t_k) b(x_j) \\ &= \sum_{1 \leq j \leq n} \mathbf{T}'(x'_j, a, t'_k) b'(x'_j) = b'_a(t'_k). \end{aligned} \quad (5.11)$$

To prove that $Pr(z|a, b) = Pr(z|a, b')$, consider the following formulas:

$$Pr(z|a, b) = \sum_{t \in X} \mathbf{O}(a, t, z) b_a(t), \quad (5.12)$$

and

$$Pr(z|a, b') = \sum_{t' \in X'} \mathbf{O}'(a, t', z) b'_a(t'). \quad (5.13)$$

By the construction of the observation function in Definition 18 and that $q \in F \iff q' \in F'$, it follows that for each integer $1 \leq j \leq n$, $\mathbf{O}(a, (s_j, q), z) = \mathbf{O}'(a, (s_j, q'), z)$.

Similarly, for each integer $1 \leq j \leq n$, $\mathbf{O}(a, (s_j, \delta(q, a)), z) = \mathbf{O}'(a, (s'_j, \delta'(q', a)), z)$.

Together, these two mean that for each integer $1 \leq k \leq 2n$, $\mathbf{O}(a, t_k, z) = \mathbf{O}'(a, t'_k, z)$.

This, combined with (5.11), proves a part of the lemma as follows:

$$\begin{aligned} Pr(z|a, b) &= \sum_{t \in X} \mathbf{O}(a, t, z) b_a(t) \\ &= \sum_{1 \leq k \leq 2n} \mathbf{O}(a, t_k, z) b_a(t_k) \\ &= \sum_{1 \leq k \leq 2n} \mathbf{O}'(a, t'_k, z) b'_a(t'_k) \\ &= \sum_{t' \in X'} \mathbf{O}'(a, t', z) b'_a(t') \\ &= Pr(z|a, b'). \end{aligned} \quad (5.14)$$

To prove that $b_2 \equiv b'_2$, consider that for each $x \in X$ and $x \in X'$, $b_2(x)$ and $b'_2(x')$ are computed as follows:

$$b_2(x) = \mathbf{O}(a, x, z) b_a(x) / Pr(z|a, b), \quad (5.15)$$

and

$$b'_2(x') = \mathbf{O}'(a, x', z) b'_a(x') / Pr(z|a, b'). \quad (5.16)$$

These two formulas combined with (5.11) and (5.14), and the fact that $\mathbf{O}(a, t_k, z) = \mathbf{O}'(a, t'_k, z)$ for all $1 \leq k \leq 2n$, imply that $b_2(t_k) = b'_2(t'_k)$ for all $1 \leq k \leq 2n$. Therefore, $d_2 = d'_2$. We now only need to prove that q_2 and q'_2 are related by a bisimulation

relation for $(\mathcal{D}, \mathcal{D}')$. Observe that if the robot's prediction of occurring event a was wrong, then $q_2 = q$ and $q'_2 = q'$, and otherwise, $q_2 = \delta(q, a)$ and $q'_2 = \delta'(q', a)$. In the former case, by definition, q_2 and q'_2 are related by a bisimulation relation R for $(\mathcal{D}, \mathcal{D}')$, and in the later case, by Proposition 2, q_2 and q'_2 are related by the same bisimulation relation q and q' were related by. Thus, we conclude $b_z^a \equiv b'_z{}^a$. \square \square

Note that for a Goal POMDP \mathcal{P} with initial belief state b_0 , $V^*(b_0)$ is the expected cost of reaching a goal belief state via an optimal policy for \mathcal{P} . We now present our result.

Theorem 14. For the structures in Definition 20, it holds that $V^*(b_0) = V'^*(b'_0)$.

Proof. For a belief MDP \mathcal{M} , let $\text{Tree}(\mathcal{M})$ to be its tree-unravelling—the tree whose paths from the root to the leaf nodes are all possible paths in \mathcal{M} that start from the initial belief state. A policy π for \mathcal{M} chooses a fixed set of paths over $\text{Tree}(\mathcal{M})$, and the expected cost of reaching a goal belief state under π is equal to

$$\sum_{p \in \text{GoalPaths}(\pi, \text{Tree}(\mathcal{M}))} C(p)W(p),$$

where $\text{GoalPaths}(\pi, \text{Tree}(\mathcal{M}))$ is the set of all paths that are chosen by π and reach a goal belief state from the root of $\text{Tree}(\mathcal{M})$, $C(p)$ is the sum of costs of all transitions in path p , and $W(p)$ is the product of the probability values of all transitions in p . The idea is that if we can overlap the tree-unravellings of the belief MDPs $\mathcal{P}_{(\mathcal{M}, \mathcal{B}; \mathcal{D})}$ and $\mathcal{P}'_{(\mathcal{M}, \mathcal{B}; \mathcal{D}')}$ in such a way that each pair of overlapped belief states are equivalent in the sense of Definition 20 and that each pair of overlapped transitions have the same probability and the same cost, then for each pair of overlapped belief states $b \in \Delta(X)$ and $b' \in \Delta(X')$, if we use $\pi^*(b)$ as the decision at the belief state b' then, because those fixed paths are overlapped, we know $V^*(b_0) \geq V'^*(b'_0)$. And, in a similar fashion, $V^*(b_0) \leq V'^*(b'_0)$, and thus, $V^*(b_0) = V'^*(b'_0)$. The following construction makes those trees and shows how they can be overlapped.

For an integer $n \geq 1$, we can make two trees T_n and T'_n as follows: (1) Set b_0 as the root of T_n and set b'_0 as the root of T'_n ; make a relation R and set $R \leftarrow \{(b_0, b'_0)\}$. (2) While $|T_n| < n$, extract a pair (b, b') from R that has not been checked yet and in which b and b' are not goal belief states; for each action a and observation z , compute b_z^a and $b'_z{}^a$, add node b_z^a and edge (b, b_z^a) to T , and add node $b'_z{}^a$ and edge $(b', b'_z{}^a)$ to T' ; label both edges (a, z) . Also assign to edge (b, b_z^a) , $Pr(z|a, b)$ as its probability value, and set the probability value of $(b', b'_z{}^a)$, $Pr(z|a, b')$; the cost of each edge is set 1. Finally, add $(b_z^a, b'_z{}^a)$ to R .

Given that $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{D}')$, by Proposition 1, states q_0 and q'_0 are related by a bisimulation relation for $(\mathcal{D}, \mathcal{D}')$, which (by Definition 20 and the construction in Definition 18) implies that $b_0 \equiv b'_0$. This combined with Lemma 10 implies that for each pair $(b, b') \in R$, $b \equiv b'$. We now match T_n and T'_n so that each pair (b, b') that are related by R overlap. By Lemma 10, each pair of overlapped edges have the same probability value and the same cost value. Since for any integer $n \geq 0$ we can overlap trees T_n and T'_n in the desired way, we can overlap the tree-unravellings of the belief MDPs of $\mathcal{P}_{(\mathcal{M}, \mathcal{B}; \mathcal{D})}$ and $\mathcal{P}'_{(\mathcal{M}, \mathcal{B}; \mathcal{D}')}$ in the desired way too; this completes the proof. \square

The upshot of this analysis is that we need attend only to the story specification language (given indirectly via \mathcal{D}), the specific presentation of that language does not impact the expected number of steps to capture an event sequence satisfying that specification.

5.4 FASTER ALGORITHMS FOR SPECIAL STRUCTURES

In this section, we consider several special cases of event models and DFAs for which new algorithms are feasible. Though less general than the approaches introduced in the prior sections, these new algorithms exploit the structure of the special cases to run significantly faster.

The DFA is loop-omitted acyclic

The first case is when the DFA adheres to the following requirement.

Definition 21. A DFA $\mathcal{D} = (Q, E, \delta, q_0, F)$ is called a *loop-omitted acyclic DFA* if for every state $q \in Q$ and string $\eta \in E^*$ for which $\delta^*(q, \eta) = q$, it holds for every prefix η' of η that $\delta^*(q, \eta') = q$.

Intuitively, a DFA is a loop-omitted acyclic DFA if it does not have any cycle except self-loops on the states. We speculate that many DFAs of interest for this sort of problem will have this property. In fact, all the DFAs in this chapter's case studies are loop-omitted acyclic DFAs; likewise for the case studies in our other extensions to this work [21, 22]

For this kind of DFA, regardless of the form of the event model, the graph underlying the Goal POMDP in Definition 18 will be a layered directed acyclic graph where each layer is, in fact, a strongly connected component (SCC). Note that all states $x = (q, s)$ within a single SCC share a single DFA state q and all those states represent a situation where either the event predicted by the robot does not happen in the next time step, making the DFA stay in the same state q , or the predicted event did happen but state q transitions back to itself with that predicted event.

For an example, see the DFA in Figure 5.4a and the event model in Figure 5.4b. The set of possible events consists of e_1 and e_2 . Event e_1 happens with probability 1 at state s_1 , while event e_2 happens with probability 1 at state s_2 , and at each state of the event model no more than one event happens. Figure 5.4c shows the state space and the transition function of the Goal POMDP constructed from the product of the DFA and the event model based on Definition 18. This product, which is decomposed into its set of SCCs in Figure 5.4d, has five SCCs C_0, C_1, C_2, C_3 , and C_4 . There is only a single topological ordering of these SCCs, namely $(C_0, C_1, C_3, C_2, C_4)$.

Now, we consider solving the RTM/FOM problem where the input DFA is loop-omitted acyclic. Recall that to solve that problem we need to compute a policy that minimizes the expected number of steps to reach a goal state, and observe that, in this case, the Goal MDP underlying the Goal POMDP in Definition 18 is a layered DAG. Thus, to compute an optimal policy for such a Goal MDP, we can use the *topological value iteration algorithm* of [30].

Their algorithm considers each SCC as a *metastate* and then computes an optimal policy for those metastates based on a reverse ordering of a topological ordering of the metastates. The optimal action for states within each metastate is computed using value iteration. In Dai and Goldsmith’s algorithm, each metastate is solved only once because the graph connecting the metastates is acyclic. Note that when a metastate (SCC) is solved, only the values of states within that metastate are backed up, whereas in the classical value iteration, at each step, the values of all states are backed up until their values converge. To illustrate their algorithm, consider again the Goal MDP in Figure 5.4c. Recall that there was only one topological ordering, $(C_0, C_1, C_3, C_2, C_4)$, between the SCCs of the MDP. Accordingly, their algorithm first computes an optimal action for the single state in C_4 , then for the states in C_2 , then for the states in C_3 , then for the states in C_1 , and finally for the only state of C_0 .

For solving the RTM and the RTM/FHM problems with input DFAs which are loop-omitted acyclic, we need to solve a Goal POMDP that has a layered DAG structure, and for solving those Goal POMDPs, one can use the algorithm of [34], which computes a policy using a point-based method. Their algorithm constructs the layered acyclic graph for the belief points by utilizing the layered acyclic structure of the POMDP.

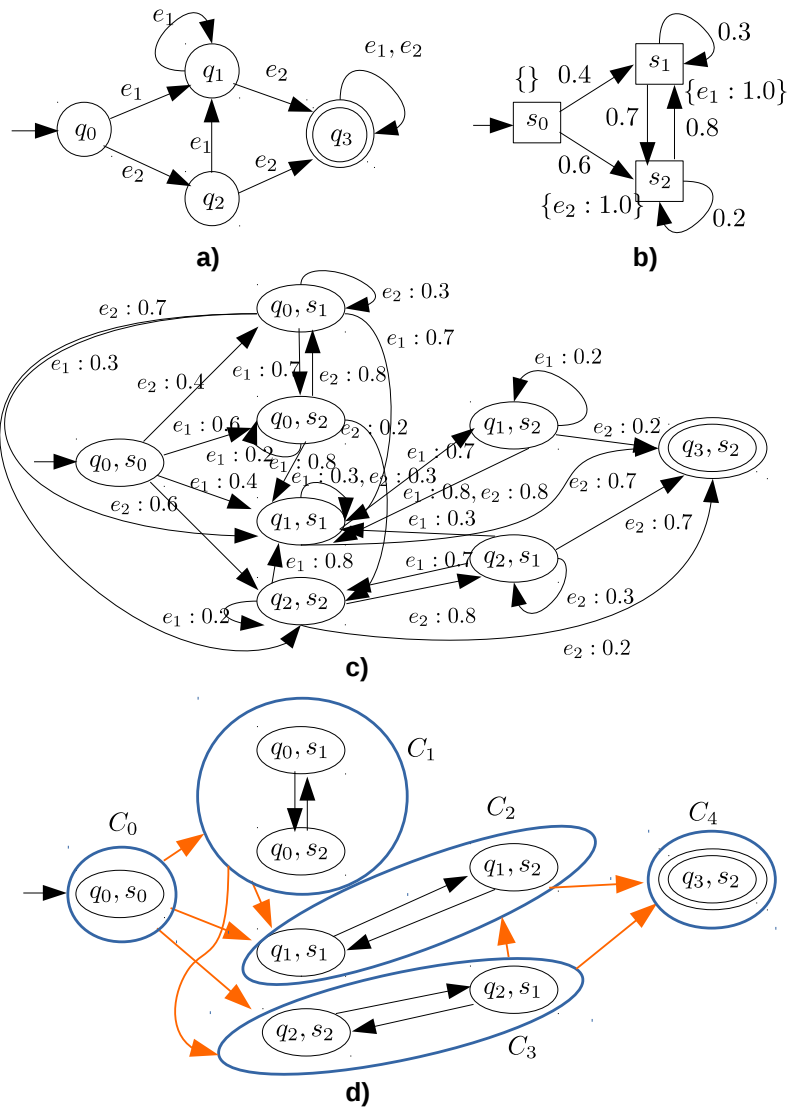


Figure 5.4: **a)** A loop-omitted acyclic DFA. **b)** A sample event model. **c)** The Goal MDP obtained from the product of the DFA and the event model in parts (a) and (b). **d)** It shows the strongly connected components of graph underlying the MDP in part c. Each blue circle is a strongly connected component. The self-loops and the edges between states of different SCCs have been omitted to reduce visual clutter.

The Goal MDP is a directed acyclic graph

Another special case is where the graph underlying the Goal MDP is a DAG if all the self-loops are removed from it.

We clarify this special case in the following definition.

Definition 22. For a given Goal MDP $\mathcal{M} = (X, A, b_0, \mathbf{T}, X_G, c)$, let $\mathbf{G}(\mathcal{M}) = (V, E)$ be the graph underlying \mathcal{M} , in which $V = X$ and $E = \{(x, t) \in X^2 \mid \mathbf{T}(x, t) > 0\}$, and let $\mathbf{Loop}^-(\mathbf{G}(\mathcal{M})) = (V, E')$ be the graph obtained from $\mathbf{G}(\mathcal{M})$ by removing the self-loops from it, that is, $E' = E \setminus \{(x, x) \mid x \in X\}$. We say that \mathcal{M} is a *loop-omitted directed acyclic graph*, or loop-omitted DAG for short, if $\mathbf{Loop}^-(\mathbf{G}(\mathcal{M}))$ is a directed acyclic graph.

Note that this is stronger than the previous case, as it corresponds to the circumstance where each SCC is a single vertex. Nevertheless, it is possible for a Goal MDP to be a loop-omitted DAG, while its underlying graph has self-loops. Figure 5.5 provides one such example. This kind of MDPs arises, in particular, in applications where the DFA specifying all desired stories is a loop-omitted acyclic DFA and the graph underlying the event model is also a DAG if the self-loops removed from that graph. A special case of that kind of event model is where the event model has only a single state, which might be created by collapsing all the states of an original event model in order to make an approximation of the original event model.

In the rest of this section, we only consider RTM/FOM problems for this kind of MDP. Observe that because any Goal MDP that is a loop-omitted DAG is a layered DAG where each strongly connected component of the graph underlying the Goal MDP has only one state, we can use the algorithm of the previous section, the topological value iteration algorithm for MDPs, to compute an optimal policy for the MDP. This requires iteration to update the value of a state using the Bellman equation until the value of the state converges. Though this algorithm is faster than the original value iteration algorithm for MDPs, for solving large MDPs, it still may require a considerable amount of time for the state values to converge, and therefore, we propose a faster algorithm that does not require value iteration at all, and the

Bellman equation for each state is solved by solving several simple, single-variable equations.

In this algorithm we first choose a topological ordering of the non-goal states (state values of all the goal states are zero) of the MDP. Then, at each step, we take a state from the MDP based on the reverse of the topological ordering to compute an optimal action for that state and the value of that state. To do so, for each state x and action $a \in A$, we introduce a variable $t_{x,a}$ to denote the expected number of steps from x to reach a goal state of the MDP if the policy assigns action a to state x . Also, for each state x , we introduce a variable t_x to denote the expected number of steps to reach a goal state from x under an optimal policy. To compute the optimal action for each state x , if x is a goal state, then $t_x = 0$, meaning that the expected number of steps to reach a goal state from x under an optimal policy is zero. If state x is not a goal state, then for each action $a \in A$, we solve the following equation

$$t_{x,a} = \mathbf{T}(x, a, x)(1 + t_{x,a}) + \sum_{\substack{x' \in X \\ x' \neq x}} \mathbf{T}(x, a, x')(1 + t_{x'}). \quad (5.17)$$

Then, we compute t_x simply as $t_x = \min_{a \in A} \{t_{x,a}\}$. For each $x \in X \setminus X_G$, we set $\pi^*(x) = \arg \min_{a \in A} \{t_{x,a}\}$. By doing so, we compute an optimal policy π^* .

We end this section by illustrating this algorithm via an example for the MDP in Figure 5.5. There is a single topological ordering of the non-goal states: $x_0 \rightarrow x_1 \rightarrow x_2$. To compute an optimal policy, we pick this (the sole) choice of ordering. Next, we compute the optimal action for x_2 . For this purpose, for actions a and b , we need to solve the following equations,

$$t_{x_2,a} = 0.2(1 + t_{x_2,a}) + 0.8(1 + t_{x_4}) \quad (5.18)$$

and

$$t_{x_2,b} = 0.4(1 + t_{x_2,a}) + 0.6(1 + t_{x_4}). \quad (5.19)$$

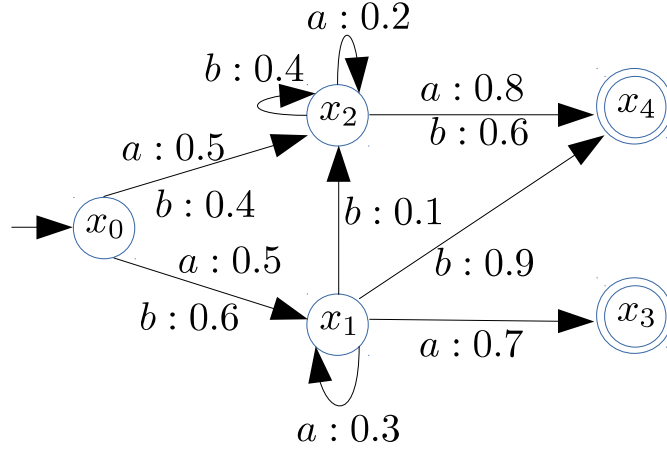


Figure 5.5: A sample of an MDP that is a directed acyclic graph when all the self-loops are removed.

Based on these two equations, we have $t_{x_2,a} = 1.25$ and $t_{x_2,b} = 1.67$. So, $t_{x_2} = 1.25$, and thus, we set $\pi^*(x_2) = a$. Then, we solve the following equations for x_1 .

$$t_{x_1,a} = 0.3(1 + t_{x_1,a}) + 0.7(1 + t_{x_3}) = 1 + 0.3t_{x_1,a} \quad (5.20)$$

and

$$t_{x_1,b} = 0.1(1 + t_{x_2}) + 0.9(1 + t_{x_4}) = 1.125. \quad (5.21)$$

So, $t_{x_1} = 1.125$, and hence, we set $\pi^*(x_1) = b$. To compute the optimal action for x_0 , we solve the following equations,

$$t_{x_0,a} = 0.5(1 + t_{x_2}) + 0.5(1 + t_{x_1}) \quad (5.22)$$

and

$$t_{x_0,b} = 0.4(1 + t_{x_2}) + 0.6(1 + t_{x_1}). \quad (5.23)$$

As such, $t_{x_0} = 2.175$, and therefore, we let $\pi^*(x_0) = b$.

5.5 GREEDY ALGORITHM

In this section, we consider a greedy algorithm for solving RTM and RTM/FHM, which we also specially adapt for RTM/FOM. This greedy algorithm will serve as a baseline for comparison for the case studies in Section 5.7.

The idea is, at each time step, simply to choose an event to capture that has the highest probability of occurring in the next time step. To do so, the robot first uses the event model $\mathcal{M} = (S, \mathbf{P}, s_0, E, g)$ and the DFA $\mathcal{D} = (Q, E, \delta, q_0, F)$, to construct the Goal POMDP $\mathcal{P}_{(\mathcal{M}, \mathcal{B}; \mathcal{D})} = (X, A, b_0, \mathbf{T}, X_G, Z, \mathbf{O}, c)$ based on the construction in Definition 18. Then, at each time step, it uses this POMDP to compute an event that has the highest probability to be chosen in the next time step. Importantly, in this greedy approach, the robot considers only a single step, and does not compute a policy that minimizes the expected number of steps to reach a goal state for this POMDP.

For each time step k , the robot maintains the current belief state $b_k \in \Delta(X)$ of the POMDP and the current state q_k of the DFA. Then, for all events e for which $\delta(q_k, e) \neq q_k$ and from $\delta(q_k, e)$ at least one accepting state is reachable, we compute the probability that e happens in the next time step given b_k as follows:

$$\begin{aligned} Pr(e \text{ happens in the next time step} \mid b_k) = \\ \sum_{x=(s,q) \in X} b_k[x] \cdot \sum_{s' \in S} \mathbf{P}(s, s') g(s', e). \end{aligned} \quad (5.24)$$

Hence, the robot hopes to record in the next time step, an event that has the greatest probability computed by this equation. In the case that several such events exist, the robot chooses one of them arbitrarily.

Given that RTM/FOM is a special form of RTM, the process described so far in this section applies for RTM/FOM too, but for RTM/FOM we can avoid constructing the product of the event model and the DFA. For RTM, the robot maintains, at each time step k , the current state s_k of the event model and the

current state q_k of the DFA. Then, at step k , it computes for all events e for which $\delta(q_k, e) \neq q_k$ and from $\delta(q_k, e)$ an accepting state is reachable, the following probability

$$\begin{aligned} Pr(e \text{ happens in the next time step} \mid s_k) = \\ \sum_{s' \in S} \mathbf{P}(s_k, s')g(s', e), \end{aligned} \quad (5.25)$$

which is essentially the probability that e happens in the next time step. Then, from among all events that obtain the highest value in this equation, the robot chooses one to attempt to record in the next time step.

We compare this greedy algorithm with our general algorithm in Section 5.7 to assess their relative solution quality.

5.6 CONSTRUCTION OF SPECIFICATION LANGUAGES

This section describes how one might construct, in a partially automated way, specifications for a variety of interesting scenarios. The idea is to use a variety of mutators to construct specification DFAs.

Multiple recipients

Suppose we would like to capture several videos, one for each of several recipients, within a single execution. Given language specifications $\mathcal{D}_1, \dots, \mathcal{D}_n \in \mathcal{D}$, where \mathcal{D} denotes the set of all DFAs over a fixed event set E , how can we form a single specification that directs the robot to capture events that can be post-processed into the individual output sequences? One way is via two relatively simple operations on DFAs:

(M_S) A *supersequence* operation $\mathbf{M}_S : \mathcal{D} \rightarrow \mathcal{D}$, where

$$\mathcal{L}(\mathbf{M}_S(\mathcal{D})) = \{w' \in E^* \mid w' \text{ is supersequence of a } w \in \mathcal{L}(\mathcal{D})\}.$$

This operation is produced by first treating \mathcal{D} as a nondeterministic finite automaton (NFA) and then, for each event and state, adding a transition labeled by that event from that state to itself, and converting result back into a DFA [105].

(M_I) An *intersection* operation $\mathbf{M}_I : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$, under which

$$\mathcal{L}(\mathbf{M}_I(\mathcal{D}_1, \mathcal{D}_2)) = \mathcal{L}(\mathcal{D}_1) \cap \mathcal{L}(\mathcal{D}_2).$$

Based on these two operations, we can form a specification that asks the robot to capture an event sequence that satisfies all n recipients as follows:

$$\mathcal{D} = \mathbf{M}_I(\mathbf{M}_I(\mathbf{M}_S(\mathcal{D}_1), \mathbf{M}_S(\mathcal{D}_2)) \dots, \mathbf{M}_S(\mathcal{D}_n))$$

Then from any $\xi \in \mathcal{L}(\mathcal{D})$, we can produce a $\xi_i \in \mathcal{L}(\mathcal{D}_i)$ by discarding (as a post-production step) some events from ξ .

Mistakes were made

What should the robot do if it simply cannot capture an event sequence that fits its specification \mathcal{D} , either because some necessary events did not occur, or because the robot failed to capture them when they did occur? One possibility is to accept some limited deviation between the desired specification and what the robot actually captures.

Let $d : E^* \times E^* \rightarrow \mathbb{Z}^+$ denote the Levenshtein distance [80], that is, a distance metric that measures the minimum number of insert, delete, and substitute operations needed to transform one string into another. A mutator that allows a bounded amount of such distance might be:

(M_L) A *Levenshtein mutator* $\mathbf{M}_L : \mathcal{D} \times \mathbb{Z}^+ \rightarrow \mathcal{D}$ that transforms a DFA \mathcal{D} into one that accepts strings within a given distance from some string in $\mathcal{L}(\mathcal{D})$.

$$\mathcal{L}(\mathbf{M}_L(\mathcal{D}, k)) = \{\xi \mid \exists \xi' \in \mathcal{L}(\mathcal{D}), d(\xi, \xi') \leq k\}.$$

This mutation can be achieved using a *Levenshtein automaton* construction [71, 128]. Then, if the robot captures a sequence in $\mathcal{L}(\mathbf{M}_L(\mathcal{D}, k))$, it can be converted to a sequence in $\mathcal{L}(\mathcal{D})$ by at most k edits. For example, an insertion edit would perhaps require the undesirable use of alternative ‘stock footage’, rendering of appropriate footage synthetically, or simply a leap of faith on the part of the viewer. By assigning the costs associated with each edit appropriately in the construction, we can model the relative costs of these kinds of repairs.

At least one good shot

In some scenarios, there are multiple distinct views available of the same basic event. We may consider, therefore, scenarios in which this kind of good/better correspondence is known between two events, and in which the robot should endeavor to capture, say, at least one better shot from that class. We define a mutator that produces such a DFA:

(M_G) An *at-least-k-good-shots* mutator $\mathbf{M}_G : \mathcal{D} \times E \times E \times \mathbb{Z}^+ \rightarrow \mathcal{D}$, in which $\mathbf{M}_G(\mathcal{D}, e, e', k)$ produces a DFA in which e' is considered to be a superior version of event e , and the resulting DFA accepts strings similar to those in $\mathcal{L}(\mathcal{D})$, but with at least k occurrences of e replaced with e' .

The construction makes a DFA in which \mathcal{D} has been copied $k + 1$ times, each called a *level*, with the initial state at level 1 and the accepting states at level $k + 1$. Most edges remain unchanged, but each edge labeled e , at all levels less than $k + 1$, is augmented by a corresponding edge labeled e' that moves to the next level. This guarantees that e' has replaced e at least k times, before any accepting state can be reached.

5.7 CASE STUDIES

We also compare our general algorithm with a the greedy algorithm of Section 5.5, which, at each time step, attempts to capture an event that has the highest probability of occurrence at the next time step.

Turisti Oulussa

In pre-COVID 2019, William is a tourist visiting Oulu as shown in Figure 5.6a. William’s family has privately contracted a robotic videography company to record him seeing the sights, specifically the Kauppahalli (k), the Hupisaaret park (h), and either Tietomaa museum (t) or the Oulu Cathedral (c). The robot does not know William’s specific plans, but it does know, through some statistics, that a typical tourist moves among those districts according to the event model in Figure 5.6b.

The desired video is specified using the DFA in Figure 5.6c. The robot is given other tasks to do aside from recording William, and thus, cannot merely follow William; it must form a strategy that predicts which events to try to capture.

We considered three settings: (1) RTM/FOM: the robot always knows the current district in which William is located, perhaps by the help of some static sensors; (2) RTM: the robot does not know at which district William is currently located but there is a single useful observation, a message sent from a security guard in district s_1 , that informs the robot that William is in district s_1 whenever he is there; (3) RTM/FHM: the robot receives no direct knowledge about William’s location. We computed the optimal policy for RTM/FOM, case (1), using the Goal MDP approach in Section 5.2. According to this policy, the expected number of steps to record under an optimal policy with full observability, a story satisfying the specification, is approximately 35.39.

The computed optimal policy for this case is shown in Figure 5.6d. Each oval in

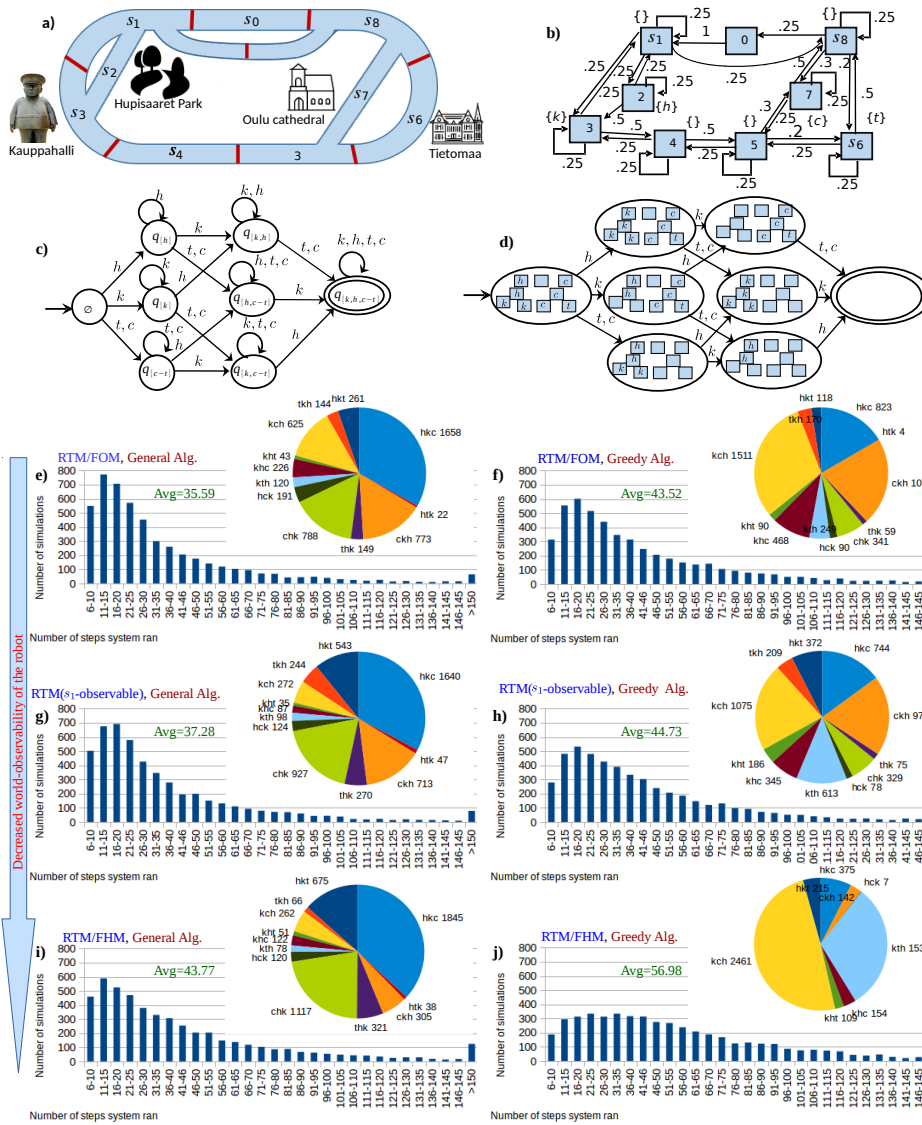


Figure 5.6: **a)** Districts of Oulu that William is touring. **b)** An event model that describes how a tourist visits those districts. **c)** A DFA specifying that the captured story must contain events k and h and at least one of c or t . **d)** The optimal policy for the RTM/FOM problem for Oulu. Subfigures **e–j** consist of a histogram showing, for 5,000 simulations, the distribution of the number of hours (steps) William (system) circulated (ran) until the robot recorded a story specified by the DFA, and a pie chart showing the distribution of recorded sequences in these simulations. Pairs **e)** and **f)** are for the RTM/FOM problem, **g)** and **h)** for the RTM problem, and **i)** and **j)** are for the RTM/FHM problem. In the left column the robot uses the general algorithm, while in the right column it uses the greedy approach. Moving down the column, the average number of steps to record a story increases as the robot’s perception of the world’s state diminishes. While the distribution of the recorded sequences by the general algorithm differs from the distribution of the recorded sequences by the greedy algorithm, for each of the greedy algorithm and the general algorithm, the distributions of the recorded sequences under different levels observability were similar.

this figure is a state of the DFA and inside each of those ovals, all the states of the event model are drawn as boxes. The event labeled inside a box is the event chosen by the optimal policy when the DFA is in the state represented by that oval and the event model is in the state represented by that box. Each empty box is assigned an arbitrary event by the policy and those events assigned to those empty boxes are irrelevant to recording a desired story.

To verify the correctness of the algorithm, we simulated the execution of this policy 5,000 times. In each simulation, William followed a random path through the city according to the event model in Figure 5.6b, and the robot executed the computed policy to capture an event sequence satisfying the specification. The average number of steps to record a satisfactory sequence for those 5,000 simulations using our general algorithm was 35.59, quite close to the expected number of steps. Figure 5.6e shows results of those simulations in form of a histogram and a pie chart. We also made 5,000 simulations of the same RTM/FOM problem and in each simulation we let the robot to use the greedy algorithm to record a desired story. The average number of steps for this experiment was 43.52, which is substantially longer than the expected number of steps to record a desired event sequence with the optimal policy for RTM/FOM, 35.59. This is justified, in particular, by the fact that when the robot is in state s_0 and it has captured neither h nor k , the greedy algorithm does not consider the fact that the best event to predict at that time to decrease the average number of steps is h because, if William enters s_2 in the next time step, then it is possible than he enters s_3 from s_2 and, thus, the robot could capture both events h and k during a single circuit of the environment. See Figure 5.6f for additional details regarding this experiment.

For cases (2) and (3), our algorithm constructed a Goal POMDP, as described by Definition 18, which is then supplied to APPL Online to perform 5,000 simulations. Also, for each of the two cases, we generated 5,000 simulations in our program and

let the greedy algorithm decide which event to try to capture. In case (2), RTM with a useful observation, the average number of steps to record a desired story using our general algorithm and the greedy algorithm were 37.28 and 44.73, respectively. In case (3), RTM/FHM, the general algorithm and the greedy algorithm had the robot record a desired story in 43.77 and 56.98 steps, respectively, on average.

The histograms and the pie charts for these four experiments are shown in Figures 5.6g–5.6j. In these figures, notice how a single observation of whether William is in s_1 helps the robot to record a story considerably faster than when it hasn't got access to that state information. To such a robot, even a stream of quite limited information, if aptly chosen, can be very useful.

A further qualitative remark: note how the histogram changes as the level of observability increases, from RTM/FOM to RTM/FHM: the robot is able to utilize the additional information to capture stories more rapidly. Also, across each of the three settings RTM/FOM, RTM, and RTM/FHM, the average number of steps needed via the greedy algorithm is considerably greater than via the general algorithm.

Wedding reception

Suppose a videographer robot is asked to produce videos that convey different stories, assembled from unpredictable events at a wedding reception. Each guest has their own sense of the events they would like to see captured: Alice is mainly interested in seeing Chris drinking or smoking, but also has plans to share the last dance with Bob; Bob cares for nothing but seeing his own dancing through the evening, but hopes to share the last dance with Alice; Chris does not care to see any events at all, but Chris's children are concerned about his unhealthy habits, and so if Chris is drinking too much coffee or smoking too much, they would like to know. The robot in that scenario is given three parallel objectives. We can formalize those as languages,

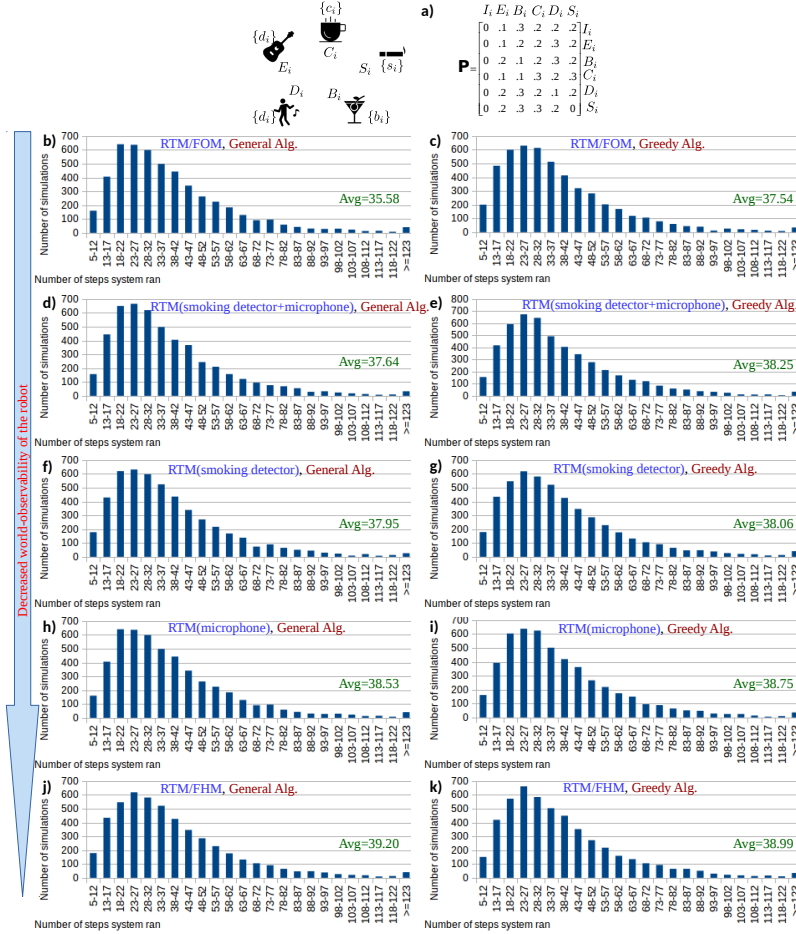


Figure 5.7: **a)** The event model for the behavior of a person attending a wedding reception, which has six states: I_i , the state of arriving; E_i , the state of being entertaining; C_i , for consuming coffee; B_i , for drinking other beverages; D_i , for dancing; and S_i , for smoking. Each histogram shows the average number of steps to record a desired story for 5,000 simulations of the wedding reception scenario. The left column, parts **b**, **d**, **f**, **h**, and **j**, is for the cases where the robot uses the general algorithm, while the right column, parts **c**, **e**, **g**, **i**, and **k** are for the cases where the robot uses the greedy algorithm. Parts **d** and **e** are for when there is a smoke detector, providing the observation of whether someone is smoking, and a microphone, capable of detecting that someone is dancing or being entertained. For parts **f** and **h**, there is only a smoke detector, while for parts **h** and **i** there is only a microphone.

It seems, at first, that the RTM problem with a smoke detector might be incompatible with RTM using a microphone, but the single useful observation in the former guarantees that at least one guest is in the state of smoking, while the single useful observation in the later case guarantees that at least one guest is either in state of dancing or in state of being entertained, which is less informative. This experiment also shows that increasing observability will decrease the time to capture a desired story. Furthermore, it shows that although the general algorithm often outperformed the greedy algorithm in terms of average number of steps, here the greedy algorithm gives a reasonable approximation to the optimal solutions.

shown here for compactness as regular expressions: for Alice, $r_1 = (s_3 + c_3)^+ d_{12}$; for Bob, $r_2 = (d_2 + d_{12} + d_{23})^+ d_{12}$; and for Chris, $r_3 = (s_3 + c_3)(s_3 + c_3)(s_3 + c_3)^+$. These three requests—where subscript labels 1, 2, and 3, respectively represent Alice, Bob, and Chris—are encoded using DFAs \mathcal{D}_1 , \mathcal{D}_2 , and \mathcal{D}_3 , respectively.

The joint behavior of the three guests is modeled by an event model \mathcal{M} obtained as the Cartesian product of the models for the individuals, which has 6^3 states in this example. The joint event model is further enhanced with joint events created from single events. To form a DFA \mathcal{D} from the given specification DFAs, the robot uses $\mathcal{D} = \mathbf{M_I}(\mathbf{M_I}(\mathbf{M_S}(\mathcal{D}_1), \mathbf{M_S}(\mathcal{D}_2)), \mathbf{M_S}(\mathcal{D}_3))$.

Our implementation for this case study considers five settings: (1) RTM/FOM: the current state of the event model is always observable to the robot, that is, the robot always knows what each of the guests are doing, (2) RTM with a smoke detector device and a microphone: the robot is not aware what each of the guests are doing, but there is a smoke detector that if at each time step tells if somebody is smoking or not but without telling who is exactly smoking, and there is a microphone whose being turned on means that someone is dancing or being entertained; (3) RTM with a smoke detector device: the robot does not know what each of the guests are doing, but using the smoke detector can detect if right now somebody is smoking or not; (4) RTM with a microphone: the robot is not aware about the current status of the guests but if the microphone is turned on, then it means that someone is dancing or being entertained; (5) RTM/FHM: the robot receives no direct information about the current behavior of the guests.

For each of these five settings, we conducted two experiments, each consisting of 5,000 simulations. In one experiment we let the robot use the general algorithm to record a desired story, while in the other one we let the robot use the greedy algorithm.

The expected number of steps for an optimal policy for RTM/FOM is 35.38, and

over the 5,000, simulations, the average number of steps to record a story using the general algorithm was 35.58; both numbers are very close. The average number of steps over 5,000 simulation using the greedy algorithm was 37.54, which shows that the greedy algorithm was also outperformed by the general algorithm in minimizing the number of steps to record for this case study.

The average number for RTM with a smoke detector and microphone using the general algorithm and the the greedy algorithm were 37.64 and 38.25, respectively. For RTM with a smoke detector, the average number was 37.95 when the general algorithm was used, and it was 38.06 when the greedy algorithm was used. The general algorithm and the greedy algorithm for RTM with a microphone respectively yielded 38.53 and 38.75. Finally, the average number of steps using the general algorithm for RTM/FHM was 39.20, while the average number using the greedy algorithm was 38.99.

Again we observed that increasing the robot's ability to perceive the world will help reduce the average number of steps to record a desirable event sequence. Also, for four out of the five considered settings, the general algorithm yielded a fewer average number of steps compared to the greedy algorithm, but for RTM/FHM, the greedy algorithm produced a slightly superior average number of steps. In this experiment, except for RTM/FOM, which we solve using an MDP rather than a POMDP, the expected number of steps for the greedy algorithm and for the general algorithm were close. This is perhaps because APPL Online, the tool we used for solving the POMDP, is an online POMDP solver and the solution is provide is an approximate solution rather than an exact solution, which is in general intractable to provide for POMDPs. This suggests this particular case study is an example where the greedy algorithm is able to closely approximate the optimal solutions.

Running a race

John and James are two runners that running a race against one another. A videographer robot is asked to record a video whose events involve John and James. The events of interest are: r_1 , John is running; h_1 , John is crossing the flag located the middle of the race field; f_1 , John is crossing the finish line; p_{12} , John is passing James; r_2 , James is running; h_2 , James is crossing the flag located the middle of the race course; f_2 , James is crossing the finish line; p_{21} , James is passing John.

To make an event model for this problem, we divide the racetrack into several sections of the identical length. Figure 5.8a shows an example in which the race is divided into 8 sections, s_0 through s_7 . To make an event model for a single runner, we represent each of those sections using a single state of the event model. The transition probability function is based on the distance a runner can travel in a single time step, and as the sections form a sequence, their neighbor-to-neighbor connections.

Figure 5.8b shows the event model for a runner i where the track is in sections s_0 – s_7 . In this example, when the runner is in state s_j then, at the next time step, based on his speed, he could be in any of states s_j , s_{j+1} , s_{j+2} , and s_{j+3} . The event model for the joint behavior of John and James is formed from the product of their individual event models. Each state of this event model represents a tuple of sections of the field in which John and James could be. Events p_{12} and p_{21} both happen with probability 0.5 at each state representing a situation where both John and James are in one section of the track. The current state of the event model is observable by the robot: perhaps, at specific locations along the course, there are stationary cameras that tell the robot the sections the runners currently occupy. The robot does not, however, know the sections which John and James will be in in the next time step because it does not know how their speed will change in the future. Thus, to find an optimal policy for capturing events, we need to solve the RTM/FOM problem.

The desired story is specified by the DFA in Figure 5.8c. To solve this problem,

we form the Goal MDP and we can use the (classical) value iteration to compute an optimal policy for the MDP. However, because the given DFA is loop-omitted acyclic, a better option would be to use the topological value iteration algorithm, introduced in Section 5.4, to compute an optimal policy for the MDP. Closer observation suggests that we can even use the algorithm introduced in Section 5.4 for loop-omitted DAG MDPs. This is because not only the DFA is loop-omitted acyclic, but the graph underlying the event model also does not have any cycles other than self-loops; these facts together make the Goal MDP a loop-omitted DAG MDP. We designate the algorithm that we introduced for solving loop-omitted MDPs, *single value iteration*, owing to the fact that only one iteration for each state is required to solve the Bellman equation for this kind of MDP, and the optimal action for each state is computed by solving several single variable equations.

In this experiment, we compare the running times of classic value iteration, topological value iteration, and the single value iteration algorithms to compute an optimal policy for the MDP. To do so, we consider 10 racetracks, varying the number of sections from 30 to 120. For each of these 10 problem sizes $\{30, 40, \dots, 120\}$, we construct a Goal MDP from the DFA and the event model, and then compute an optimal policy for that MDP using each of the three algorithms. Figures 5.8d and 5.8e give a sense of the sizes of those MDPs; the former shows for each of those MDPs, the number of states of the MDP, while the latter shows the number of edges of the graph underlying the MDP.

For each of those 10 problem sizes, we conducted 10 trials for each of the three algorithms. Figure 5.8f shows, for each of those 10 problem sizes, the average computation time of each of the algorithms. The computation time for the topological value iteration in this diagram includes the computation for forming the SCCs of the graph and the time needed to find a topological ordering. Figure 5.8g shows how much time these steps contribute to the overall computation time of the topological

value iteration algorithm. Likewise, Figure 5.8h shows the contributions for the two phases that make up the single value iteration, namely, finding a topological ordering of the MDP states, and solving the single value equations to find optimal actions for states.

In this experiment, we observe that the topological value iteration outperforms the classical value iteration and the single value iteration outperforms both of them. For the last MDP, which has approximately 57,000 states and approximately 7,000,000 nonzero entries within its transition function, it took 174 seconds on average for the classical value iteration to compute an optimal policy. In contrast, the single value iteration on average took less than 3 seconds to compute an optimal policy. This experiment justifies the use of the algorithms we introduced in Section 5.4 for special inputs of our problem.

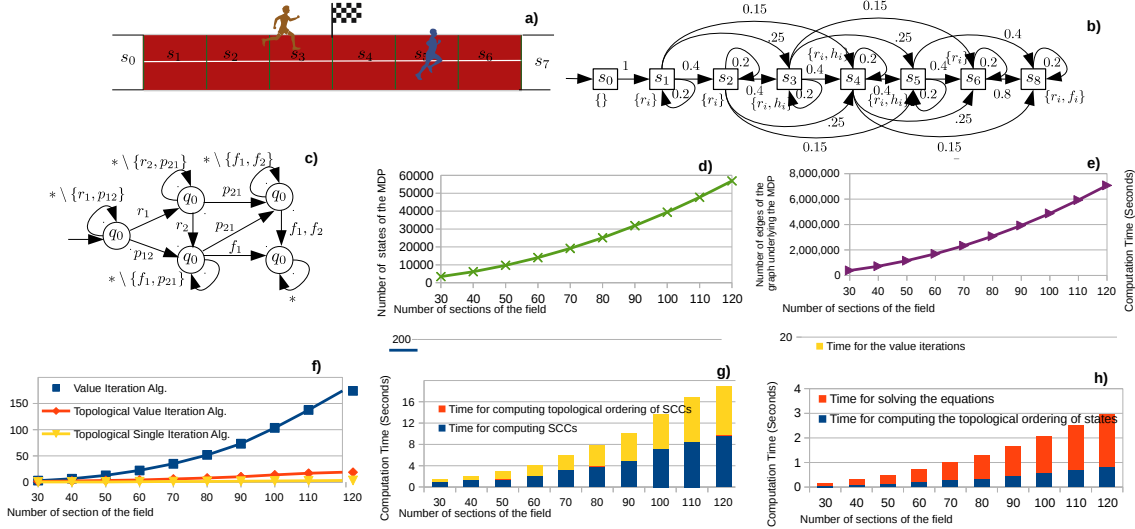


Figure 5.8: **a)** An example of a race source, which we have divided into 8 sections s_0 through s_7 form an event model for the race. **b)** The event model for a single runner $i \in \{1, 2\}$, including events r_i , runner i is running; h_i , runner i is crossing the race flag at the middle of the race field; f_i , runner i is crossing the finish line. The event model for the two runners John and James is made by the product of the event models for each of them. Two more events, p_{12} and p_{21} , are introduced to the joint event model. Event p_{12} denotes that John is overtaking James, while p_{21} means that James is overtaking John. Each of these two events happens with probability 0.5 at each state that represents the runners are in the same section. **c)** The DFA specifying the set of all desirable videos for the race example. **d)** A diagram showing the number of states of the MDP created by the product of the event model and the DFA in this race example. **e)** A diagram showing the number of edges of the graph underlying the MDP. Parts **d** and **e** together show how the size of the MDP increases as the size of the problem increases. They show as the problem's size doubles, the MDP's size approximately quadruplicates. **f)** A diagram showing the computation time for classical value iteration, which we use in our general algorithm, the computation time of the topological value iteration algorithm, which use to solve our problem where the DFA is loop-omitted acyclic, and the computation time of the topological value iteration, which we use to solve our problem when Goal MDP is a loop-omitted DAG. For this experiment, the topological single iteration was on average 6.5 faster than the topological value iteration and also the latter on average was 6 times faster than the general, classical value iteration algorithm. **g)** A diagram showing, for topological single value iteration, the breakdown of its computation time into the three phases of the algorithm: decomposing the MDP into its SCCs, finding a topological ordering of the SCCs, and performing the value iteration. **h)** A diagram showing, for the topological single value iteration algorithm, how much each of the two phases of the algorithm, namely, finding a topological ordering of the states of the MDP, and computing an optimal action for each state via solving the single variable equations, contribute to the computation time of the algorithm. The results for each of the section sizes 30-120 in the diagrams in parts **(f)**, **(g)**, and **(h)** are the average computation times across 10 trials.

CHAPTER 6

OPTIMAL LTL PLANNING WITH LDL_f SOFT CONSTRAINTS

In this chapter, we consider linear temporal logic planning with LDL_f soft specifications. The material in this chapter is based on our work Rahmani and O’Kane [107], which appeared in IROS 2019.

The organization of this chapter is as follows. In Section 6.1, we present preliminary definitions, in Section 6.2, we present our problem statement, in Section 6.3, we present our algorithm, and in Section 6.4, we present our case studies.

6.1 PRELIMINARY DEFINITIONS: WORDS, TRANSITION SYSTEMS, LTL, BÜCHI AUTOMATA, AND LDL_f

In this section, we review the definitions of several formal tools that we utilize to define and solve the preference-guided temporal planning problem. We present these standard definitions here to help ensure that this chapter remains self-contained; readers already familiar with these tools may wish to skip ahead to Section 6.2.

Words, finite and infinite

The set of all finite words over an alphabet Σ is denoted Σ^* ; the set of all infinite words on the same alphabet is denoted Σ^ω . For any integer $j \geq 0$ and infinite word $w = a_0a_1a_2 \cdots \in \Sigma^\omega$, we use $w[..j]$ and $w[j..]$ to denote respectively the prefix $a_0a_1 \dots a_j$

and the postfix $a_j a_{j+1} \dots$ of w . We write $w[i..j]$ and $w[i]$ to denote $a_i a_{i+1} \dots a_j$ and a_i , respectively. For finite words, these same four notations apply. The infinite repetition of a finite word $r \in \Sigma^*$, which is an infinite word in Σ^ω , is denoted r^ω . A *lasso* is formed when such an infinite repetition is concatenated to a finite word, that is, a lasso is an infinite word of the form $r_1(r_2)^\omega$, in which $r_1 \in \Sigma^*$ and $r_2 \in \Sigma^+$.

Transition systems

Next we consider a structure to model the environment.

Definition 23. [transition system] A *transition system* is a tuple $\mathcal{T} = (S, R, s_0, AP, L)$, in which S is a finite set of states; $R \subseteq S \times S$ is a transition relation; $s_0 \in S$ is the initial state; AP is a set of atomic propositions; and $L : S \rightarrow 2^{AP}$ is a function associating atomic propositions to each state.

A transition system may perhaps be most readily understood as a directed graph; see Figure 1.3. An *infinite path* on the transition system is a sequence of states $s_0 s_1 s_2 \dots \in S^\omega$, starting from the initial state s_0 , and for which $(s_i, s_{i+1}) \in R$ for all $i \geq 0$. Since we are interested in infinite paths, we assume that the transition system does not have any *blocking states*, that is, states without any outgoing edges.

To each state s , the labeling function L assigns a (possibly empty) set of atomic propositions $L(s)$, describing the properties of interest that hold at that state. For any infinite path $\pi = s_0 s_1 \dots \in S^\omega$, we define its *trace* as $\text{trace}(\pi) = L(s_0)L(s_1)\dots \in (2^{AP})^\omega$. The trace of a finite path is defined similarly.

Linear temporal logic

The overall mission for our robot is expressed as a formula expressed in LTL over the atomic propositions in the transition system.

Definition 24. [LTL syntax] Given a set of atomic propositions AP , an LTL formula φ over AP is a word generated by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \diamond\varphi \mid \square\varphi \mid \varphi \mathcal{U}\varphi,$$

in which $p \in AP$; \neg (negation), \wedge (conjunction), and \vee (disjunction) are Boolean operators; \bigcirc (next), \diamond (eventually), \square (always), and \mathcal{U} (until) are temporal operators.

Such a formula is interpreted as follows.

Definition 25. [LTL semantics] Let $\sigma = A_0A_1A_2\cdots \in (2^{AP})^\omega$ be a trace, and φ be an LTL formula. We say that σ *satisfies* φ , denoted $\sigma \models \varphi$, if the tuple (σ, φ) is related by the satisfaction relation \models , defined recursively as follows:

- $\sigma \models p$ iff $p \in A_0$;
- $\sigma \models \neg\varphi$ iff $\sigma \not\models \varphi$;
- $\sigma \models \varphi_1 \wedge \varphi_2$ iff $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$;
- $\sigma \models \varphi_1 \vee \varphi_2$ iff $\sigma \models \varphi_1$ or $\sigma \models \varphi_2$;
- $\sigma \models \bigcirc\varphi$ iff $\sigma[1..] \models \varphi$;
- $\sigma \models \diamond\varphi$ iff $\exists k \geq 0, \sigma[k..] \models \varphi$;
- $\sigma \models \square\varphi$ iff $\forall k \geq 0, \sigma[k..] \models \varphi$;
- $\sigma \models \varphi_1 \mathcal{U} \varphi_2$ iff $\exists j \geq 0, \sigma[j..] \models \varphi_2$ and $\forall 0 \leq i < j, \sigma[i..] \models \varphi_1$;

As an example, recall the mission of the robot in Figure 1.3, $\varphi = \square\diamond k \wedge \square\diamond h$, which describes trajectories that always eventually visit the kitchen (that is, visit the kitchen infinitely often), and also always eventually visit the bathroom. The formula $\varphi_2 = \diamond(k \wedge \diamond(c \wedge \diamond\square h))$ specifies all trajectories in which the robot visits the kitchen, the closet, and the bathroom, in that order at least once, and then ultimately stays

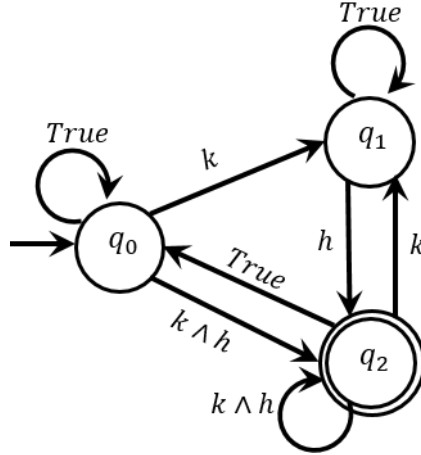


Figure 6.1: A Büchi automaton A_φ for LTL formula $\varphi = \Box\Diamond k \wedge \Box\Diamond h$. For the transition system in Figure 1.3, both the formula and the corresponding automaton shown here specify trajectories that infinitely often visit both the kitchen and the bathroom.

in the bathroom forever. The LTL formula $\varphi_3 = \varphi_2 \wedge \Box(\neg k \vee \bigcirc\Box\neg k)$ imposes the restriction of “not allowing the kitchen to be visited more than once” onto φ_2 .

Note that one could define \Diamond (‘eventually’) as $\Diamond\varphi := \top \mathcal{U}\varphi$, and \Box (‘globally’) as $\Box\varphi := \neg\Diamond\neg\varphi$, are also used.

Büchi automata

Though our approach accepts the mission specification as an LTL formula, the operation of our algorithm constructs and utilizes a different representation of the mission, called a Büchi automaton, defined as follows.

Definition 26. [Büchi automaton] A *Büchi automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, in which Q is a finite set of states; Σ is an alphabet; $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation; $q_0 \in Q$ is the initial state; and $F \subseteq Q$ is a set of accepting states.

Büchi automata may readily be visualized as directed graphs. See Figure 6.1. Note the structural similarities to nondeterministic finite automata.

An *infinite run* r over a Büchi automaton is an infinite sequence of states $r = q_0q_1q_2 \cdots \in Q^\omega$ starting at the initial state q_0 , such that for all $i \geq 0$, there exists an $a \in \Sigma$ such that $(q_i, a, q_{i+1}) \in \delta$. A sequence $r = q_0q_1q_2 \cdots \in Q^\omega$ is a run for a word $w = a_0a_1 \cdots \in \Sigma^\omega$ if $(q_i, a_i, q_{i+1}) \in \delta$ for each $i \geq 0$. We use $\text{inf}(r)$ to denote the set of states that appear infinitely many times in an infinite run $r = q_0q_1q_2 \dots$, that is, $\text{inf}(r) = \{q \in Q \mid \forall i \geq 0, \exists j \geq i, q_j = q\}$. Run r is *accepting* if $\text{inf}(r) \cap F \neq \emptyset$. The *language* of a Büchi automaton \mathcal{A} , denoted $\mathcal{L}_\omega(\mathcal{A})$, is defined as follows:

$$\mathcal{L}_\omega(\mathcal{A}) = \{w \in \Sigma^\omega \mid \text{there exists an accepting run } r \text{ for } w\}.$$

Büchi automata are of interest to us because they can encode our robot's LTL mission in a form more amenable to algorithmic analysis. Specifically, for any LTL formula φ over a set of atomic propositions AP , there exists a Büchi automaton \mathcal{A}_φ with alphabet $\Sigma = 2^{AP}$ such that $\mathcal{L}_\omega(\mathcal{A}_\varphi) = \{\sigma \in (2^{AP})^\omega \mid \sigma \models \varphi\}$, that is, automaton \mathcal{A}_φ accepts exactly all traces satisfying φ . Algorithms to perform this kind of construction of a Büchi automaton from an LTL formula are well-known [3, 46, 137, 160].

Linear dynamic logic over finite traces

Finally, we review the formal language, namely Linear Dynamic Logic Over Finite Traces (LDL_f), used to specify preferences as part of the problem input. The syntax of LDL_f is defined below.

Definition 27. [LDL_f syntax] Given a set of atomic propositions AP , an *LDL_f formula* ψ over AP is a word generated by the following grammar, with start symbol ψ :

$$\begin{aligned} \psi &::= p \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \langle \rho \rangle \psi \mid [\rho] \psi \\ \rho &::= \phi \mid \psi? \mid \rho; \rho \mid \rho + \rho \mid \rho^* \end{aligned}$$

in which p is an atomic proposition in AP and ϕ is a propositional formula over the atomic propositions in AP .

Notice that LDL_f formulas can combine two types of constructions, realized in the symbols ψ and ρ . The first type, denoted ψ , is similar to an LTL formula except that it has modal operators ($\langle \rangle$ and $[\]$) instead of temporal operators. The second type, denoted ρ , is called a *path expression*, in which operators ‘;’ (concatenation), ‘+’ (union), and ‘*’ (Kleene star) are exactly the operators used within regular expressions. To make a path expression, we can use test constructs in the form $\psi?$ too, which allow for checking of LDL_f formulas’ satisfaction.

As an example path expression, consider $\rho = k; (true)^*; h$ over $AP = \{l, k, d, h, c\}$, the set of atomic propositions used for the example in Figure 1.3. This formula specifies all finite traces that start from the kitchen and end in the bathroom.

LDL_f uses the operators $\langle \rangle$ and $[\]$ to encapsulate an RE_f expression ρ to create formulas of the form $\langle \rho \rangle \psi$ and $[\rho] \psi$. Informally, the former means that from the current time instance (the current position), with a trace satisfying ρ we can reach a time satisfying formula ψ ; while the latter means that from the current time instance, *all* traces satisfying ρ end in a time instance satisfying ψ .

Each LDL_f formula ψ over AP specifies a set of finite traces $\text{traces}(\psi) \subseteq (2^{AP})^*$ such that each finite trace $\hat{\sigma} \in \text{traces}(\psi)$ satisfies ψ , which is denoted $\hat{\sigma}, 0 \models \psi$. The semantics of LDL_f is defined as follows.

Definition 28. Given a set of atomic propositions AP and a finite trace $\hat{\sigma} \in (2^{AP})^*$, it is inductively defined whether formula ψ is true at an instant $0 \leq i \leq |\hat{\sigma}| - 1$ of $\hat{\sigma}$ —denoted $\hat{\sigma}, i \models \psi$ —as follows:

- $\hat{\sigma}, i \models p$ iff $p \in \hat{\sigma}[i]$
- $\hat{\sigma}, i \models \neg \psi$ iff $\hat{\sigma}, i \not\models \psi$

- $\hat{\sigma}, i \models \psi_1 \wedge \psi_2$ iff $\hat{\sigma}, i \models \psi_1$ and $\hat{\sigma}, i \models \psi_2$
- $\hat{\sigma}, i \models \psi_1 \vee \psi_2$ iff $\hat{\sigma}, i \models \psi_1$ or $\hat{\sigma}, i \models \psi_2$
- $\hat{\sigma}, i \models \langle \rho \rangle \psi$ iff for some $i \leq j \leq |\hat{\sigma}| - 1$, it holds that $(i, j) \in \mathcal{R}(\rho, \hat{\sigma})$ and $\hat{\sigma}, j \models \psi$
- $\hat{\sigma}, i \models [\rho] \psi$ iff for all $i \leq j \leq |\hat{\sigma}| - 1$ such that $(i, j) \in \mathcal{R}(\rho, \hat{\sigma})$, it holds that $\hat{\sigma}, j \models \psi$

where the relation $\mathcal{R}(\rho, s)$ is defined recursively as follows:

- $\mathcal{R}(\phi, s) = \{(i, i + 1) \mid s, i \models \phi\}$
- $\mathcal{R}(\psi?, s) = \{(i, i) \mid s, i \models \psi\}$
- $\mathcal{R}(\rho_1; \rho_2, s) = \{(i, j) \mid \text{exists } k \text{ such that } (i, k) \in \mathcal{R}(\rho_1, s) \text{ and } (k, j) \in \mathcal{R}(\rho_2, s)\}$
- $\mathcal{R}(\rho_1 + \rho_2, s) = \mathcal{R}(\rho_1, s) \cup \mathcal{R}(\rho_2, s)$
- $\mathcal{R}(\rho^*, s) = \{(i, i)\} \cup \{(i, j) \mid \text{exists } k \text{ such that } (i, k) \in \mathcal{R}(\rho, s) \text{ and } (k, j) \in \mathcal{R}(\rho^*, s)\}$.

To illustrate the idea of LDL_f , recall the example preferences from Section 1.4. Preference (a), which states that the robot should ideally not go into d_1 , can be expressed by $\psi_1 = [\text{true*}] \neg d_1$, or by the formula $\psi'_1 = \neg \langle \text{true*} \rangle d_1$, which is obtained via the duality rule between the operators $\langle \rangle$ and $[\]$, according to which $[\rho] \neg \psi \Leftrightarrow \neg \langle \rho \rangle \psi$ for any ψ and ρ . Similarly, preference (b) can be expressed in LDL_f as $\psi_2 = [\text{true*}] \neg d_2$ and preference (c) as $\psi_3 = ([\text{true*}] \neg (d_1 \vee d_2)) \vee (\langle (\neg (d_1 \vee d_2))^* \rangle c)$. The first part of ψ_3 says that the robot does not go into any of d_1 and d_2 , and the second part says that the robot should go to c before going to d_1 or d_2 .

6.2 OPTIMAL PREFERENCE PLANNING

This section presents the definition of our optimal preference planning problem. An instance of this problem is defined by three elements: (1) a transition system \mathcal{T} , describing the connectivity of the environment in which the robot moves; (2) a goal specification, an LTL formula φ ; and (3) an ordered list of n preferences, expressed as LDL_f formulas $\psi_1, \psi_2, \dots, \psi_n$ that the user would prefer to be satisfied.

We focus on preferences specified in LDL_f , rather than some other specification language, because of its balance of human legibility, expressivity, and efficiency of computation. One alternative would be Linear Temporal Logic for Finite Traces (LTL_f). However, De Giacomo and Vardi [31] showed that this language is not as powerful as it was traditionally assumed, and hence, they proposed LDL_f as a strictly more expressive language with similar computational complexity to LTL_f .

Based on these preferences, we define for the traces of finite trajectories, a cost function $f : (2^{AP})^* \rightarrow \{0, 1, \dots, n^n\}$ as follows:

$$f(\hat{\sigma}) = \sum_{\hat{\sigma} \notin \text{traces}(\psi_i)} n^{n-i}. \quad (6.1)$$

Notice that a trajectory that violates higher ranked preferences receives a higher cost compared to one who violates lower ranked preferences. Accordingly, among two finite trajectories, the one with lower cost is preferred.

Our plans, however, will be infinite trajectories, for the traces of which we define a new cost function $f_\omega : (2^{AP})^\omega \rightarrow \{0, 1, \dots, n^n\}$, such that for any $\sigma \in (2^{AP})^\omega$,

$$f_\omega(\sigma) = \max_{i \geq 0} f(\sigma[.i]). \quad (6.2)$$

We combine these elements to form the OPP problem.

Algorithm 3: OPTIMALPREFERENCEPLANNING

1 **Input:** A transition system \mathcal{T} , an LTL formula φ , and a sequence of n LDL_f formulas $\psi_1, \psi_2, \dots, \psi_n$ **Output:** An infinite path $\pi = s_0s_1s_2\cdots$ on \mathcal{T} such that $\text{trace}(\pi) \models \varphi$ and $f_\omega(\text{trace}(\pi))$ is minimized.

2 **for** $i = 1$ **to** n **do**

3 | $D_i \leftarrow \text{LDLF2DFA}(\psi_i)$

4 $\mathcal{F} \leftarrow \text{INTEGRATEDFAS2FILTER}(D_1, D_2, \dots, D_n)$

5 $\mathcal{A}_\varphi \leftarrow \text{LTL2BÜCHIAUTOMATON}(\varphi)$

6 $\mathcal{P} \leftarrow \mathcal{A}_\varphi \times \mathcal{T} \times \mathcal{F}$

7 **if** *not* $\text{HASACCEPTINGRUN}(\mathcal{P})$ **then**

8 | **return** *nil*

9 $(r_1, q_f, r_2) \leftarrow \text{MINIMAXACCEPTINGRUN}(\mathcal{P})$

10 $\pi = \text{CONVERT2PATHONTS}(r_1, q_f, r_2)$

11 **return** π

Problem: Optimal Preference Planning (OPP)

Input: A transition system \mathcal{T} , an LTL formula φ , and n LDL_f formulas $\psi_1, \psi_2, \dots, \psi_n$.

Output: An infinite path π over \mathcal{T} such that $\text{trace}(\pi) \models \varphi$ and $f_\omega(\text{trace}(\pi))$ is minimized.

6.3 ALGORITHM DESCRIPTION

This section describes an algorithm for the OPP problem. The algorithm operates in three phases. First, we construct a representation of the given preferences in the form of a combinatorial filter whose outputs are the costs as determined by Equations 6.1 and 6.2. Then, the algorithm forms a product graph of this filter with a Büchi automaton for the LTL goal specification, forming a certain type of state-weighted Büchi automaton the accounts for both the goal and the costs arising from violated preferences. Finally, we can generate the optimal solution trajectory on this automaton. Algorithm 3 summarizes the process.

Cost filters

The first step in Algorithm 3 is to construct a representation that integrates the effects of all of the LDL_f formulas. This representation is a filter $\mathcal{F} = (V, Y, C, \tau, v_0, c)$ in which $C = \mathbb{Z}_{\geq 0}$ and τ is a complete transition function. Note that this filter can be viewed as a slight generalization of the standard deterministic finite automaton, in which, rather than a single distinction between accepting or non-accepting states, the filter can produce many different output values. Also, for any word $w \in Y^*$, $\tau^*(v_0, w)$ is the state to which the automaton reaches by reading word w . The filter partitions the set of all finite words in Y^* such that all words w for whom $c(\tau^*(v_0, w))$ are equal fall into a single equivalence class. We leverage the filter to represent the cost function f (Equation 6.1), as implied by the following result.

Lemma 11. Let $\psi_1, \psi_2, \dots, \psi_n$ be n LDL_f formulas over a set of automatic proposition AP , and let f be the function in Equation 6.1 defined for these formulas. Then there exists a filter $\mathcal{F} = (V, 2^{AP}, \tau, v_0, c)$ such that, for each $\hat{\sigma} \in (2^{AP})^*$, $c(\tau^*(v_0, \hat{\sigma})) = f(\hat{\sigma})$.

Proof. The algorithm of De Giacomo and Vardi [31] ensures that, for any LDL_f formula ψ_i , we can construct a DFA $\mathcal{D}_i = (V_i, 2^{AP}, \tau_i, v_{0,i}, F_i)$, for which $\mathcal{L}(\mathcal{D}_i) = \text{traces}(\psi_i)$. From these DFAs, construct $\mathcal{F} = (V, 2^{AP}, \tau, v_0, c)$ such that

- $V = V_1 \times V_2 \times \dots \times V_n$,
- $v_0 = (v_{0,1}, v_{0,2}, \dots, v_{0,n})$,
- for any state $(v_1, v_2, \dots, v_n) \in V$ and any $A \in 2^{AP}$,

$$\tau((v_1, v_2, \dots, v_n), A) = (\tau_1(v_1, A), \tau_2(v_2, A), \dots, \tau_n(v_n, A)),$$

and

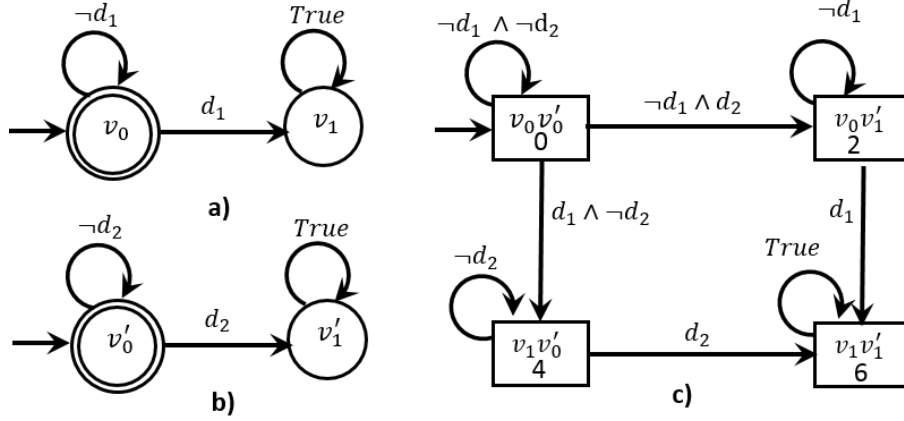


Figure 6.2: **(a)** A DFA \mathcal{D}_1 accepting traces satisfying LDL_f formula $\psi_1 = [true^*]\neg d_1$ **(b)** A DFA \mathcal{D}_2 accepting all the traces satisfying LDL_f formula $\psi_2 = [true^*]\neg d_2$ **(c)** Filter \mathcal{F} constructed by Lemma 11 for \mathcal{D}_1 and \mathcal{D}_2 .

- for each state $(v_1, v_2, \dots, v_n) \in V$,

$$c((v_1, v_2, \dots, v_n)) = \sum_{v_i \notin F_i} n^{n-i}.$$

From the construction, it is easy to observe that $f(\hat{\sigma}) = c(\tau^*(v_0, \hat{\sigma}))$ for any $\hat{\sigma} \in (2^{AP})^*$. \square

In Algorithm 3, lines 2–4 utilize Lemma 11 to construct a cost filter \mathcal{F} . Figure 6.2 shows an example. The interpretation of this \mathcal{F} is that the value assigned in \mathcal{F} to the state reached by some finite trajectory in \mathcal{T} is equal to the cost determined by the given preferences for that finite trajectory.

The product automaton

The next portion of Algorithm 3 (lines 5 and 6) forms data structure that integrates both the preferences and the goal specification. To do so, a specific product of automata is constructed according to the following definition.

Definition 29. Given a transition system $\mathcal{T} = (S, R, s_0, AP, L)$, a Büchi automaton $\mathcal{A} = (Q, 2^{AP}, \delta, q_0, F)$, and a filter $\mathcal{F} = (V, 2^{AP}, \tau, v_0, c)$, the *product automaton* $\mathcal{P} = \mathcal{A} \times \mathcal{T} \times \mathcal{F}$ is a tuple $\mathcal{P} = (Q_{\mathcal{P}}, \delta_{\mathcal{P}}, q_{0,\mathcal{P}}, F_{\mathcal{P}}, w_{\mathcal{P}})$ in which

1. $Q_{\mathcal{P}} = Q \times S \times V$ is a finite set of states;
2. $q_{0,\mathcal{P}} = (q_0, s_0, v_0)$ is the initial state;
3. $\delta_{\mathcal{P}} \subseteq Q_{\mathcal{P}} \times Q_{\mathcal{P}}$ is a transition relation, such that $((q, s, v), (q', s', v')) \in \delta_{\mathcal{P}}$ if and only if $(s, s') \in R$, $(q, L(s), q') \in \delta$, and $\tau(v, L(s)) = v'$;
4. $F_{\mathcal{P}} = F \times S \times V$ is a set of accepting states;
5. $w_{\mathcal{P}} : Q_{\mathcal{P}} \rightarrow \{0\} \cup \mathbb{Z}^+$ is a state-weighting function, such that $w_{\mathcal{P}}((q, s, v)) = c(v)$ for each $(q, s, v) \in Q_{\mathcal{P}}$.

Notice that this product automaton \mathcal{P} is itself a Büchi automaton with the trivial alphabet, but to each state of it a weight has been assigned.

The purpose of the product automaton is to encapsulate the effects of both the goal mission and the preferences. To demonstrate how, the following lemma establishes the relationship between \mathcal{P} and the Büchi automaton \mathcal{A} derived from the goal φ .

Lemma 12. Let \mathcal{T} , \mathcal{A} , \mathcal{F} , and \mathcal{P} be the structures in Definition 29. It holds that for any accepting run $r_{\mathcal{P}} = (q_0, s_0, v_0)(q_1, s_1, v_1)(q_2, s_2, v_2) \cdots$ over \mathcal{P} , the sequence $\pi = s_0 s_1 s_2 \cdots$ is a path for \mathcal{T} such that $\text{trace}(\pi) \in \mathcal{L}_{\omega}(\mathcal{A})$. Moreover, for any path $\pi = s_0 s_1 s_2 \cdots$ in \mathcal{T} such that $\text{trace}(\pi) \in \mathcal{L}_{\omega}(\mathcal{A})$, there exists an accepting run $r_{\mathcal{P}} = (q_0, s_0, v_0)(q_1, s_1, v_1)(q_2, s_2, v_2) \cdots$ over \mathcal{P} .

Proof. For the first claim, given the definitions of $Q_{\mathcal{P}}$, $Q_{0,\mathcal{P}}$, and $\delta_{\mathcal{P}}$, the sequence $\pi = s_0 s_1 s_2 \cdots$ is a path over \mathcal{T} , and that the sequence $r = q_0 q_1 q_2 \cdots$ is a run for $\text{trace}(\pi) = L(s_0)L(s_1)L(s_2) \cdots$ over \mathcal{A} . Given the definition of $F_{\mathcal{P}}$ and that $r_{\mathcal{P}}$ is an

accepting run for \mathcal{P} , sequence r , which is a run for π , is an accepting run over \mathcal{A} . Thus, $\text{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$.

For the second claim, given that $\pi \in \mathcal{L}_\omega(\mathcal{A})$, there exists an accepting run $r = q_0q_1q_2 \cdots \in Q^\omega$ for $\text{trace}(\pi)$ over \mathcal{A} . Also due to that τ is a complete function, there exists a unique run $v_0v_1v_2 \cdots$ for $\text{trace}(\pi)$ on \mathcal{F} . Considering the way the components of \mathcal{P} are constructed, it is easy to observe that $r_{\mathcal{P}} = (q_0, s_0, v_0)(q_1, s_1, v_1)(q_2, s_2, v_2) \cdots$ is a run over \mathcal{P} . Having the definition of $F_{\mathcal{P}}$ and that the sequence $r = q_0q_1q_2 \cdots$ is an accepting run over \mathcal{A} , it holds that $\text{inf}(r) \cap F \neq \emptyset$, which implies that for infinitely many i 's, $(q_i, s_i, v_i) \in F_{\mathcal{P}}$. Thus, run $r_{\mathcal{P}}$ is accepting over \mathcal{P} . \square

Lemma 12 establishes that, for any accepting run over the product automaton, there is a feasible solution for the OPP problem, and likewise for any feasible solution of OPP, there is an accepting run over the product automaton.

The key remaining question is when that solution is optimal, in the sense of minimizing the cost of preference violations. To answer that question, we first define a function $f_{\mathcal{P}}$ over the infinite runs of \mathcal{P} . Specifically, for any infinite run $r_{\mathcal{P}} = p_0p_1p_2 \cdots \in Q_{\mathcal{P}}^\omega$, we let

$$f_{\mathcal{P}}(r_{\mathcal{P}}) = \max_{i \geq 0} w_{\mathcal{P}}(p_i). \quad (6.3)$$

Between function $f_{\mathcal{P}}$ and f_ω there is a special connection, revealed by the following lemma.

Lemma 13. Given the structures \mathcal{T} , \mathcal{A} , \mathcal{F} , and \mathcal{P} from Definition 29, let $\pi = s_0s_1s_2 \cdots$ be an infinite path over \mathcal{T} such that $\text{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$ and $r_{\mathcal{P}} = (q_0, s_0, v_0)(q_1, s_1, v_1)(q_2, s_2, v_2) \cdots$ be any run over \mathcal{P} . Then $f_{\mathcal{P}}(r_{\mathcal{P}}) = f_\omega(\text{trace}(\pi))$.

Proof. Due to the construction of \mathcal{P} , the sequence $r = q_0q_1q_2 \cdots$ is a run for $\text{trace}(\pi)$ over \mathcal{A} , and $t = v_0v_1v_2 \cdots$ is a run for $\text{trace}(\pi)$ over \mathcal{F} . For any $i \geq 0$, $w_{\mathcal{P}}((q_i, s_i, v_i)) =$

$c(v_i)$. Therefore, $\max_{i \geq 0} w_{\mathcal{P}}((q_i, s_i, v_i)) = \max_{i \geq 0} c(v_i)$, which implies that $f_{\mathcal{P}}(r_{\mathcal{P}}) = f_{\omega}(\text{trace}(\pi))$. \square

An impact of Lemma 12 combined with Lemma 13 is that to solve OPP, we can compute the state-weighted Büchi automaton \mathcal{P} , and then find on \mathcal{P} an accepting run $r_{\mathcal{P}}$ that minimizes $f_{\mathcal{P}}(r_{\mathcal{P}})$. Subsequently, the projection of $r_{\mathcal{P}}$ on \mathcal{T} will be a solution to OPP.

Trajectory generation

The final phase of Algorithm 3, shown in lines 7–10, is to find an optimal accepting run over \mathcal{P} . As a first step, using standard algorithms for checking emptiness of the language of a Büchi automaton, one can decide in $\mathcal{O}(|Q_{\mathcal{P}}|)$ time whether the input \mathcal{P} has a feasible solution or not. If not, Algorithm 3 reports an error and terminates.

However, if there is a feasible solution, then there may be many—even infinitely many—optimal solutions. Fortunately, we need to find only a single solution. The following result establishes that we need only search for solutions with a specific structure.

Lemma 14. If the OPP instance with product automaton \mathcal{P} has a feasible solution, then it has an optimal solution of the form $r_{\mathcal{P}} = r_1(q_f r_2)^{\omega}$ where $r_1, r_2 \in Q_{\mathcal{P}}^*$, $q_f \in F_{\mathcal{P}}$, $r_1[i] \neq q_f$ for any $0 \leq i < |r_1|$, and $r_2[j] \neq q_f$ for any $0 \leq j < |r_2|$.

Proof. The idea is that from any optimal solution $r'_{\mathcal{P}}$, we can construct an optimal solution of the form $r_{\mathcal{P}} = r_1(q_f r_2)^{\omega}$ such that $f_{\mathcal{P}}(r'_{\mathcal{P}}) = f_{\mathcal{P}}(r_{\mathcal{P}})$. Given that $r'_{\mathcal{P}}$ is accepting—going through an accepting state infinitely many times—we have that $r'_{\mathcal{P}} = r_1 q_f r_2 q_f r_3 \cdots$ for a $q_f \in F_{\mathcal{P}}$ such that $r_i \in Q_{\mathcal{P}}^*$ for any $i \geq 1$, and q_f does not appear in any of the r_i 's. Now, from $r'_{\mathcal{P}}$, we construct $r_{\mathcal{P}} = r_1 q_f r_2 q_f r_2 \cdots = r_1 (q_f r_2)^{\omega}$. Clearly, run $r_{\mathcal{P}}$ is defined over the automaton given that $r_1 q_f r_2 q_f$ was a prefix of $r'_{\mathcal{P}}$,

and that $r_{\mathcal{P}}$ is accepting given that $q_f \in \text{inf}(r_{\mathcal{P}})$. To show that $f_{\mathcal{P}}(r'_{\mathcal{P}}) = f_{\mathcal{P}}(r_{\mathcal{P}})$, observe that it holds that $f_{\mathcal{P}}(r'_{\mathcal{P}}) \leq f_{\mathcal{P}}(r_{\mathcal{P}})$ given that $r'_{\mathcal{P}}$ is optimal, and it holds that $f_{\mathcal{P}}(r'_{\mathcal{P}}) \geq f_{\mathcal{P}}(r_{\mathcal{P}})$ given the definition of $f_{\mathcal{P}}$ (Equation 6.3) and that $r'_{\mathcal{P}}$ contains all the states of $r_{\mathcal{P}}$. \square

The upshot of Lemma 14 is that we can restrict our attention to solutions composed of a prefix that reaches some accepting state q_f , followed by repetitions of a cycle including q_f . The following lemma restricts this form of run further.

Lemma 15. There exists a run $r_{\mathcal{P}} = r_1(q_f r_2)^\omega$ for Lemma 14 in which both r_1 and r_2 are *simple*; that is, $r_1[i] \neq r_1[j]$ for any $0 \leq i \neq j < |r_1|$, and $r_2[i] \neq r_2[j]$ for any $0 \leq i \neq j < |r_2|$.

Proof. The idea is to remove duplicate states from r_1 and r_2 until they become simple. Then, we need to show that the new run is mapped by function $f_{\mathcal{P}}$ to the same value to which the original run was mapped. Assume that r_2 is not simple, that is, it passes through a state q at least twice. Let $r_2 = q_0 q_1 \cdots q_i \cdots q_j q_{j+1} \cdots q_{|r_2|-1}$ such that $q_i = q_j = q$ where i and j are respectively the first and the last positions at which q appears. There are two cases: the first one where $j < |r_2| - 1$, and the second one where $j = |r_2| - 1$. In the former case, we replace r_2 by a new sequence $q_0 q_1 \cdots q_i q_{j+1} \cdots q_{|r_2|-1}$, and in the later case, we replace r_2 by a new sequence $q_0 q_1 \cdots q_i$. In both cases, the new sequence, which is clearly a run for the automaton, passes through q exactly once. This process of removing duplicate states is continued until r_2 becomes a simple cycle. The same process applicable on r_1 . Let assume that the new run is $r'_{\mathcal{P}}$. Run $r_{\mathcal{P}}$ was optimal, so it holds that $f_{\mathcal{P}}(r_{\mathcal{P}}) \leq f_{\mathcal{P}}(r'_{\mathcal{P}})$. Given the definition of $f_{\mathcal{P}}$ and that $r_{\mathcal{P}}$ contains all states appeared in $r'_{\mathcal{P}}$, it holds that $f_{\mathcal{P}}(r_{\mathcal{P}}) \geq f_{\mathcal{P}}(r'_{\mathcal{P}})$. Therefore, $f_{\mathcal{P}}(r_{\mathcal{P}}) = f_{\mathcal{P}}(r'_{\mathcal{P}})$. \square

The combined impact of Lemma 14 and Lemma 15 is that we can use some graph algorithms to synthesize an optimal run. Specifically, if we think of \mathcal{P} as a directed graph with $Q_{\mathcal{P}}$ as its vertex set and with $\delta_{\mathcal{P}}$ as its edge set, there is an optimal run $r_{\mathcal{P}} = r_1(q_f r_2)^\omega$ for which r_1 and r_2 are simple paths on this graph and, moreover,

1. path $r_1 q_f$ is a simple path from q_0 to q_f such that the maximum weight of its vertices (states) is minimum among all simple paths starting from q_0 and ending at q_f , and
2. cycle $q_f r_2 q_f$ is a simple cycle starting at q_f such that the maximum weight of its vertices (states) is minimum among all simple cycles starting from q_f .

Accordingly, for each $q_f \in F_{\mathcal{P}}$, we can synthesize a run $r_{\mathcal{P}, q_f} = r_1(q_f r_2)^\omega$ by finding a path $r_1 q_f$ and a cycle $q_f r_2 q_f$ with those properties. Subsequently, we synthesize a solution by choosing an optimal run among those runs synthesized for all vertices (states) $q_f \in F_{\mathcal{P}}$.

With these conditions in mind, the only thing remains is to find, from a given source vertex to a given destination vertex, a path that minimizes the maximum weight of its vertices. Notice that for path $r_1 q_f$, the source vertex is the initial state and the destination vertex is q_f , while for the cycle $q_f r_2 q_f$, both the vertex and destination vertex are q_f . Finding this kind of path is the concern of the problem *minimax path for vertex-weighted graphs*, which can be solved in several different ways.

Path extraction via Dijkstra's algorithm One option is to find the paths by a modified version of one of the well-known algorithms for *the shortest path problem*, including Dijkstra's algorithm. Dijkstra's algorithm.

We run this algorithm for each state $q_f \in F_{\mathcal{P}}$ twice, one to find a minimax path from the initial state to q_f (path $r_1 q_f$), and the other to find minimax path

from q_f to q_f ($q_f r_2 q_f$). By combining these two parts, for each accepting state q_f , we synthesize a run of the form $r_{\mathcal{P}} = r_1(q_f r_2)$ as described in Lemma 14 and Lemma 15. Among the synthesized runs for all accepting states, we choose the one with the minimum weight as the optimal solution. Hence, the total running time is $\mathcal{O}(|F_{\mathcal{P}}|(|Q_{\mathcal{P}}| \log |Q_{\mathcal{P}}| + |\delta_{\mathcal{P}}|))$, as the time complexity of Dijkstra’s algorithm to compute a minimax path is $\mathcal{O}(|Q_{\mathcal{P}}| \log |Q_{\mathcal{P}}| + |\delta_{\mathcal{P}}|)$. For sparse automata, this descends to $\mathcal{O}(|F_{\mathcal{P}}|(|Q_{\mathcal{P}}| \log |Q_{\mathcal{P}}|))$, and even to $\mathcal{O}(|Q_{\mathcal{P}}| \log |Q_{\mathcal{P}}|)$ for sparse automata whose numbers of accepting states are constant. However, this complexity is $\mathcal{O}(|Q_{\mathcal{P}}|^3)$ for dense automata with many accepting states. Therefore, for dense automata we use the second algorithm, described below.

Path extraction via Shapira-Yuster-Zwick Alternatively, one can solve the minimax path problem using any algorithm for the *maximum bottleneck path* problem (also called the *maximum capacity path* or *widest path* problem), which is concerned with finding a path maximizing the minimum weight of vertices in the path. For example, one might use the algorithm of Shapira *et al.* [132]. For this purpose, the weight of each vertex p is replaced by $(\max_{p' \in Q_{\mathcal{P}}} w_{\mathcal{P}}(p')) - w_{\mathcal{P}}(p)$. By doing so, any bottleneck path in the converted graph becomes a minimax path from that source to that destination in the original graph. Finding the all-pairs bottleneck paths using this approach takes $\mathcal{O}(|Q_{\mathcal{P}}|^{2.575})$ time.

Combining these options gives the following analysis of the process of generating the optimal path in \mathcal{P} .

Lemma 16. For any state-weighted Büchi automaton $\mathcal{P} = (Q_{\mathcal{P}}, \delta_{\mathcal{P}}, Q_{0,\mathcal{P}}, F_{\mathcal{P}}, c_{\mathcal{P}})$, an optimal trajectory can be generated in time $\mathcal{O}(\min(|F_{\mathcal{P}}|(|Q_{\mathcal{P}}| \log |Q_{\mathcal{P}}| + |\delta_{\mathcal{P}}|), |Q_{\mathcal{P}}|^{2.575}))$.

Thus, our algorithm for synthesizing an optimal accepting run on \mathcal{P} is based on this lemma. This algorithm decides based on the sizes of $\delta_{\mathcal{P}}$ and $F_{\mathcal{P}}$ —or more

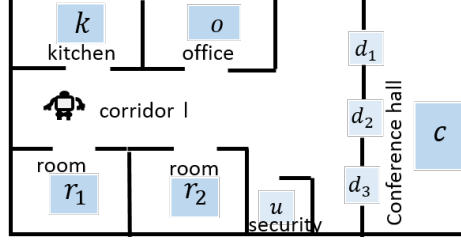


Figure 6.3: An environment consisting of a corridor, an office, a kitchen, two rooms, a conference hall, and a security room.

Scenario	dept	race
Illustration	Figure 6.3	Figure 6.5
Goal	$\varphi = (\neg(d_1 \vee d_2 \vee r_1 \vee r_2 \vee k) \mathcal{U} o)$ $\wedge \diamond(c \wedge \bigcirc \diamond(r_1 \wedge \bigcirc \diamond(r_2 \wedge \bigcirc \diamond \square k)))$	$\varphi = (\square \diamond r_2) \wedge (\square \diamond p)$
Preferences	$\psi_1 = \neg \langle true^* \rangle o$ $\psi_2 = \neg \langle true^* \rangle d_1$ $\psi_3 = \neg \langle true^* \rangle (c \wedge \langle (\neg d_3)^* \rangle d_2)$ $\psi_4 = \neg \langle true^* \rangle d_3$ $\psi_5 = \neg \langle (\neg u)^* \rangle r_1$	$\psi_1 = \neg \langle true^* \rangle (f \wedge \langle true \rangle s) \wedge \neg \langle true^* \rangle (s \wedge \langle true \rangle f)$ $\psi_2 = [true^*](p \rightarrow s)$ $\psi_3 = \neg \langle true^* \rangle (r_1 \wedge f) \wedge \neg \langle true^* \rangle (r_3 \wedge f)$ $\psi_4 = [true^*]((r_2 \vee r_4) \rightarrow f)$
Solution	$s_l s_u s_l s_o s_l s_{d_2} s_c s_{d_3} s_l s_{r_1} s_l s_{r_2}$ $s_l (s_k)^\omega$	$r_1^m r_2^f r_3^m r_4^f r_1^m p^s (r_3^m r_4^f r_1^m r_2^f r_3^m r_4^f r_1^m p^s)^\omega$
Satisfied Pref.	ψ_2, ψ_3, ψ_5	$\psi_1, \psi_2, \psi_3, \psi_4$
Comp. time	18.7s	5.7s

Figure 6.4: Formulation and results for two instances of the OPP problem. precisely, whether $|F_{\mathcal{P}}|(|Q_{\mathcal{P}}| \log |Q_{\mathcal{P}}| + |\delta_{\mathcal{P}}|) < |Q_{\mathcal{P}}|^{2.575}$ —one of the two algorithms mentioned above.

6.4 IMPLEMENTATION AND COMPUTED EXAMPLES

We have implemented Algorithm 3 in Java. In this section, we present example instances, to illustrate its operation in solving OPP problems. The computed results were executed on an Ubuntu 16.04 computer with a 3.6GHz processor.

For the first example, called **dept**, we use the environment in Figure 6.3, in which a mobile robots moves within a university department consisting of a corridor, an

office, a kitchen, two rooms, a security section, and a conference hall. Blue rectangles show regions of interest to the robot, consisting of c in the conference hall, d_1 the first entrance of the conference hall, d_2 the second door of the conference hall, d_3 the third door of the conference hall, r_1 in the first room, r_2 in the second room, u in the security room, k in the kitchen, and o in the office.

Suppose the robot is tasked with this non-trivial goal:

Take the keys from office o but before that do not go to any of d_1 , d_2 , r_1 , r_2 , or k . Additionally, visit c , r_1 , r_2 , and k in that order. After completing these tasks, stay in k .

In completing this goal, the user prefers the robot to maintain several behaviors, modeled as preferences, if possible:

1. Do not go to the office.
2. Do not use door d_1 .
3. Do not exit through door d_2 from the conference hall.
4. Do not use door d_3 .
5. Check the security room before going to room r_1 .

Details of the encoding into an instance of OPP and a solution trajectory, as computed by our implementation, and shown in the left column of Figure 6.4. Notice that due to the goal specification, the first preference cannot be satisfied. Also, since preferences 2, 3, and 4 cannot be satisfied together, the planner satisfies preferences 2 and 3, which have higher priority. In addition, preference 5 is also satisfied.

Figure 6.5 illustrates a second example, called **race**, in which an autonomous race car travels around a cyclic track. Its goal is to circle the track repeatedly, visiting the pit (p) infinitely often to refuel. States in the transition system model both the

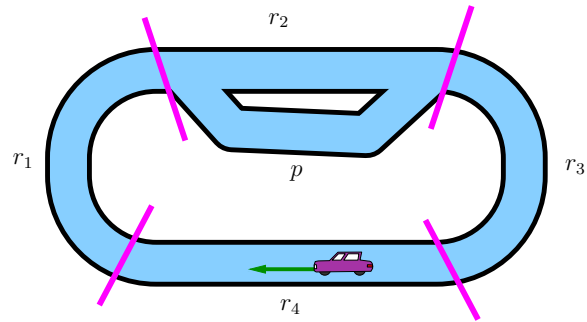


Figure 6.5: A race car traveling around a circular track.

region in which the race car is moving (r_1, r_2, r_3, r_4, p) and its current speed ('fast, 'm'edium, or 's'low). Several preferences are associated with this robot's motion.

1. Do not accelerate or decelerate abruptly. That is, do not change speed from fast directly to slow, nor from slow directly to fast.
2. Drive slowly in the pit.
3. Do not drive fast on the curves.
4. Do drive fast on the straight parts.

An encoding and solution are shown in Figure 6.4.

CHAPTER 7

LTL PLANNING WITH SOFT LTL SPECIFICATIONS

The previous chapter considered a related problem in which the soft constraints were expressed in linear dynamic logic for finite traces (LDL_f) [31]. Such formulas can express constraints only on finite prefixes of the trajectory, rather than on the entire trajectory as a whole. A limitation of that work is that LDL_f soft constraints cannot express soft goals that are satisfied only by infinite (rather than finite) trajectories. As an example, a task that requires a social enrichment robot to infinitely often perform the act of juggling is a simple soft goal that cannot be expressed by LDL_f . The difference in the language used to express the soft constraints not only improves the expressivity of the approach, but it leads to significant (and new, compared to the LDL_f case) algorithmic challenges.

The material of this chapter is based of our work [109], which appeared in IROS 2020.

The organization of this chapter is as follows. We first present our problem statement in Section 7.1. In Section 7.2, we propose our algorithm, and in Section 7.3, we present two case studies.

7.1 DEFINITIONS AND PROBLEM STATEMENT

In this section, we review some preliminary tools and introduce the main problem we address.

Preliminaries

In the algorithm of this chapter, we need to be sure that each Büchi automaton created for a soft constraint is *nonblocking*, that is, for any state q in the automaton and every letter $a \in \Sigma$, there is at least one state q' such that $(q, a, q') \in \delta$. Any Büchi automaton is converted to a nonblocking Büchi automaton by adding a trapping state, to which all missing transitions are added. We also consider a variant of Büchi automaton called *generalized Büchi automaton*, which has the same syntax of Büchi automaton except that it has a set $\mathcal{F} \subseteq 2^Q$ rather than a set $F \subseteq Q$ as its acceptance set. More precisely, the acceptance set of the automaton is a set \mathcal{F} consisting of sets F_1, F_2, \dots, F_k with $F_i \subseteq Q$ for each $i \in \{1, \dots, k\}$. Accordingly, an infinite run r over a generalized Büchi automaton \mathcal{G} is accepting if for each $F \in \mathcal{F}$, it holds that $\text{inf}(r) \cap F \neq \emptyset$. The language of \mathcal{G} , $L_\omega(\mathcal{G})$, is the set of all infinite words for each of which there is an accepting run.

LTL planning with soft constraints

Our goal in this problem is to find, in a transition system modeling the environment, an infinite path whose trace satisfies a goal mission expressed as an LTL formula φ while optimally satisfying a prioritized list of soft constraints $\psi_1, \psi_2, \dots, \psi_n$, where each ψ_i is an LTL formula, given in order of decreasing importance. For this purpose, we define a cost function $f_\omega : (2^{AP})^\omega \rightarrow \mathbb{Z}_{\geq 0}$, such that for any $\sigma \in (2^{AP})^\omega$,

$$f_\omega(\sigma) = \sum_{i: \sigma \notin \text{Words}(\psi_i)} n^{n-i}. \quad (7.1)$$

Note that this cost function guarantees to impose the standard lexicographic ordering between all Boolean vectors, where each vector has an entry for each LTL constraint showing whether that LTL constraint is satisfied or not. As a result, a constraint with a higher priority (smaller number) is never sacrificed to satisfy a constraint with a

lower priority. Accordingly, we want a trajectory whose trace minimizes this function. With this in mind, our problem is defined as:

Problem: Optimal LTL Planning with Soft Constraints (OLPSC)

Input: A transition system \mathcal{T} , an LTL formula φ , and a prioritized list of n LTL formulas $\psi_1, \psi_2, \dots, \psi_n$.

Output: An infinite path π over \mathcal{T} such that $\text{trace}(\pi) \models \varphi$ and $f_\omega(\text{trace}(\pi))$ is minimized.

7.2 ALGORITHM DESCRIPTION

This section presents an algorithm for solving the OLPSC problem. See Algorithm 4. The two main steps of this algorithm are constructing a product automaton (line 5), and computing a lasso with minimum cost on the product automaton (line 6). In the sequel, we explain those steps.

The product automaton

The first step of the algorithm is, following an established pattern in the literature [20, 75, 151, 152], to construct a form of product automaton [159]. To that end, the algorithm first makes the Büchi automata representations of the LTL formulas—an automaton \mathcal{A} for φ , and an automaton \mathcal{B}_i for each ψ_i . It then ensures that \mathcal{B}_i 's are nonblocking and uses all those automata along with the transition system to construct a product automaton based on the following definition.

Definition 30. For a Büchi automaton $\mathcal{A} = (Q, 2^{AP}, \delta, q_0, F)$, a transition system $\mathcal{T} = (S, R, s_0, AP, L)$, and a prioritized list of n nonblocking Büchi automata $\mathcal{B}_i = (Q_i, 2^{AP}, \delta_i, q_{0,i}, F_i)$ for $i \in \{1, \dots, n\}$, the *product automaton* is a tuple $\mathcal{P} = (Q_{\mathcal{P}}, \delta_{\mathcal{P}}, q_{0,\mathcal{P}}, F_{\mathcal{P}}, \mathbf{w})$ in which

1. $Q_{\mathcal{P}} = Q \times S \times Q_1 \times \dots \times Q_n$ is a finite set of states;

2. $q_{0,\mathcal{P}} = (q_0, s_0, q_{0,1}, \dots, q_{0,n})$ is the initial state;
3. $\delta_{\mathcal{P}} \subseteq Q_{\mathcal{P}} \times Q_{\mathcal{P}}$ is a transition relation, s.t. $((q, s, q_1, \dots, q_n), (q', s', q'_1, \dots, q'_n)) \in \delta_{\mathcal{P}}$ if and only if $(s, s') \in R$, $(q, L(s), q') \in \delta$, and $(q_i, L(s), q'_i) \in \delta_i$ for each $i \in \{1, \dots, n\}$;
4. $F_{\mathcal{P}} = F \times S \times Q_1 \times \dots \times Q_n$ is the set of accepting states;
5. $\mathbf{w} : Q_{\mathcal{P}} \rightarrow \{\text{T}, \text{F}\}^n$ is a state-weighting function that assigns to each state $(q, s, q_1, \dots, q_n) \in Q_{\mathcal{P}}$, a Boolean vector \mathbf{v} such that for any $1 \leq i \leq n$, it holds that $\mathbf{v}[i] = \text{T}$ if and only if $q_i \in F_i$.

This product automaton can be thought of as a Büchi automaton with a trivial alphabet, and thus, all definitions related to Büchi automata are applicable on it.

For a state $(q, s, q_1, \dots, q_n) \in Q_{\mathcal{P}}$, $\mathbf{w}((q, s, q_1, \dots, q_n))$ indicates which of the q_i 's were accepting in their original Büchi automata. Accordingly, for any $1 \leq i \leq n$, we use $F_{i,\mathcal{P}}$ to denote in \mathcal{P} , the set of all states that are accepting for automaton \mathcal{B}_i , i.e., $F_{i,\mathcal{P}} = \{p \in Q_{\mathcal{P}} \mid \mathbf{w}(p)[i] = \text{T}\}$. For a run $r_{\mathcal{P}} = q_0 q_1 q_2 \dots \in Q_{\mathcal{P}}^\omega$, we use $\mathbf{inf}(r_{\mathcal{P}})$ to denote a vector $\mathbf{v} \in \{\text{T}, \text{F}\}^n$ in which for each $1 \leq i \leq n$, $\mathbf{v}[i] = \text{T}$ if and only if there are infinitely many $j \geq 0$ such that $\mathbf{w}(q_j)[i] = \text{T}$. Subsequently, by having a cost function $f_{\mathbf{w}} : \{\text{T}, \text{F}\}^n \rightarrow \mathbb{Z}_{\geq 0}$, in which for any $\mathbf{v} \in \{\text{T}, \text{F}\}^n$,

$$f_{\mathbf{w}}(\mathbf{v}) = \sum_{i:\mathbf{v}[i]=\text{F}} n^{n-i}, \quad (7.2)$$

the cost of $r_{\mathcal{P}}$ will be $f_{\mathbf{w}}(\mathbf{inf}(r_{\mathcal{P}}))$. The purpose of constructing \mathcal{P} is to synthesize a run $r_{\mathcal{P}}$ that has the minimum cost. To see why, we first consider the following lemmas.

Lemma 17. Given the structures in Definition 30,

let $r_{\mathcal{P}} = (q_0, s_0, q_{0,1}, \dots, q_{0,n})(q_1, s_1, q_{1,1}, \dots, q_{1,n}) \dots$ be a run over \mathcal{P} . It holds that:

Algorithm 4: OPTIMALLTLPLANNINGWSOFTCONSTS

Data: $\mathcal{T}, \varphi, \psi_1, \psi_2, \dots, \psi_n$

Result: A path $\pi = s_0s_1s_2 \dots$ on \mathcal{T} s.t $\pi \models \varphi$ and $f_\omega(\text{trace}(\pi))$ is minimum

```
1 for  $i = 1$  to  $n$  do
2   |  $\mathcal{B}_i \leftarrow \text{LTL2BÜCHIAUTOMATON}(\psi_i)$ 
3   |  $\mathcal{B}_i \leftarrow \text{MAKENONBLOCKING}(\mathcal{B}_i)$ 
4  $\mathcal{A} \leftarrow \text{LTL2BÜCHIAUTOMATON}(\varphi)$ 
5  $\mathcal{P} \leftarrow \mathcal{A} \times \mathcal{T} \times \mathcal{B}_1 \times \mathcal{B}_2 \dots \mathcal{B}_n$ 
6  $r \leftarrow \text{MINIMUMCOSTACCEPTINGLASSO}(\mathcal{P})$ 
7 if  $r = \text{nil}$  then return nil
8 return  $\text{CONVERT2PATHONTS}(r)$ 
```

1. If $r_{\mathcal{P}}$ is accepting for \mathcal{P} , then the sequence $\pi = s_0s_1s_2 \dots$ is a path for \mathcal{T} such that $\text{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$.
2. For any $i \in \{1, \dots, n\}$, if $\mathbf{inf}(r_{\mathcal{P}})[i] = \text{T}$, then the sequence $\pi = s_0s_1s_2 \dots$ is a path for \mathcal{T} such that $\text{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{B}_i)$.

Proof. (1) From the construction of \mathcal{P} , it follows that the sequence $\pi = s_0s_1s_2 \dots$ —the projection of $r_{\mathcal{P}}$ onto \mathcal{T} — is a path over \mathcal{T} , and that the sequence $r = q_0q_1q_2 \dots$ is a run for $\text{trace}(\pi) = L(s_0)L(s_1)L(s_2) \dots$ over \mathcal{A} . Given that $r_{\mathcal{P}}$ is an accepting run for \mathcal{P} , there are infinitely many i 's for $r = q_0q_1q_2 \dots$ such that $q_i \in F$, implying that r is accepting, and thus, $\text{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$. (2) The proof is similar to the proof of (1) with the consideration that in this case, for each i , sequence $r_i = q_{0,i}q_{1,i}q_{2,i} \dots$ is a run for $\text{trace}(\pi) = L(s_0)L(s_1)L(s_2) \dots$ over \mathcal{B}_i . \square

Lemma 18. Assuming the structures in Definition 30, for any $I \subseteq \{1, \dots, n\}$ and for any path $\pi = s_0s_1s_2 \dots$ in \mathcal{T} such that $\text{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$ and $\text{trace}(\pi) \in \bigcap_{i \in I} \mathcal{L}_\omega(\mathcal{B}_i)$, there exists an accepting run $r_{\mathcal{P}} = (q_0, s_0, q_{0,1}, \dots, q_{0,n})(q_1, s_1, q_{1,1}, \dots, q_{1,n}) \dots$ over \mathcal{P} such that $\mathbf{inf}(r_{\mathcal{P}})[i] = \text{T}$ for all $i \in I$.

Proof. Let r be an accepting run for π over \mathcal{A} , and let for each $i \in I$, r_i be an accepting run for π over \mathcal{B}_i . Given that all Büchi automata created for the soft constraints are nonblocking, for each $j \in \{1, 2, \dots, n\}$ such that $j \notin I$, there exists an infinite run r_j for π over \mathcal{B}_j . Now we choose one such r , one such r_i for each i , and one such r_j for each j . Then we combine π , the chosen r , all the chosen r_i 's, and all the chosen r_j 's to form an $r_{\mathcal{P}}$. This constructed $r_{\mathcal{P}}$ has the properties claimed in this lemma. \square

The impact of these lemmas is that for any optimal solution of the OLPS problem, there is an accepting run $r_{\mathcal{P}}$ over \mathcal{P} for which $f_{\mathbf{w}}(\mathbf{inf}(r_{\mathcal{P}}))$ is minimum, and that from any accepting run $r_{\mathcal{P}}$ that minimizes $f_{\mathbf{w}}(\mathbf{inf}(r_{\mathcal{P}}))$, one can create an optimal solution to the OLPS problem via projecting $r_{\mathcal{P}}$ into \mathcal{T} . Accordingly, one can solve the OLPS problem by computing over \mathcal{P} , a run $r_{\mathcal{P}}$ with minimum cost.

Trajectory generation

A run $r_{\mathcal{P}}$ with minimum cost is constructed in Line 6 of Algorithm 4. The product automaton may have many, or even infinitely many optimal runs; in fact, there could exist an optimal run whose sequence of states cannot be specified by any pattern; however, we are interested in only one kind, which is revealed by the following result.

Lemma 19. If $\mathcal{L}_{\omega}(\mathcal{P}) \neq \emptyset$, then \mathcal{P} has an accepting lasso $r_{\mathcal{P}} = r_1(r_2)^{\omega}$ such that $r_1 \in Q_{\mathcal{P}}^*$, $r_2 \in Q_{\mathcal{P}}^+$, and that $f_{\mathbf{w}}(\mathbf{inf}(r_{\mathcal{P}}))$ is minimum.

Proof. We show that from any accepting run $r'_{\mathcal{P}} = p_0p_1p_2\dots$ minimizing $f_{\mathbf{w}}(\mathbf{inf}(r'_{\mathcal{P}}))$, we can construct an accepting lasso $r_{\mathcal{P}}$ such that $f_{\mathbf{w}}(\mathbf{inf}(r'_{\mathcal{P}})) = f_{\mathbf{w}}(\mathbf{inf}(r_{\mathcal{P}}))$. Given that $r'_{\mathcal{P}}$ has an infinite length while $Q_{\mathcal{P}}$ has only a finite number of states, there exists an integer $k \geq 0$ such $\mathbf{inf}(r'_{\mathcal{P}}) = \{p_j \in r'_{\mathcal{P}} \mid j \geq k\}$. We choose l to be the smallest such k .

Algorithm 5: MinimumCostAcceptingLasso

Data: Product automaton $\mathcal{P} = (Q_{\mathcal{P}}, \delta_{\mathcal{P}}, q_{0,\mathcal{P}}, F_{\mathcal{P}}, \mathbf{w})$
Result: An accepting lasso for \mathcal{P} minimizing $f_{\mathbf{w}}$

- 1 $SCCs = \text{STRONGLYCONNECTEDCOMPONENTS}(\mathcal{P});$
- 2 $O \leftarrow \text{nil};$
- 3 $minW \leftarrow \infty;$
- 4 **forall** $C \in SCCs$ **do**
- 5 **if** $C.accepting = \text{True}$ **then**
- 6 **if** $f_{\mathbf{w}}(C.\mathbf{w}) < minW$ **then**
- 7 $O \leftarrow C;$
- 8 $minW \leftarrow f_{\mathbf{w}}(C.\mathbf{w});$
- 9 **if** $O = \text{nil}$ **then return nil**
- 10 $r_1 = \text{BFSSHORTESTPATH}(q_{0,\mathcal{P}}, O.leader);$
- 11 $r_2 = \text{MINCOSTACCEPTINGCYCLE}(O);$
- 12 **return** $(r_1, r_2);$

Let $I = \{1 \leq i \leq n \mid \mathbf{inf}(r'_{\mathcal{P}})[i] = \text{T}\}$. We choose an integer $j \geq l$ such that $r'_{\mathcal{P}}[l..j]$ contains at least one state $p \in F_{\mathcal{P}}$ and it contains at least a state $q_i \in F_{i,\mathcal{P}}$ for each $i \in I$. We choose u to be the smallest such integer j . We now set $r_2 = r'_{\mathcal{P}}[l..u] = p_l p_{l+1} \dots p_u$ and set $r_1 = r'_{\mathcal{P}}[0..l-1] = p_0 p_1 \dots p_{l-1}$. Clearly, lasso $r_{\mathcal{P}}$ is accepting. Moreover, $f_{\mathbf{w}}(\mathbf{inf}(r'_{\mathcal{P}})) = f_{\mathbf{w}}(\mathbf{inf}(r_{\mathcal{P}}))$. \square

The punchline is that to synthesize an optimal run, it is sufficient to consider those runs who are lassos. We are also interested in finding a shortest such lasso—a lasso $r_{\mathcal{P}} = r_1(r_2)^w$ for which $|r_1| + |r_2|$ is minimum. Unfortunately, the following result reveals that finding a shortest such lasso is not easy.

Lemma 20. Given a product automaton \mathcal{P} , the problem of finding over \mathcal{P} , a shortest lasso $r_{\mathcal{P}} = r_1(r_2)^w$ that minimizes $f_{\omega}(r_{\mathcal{P}})$ is NP-hard.

Proof. We prove by reduction from the problem of finding a shortest accepting lasso for a generalized Büchi automaton, which is known to be NP-hard [25, 36]. For each generalized Büchi automaton $\mathcal{G} = (Q, \Sigma, \delta, q_0, \mathcal{F} := \{F_1, F_2, \dots, F_n\})$, we make a product automaton $\mathcal{P} = (Q_{\mathcal{P}}, \delta_{\mathcal{P}}, q_{0,\mathcal{P}}, F_{\mathcal{P}}, \mathbf{w})$ such that $Q_{\mathcal{P}} = Q$; $q_{0,\mathcal{P}} = q_0$; $F_{\mathcal{P}} = Q$;

for each $q, q' \in Q$, $(q, q') \in \delta_{\mathcal{P}}$ iff $(q, a, q') \in \delta$ for an $a \in \Sigma$; and for each state $q \in Q_{\mathcal{P}}$, function \mathbf{w} assigns a vector $\mathbf{v} \in \{\text{T}, \text{F}\}^n$ such that for each $j \in \{1, 2, \dots, n\}$, $\mathbf{v}[j] = \text{T}$ if $q \in F_j$, and otherwise, $\mathbf{v}[j] = \text{F}$. Consider that any run over \mathcal{P} is accepting, and that $f_{\mathbf{w}}(\mathbf{inf}(r_{\mathcal{P}})) = 0$ for any optimal lasso $r_{\mathcal{P}}$ in \mathcal{P} . Any shortest lasso $r_{\mathcal{P}}$ over \mathcal{P} for which $f_{\mathbf{w}}(\mathbf{inf}(r_{\mathcal{P}})) = 0$ is a shortest accepting lasso for \mathcal{G} . This completes the proof. \square

As a result of this lemma, unless $\text{P}=\text{NP}$ we cannot compute in a time polynomial to the size of \mathcal{P} , a shortest lasso that is accepting for \mathcal{P} and for which $f_{\mathbf{w}}(r_{\mathcal{P}})$ is minimum. Unfortunately, it is also NP-hard to approximate within any constant factor, the length of such a lasso (the proof would utilize the same reduction in Lemma 20 along with the fact due to Ehlers [36], according to which it is NP-hard to approximate within any constant factor the length of a shortest accepting lasso for a generalized Büchi automaton). Consequently, we utilize a greedy algorithm to find a shortest such lasso which has the minimum cost.

Our algorithm uses graph algorithms to minimize $|r_1|$ and $|r_2|$ separately. Algorithm 5 shows the process. Consider that \mathcal{P} can be thought of as a directed graph with vertex set $Q_{\mathcal{P}}$ and edge set $\delta_{\mathcal{P}}$. Additionally, all vertices (states) in r_2 are in a *strongly connected component* (SCC) of the graph given that they are contained in a cycle, $r_2.r_2[0]$. With these in mind, our algorithm first decomposes the graph into its strongly connected components, (Line 1); then finds a SCC that contains $|r_2|$ of a lasso $r_{\mathcal{P}} = r_1(r_2)^{\omega}$ with minimum $f_{\mathbf{w}}(r_{\mathcal{P}})$ (Lines 2–8); and then construct r_1 and r_2 (Lines 10 and Line 11 respectively). See Figure 7.1.

To find the set of SCCs of the graph, we use the well-known algorithm of Tarjan [142]. This algorithm uses depth first search (DFS) to traverse all the vertices (states) of the graph in one pass. During this traversal, each vertex p is assigned a unique integer $p.number$, which is, in fact, the traversal's step number at which p is reached. Each vertex is assigned another integer $p.lowlink$, whose value is set to the

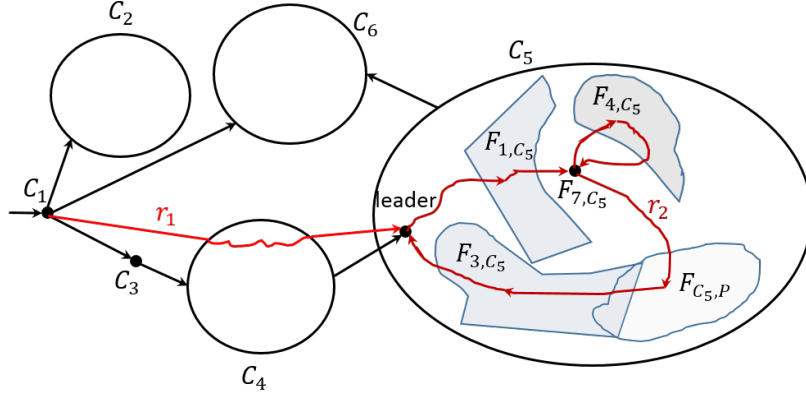


Figure 7.1: Showing our algorithm for finding an optimal lasso $r_{\mathcal{P}} = r_1(r_2)^\omega$ over product automaton \mathcal{P} . Each C_i is a strongly connected component of the graph underlying \mathcal{P} . Component C_5 contains the suffix of an optimal lasso. Set $F_{C_5, \mathcal{P}}$ contains those state in C_5 that are accepting for \mathcal{P} , i.e., $F_{C_5, \mathcal{P}} = F_{\mathcal{P}} \cap C_5$. For each $i \in \{1, 3, 4, 7\}$, set F_{i, C_5} are those states in C_5 that are accepting for Büchi automaton \mathcal{B}_i —the one who represents preference ψ_i .

smallest index of any node reachable from p , including p itself. During this algorithm, all vertices that are assigned the same value of *lowlink* will be in the same SCC of the graph, and among those vertices, the one whose number is equal to its *lowlink* is the leader (representative) of the SCC.

As Tarjan’s algorithm executes, we also compute for each SCC C , the value of $C.\text{accepting}$, which gets *True* only when C contains an accepting state of \mathcal{P} and that C is not a singleton *vertex* who does not have a loop. We also compute the value of Boolean vector $C.\mathbf{w}$, whose value is set to $C.\mathbf{w} = \sum_{q \in C} \mathbf{w}(q)$. Notice that during the same pass of the Tarjan’s algorithm, one can keep for each vertex, a link to its parent in the DFS search. Accordingly, later, the algorithm can use those links to find, for each state, a path from the initial state to that state. These paths can be used in Line 10 to compute r_1 , which is, in fact, a path from the initial state to the leader of the component which we choose to construct r_2 from. However, a path r_1 that is constructed in that way may not have minimum length. Accordingly, we use BFS to find a shortest simple path from $q_{0, \mathcal{P}}$ to $C.\text{leader}$.

The final phase of finding an accepting lasso is to synthesize the suffix r_2 of it,

Line 11 of Algorithm 5. This suffix is synthesized inside an optimal SCC O using the following greedy algorithm. Let $r'_P \in Q_P^\omega$ be any run that minimizes $f_\omega(r'_P)$, and let $I = \{1 \leq i \leq n \mid \mathbf{w}(r'_P)[i] = \text{T}\}$. Let for each $i \in I$, $F_{i,O}$ be the set of states in O that are accepting for Büchi automaton \mathcal{B}_i , i.e., $F_{i,O} = O \cap \{q \in Q \mid \mathbf{w}(q)[i] = \text{T}\}$, and let $F_{O,P}$ be the set of accepting states in O , i.e., $F_{O,P} = F_P \cap O$. Our greedy algorithm synthesizes r_2 as the vertices (states) of a cycle starting from $O.\text{leader}$ such that the cycle contains at least a state of $F_{O,P}$ and at least a state in $F_{i,P}$ for each $i \in I$. To do so, it uses variable U with initial value $F_{O,P} \cup \{F_{i,O} \mid i \in I\}$. It starts from $O.\text{leader}$, and performs *breadth first search* (BFS) until it finds a state s for which there is a set $M \in U$ such that $s \in M$. Using the parent links set during BFS, the algorithm records the shortest path from $O.\text{leader}$ to s as a first portion of r_2 , and removes from U all those sets M for which $s \in M$. It then does a similar task, BFS traversal, starting from s , and then records the path traversed from s to the new found state. It repeatedly does this process until U becomes \emptyset . At this time, it does a BFS to find the shortest path back to $O.\text{leader}$. By this time, it has made r_2 as the vertices of a cycle. Figure 7.1 illustrates how inside SCC C_5 , the algorithm constructs r_2 .

Given this discussion, we now analyze the time complexity of our algorithm.

Lemma 21. For any automaton $\mathcal{P} = (Q_P, \delta_P, Q_{0,P}, F_P, \mathbf{w})$, a lasso with minimum cost can be generated in time $\mathcal{O}(n(|\delta_P| + |Q_P|))$.

Proof. It takes $\mathcal{O}(|\delta_P| + |Q_P|)$ time to find the set of strongly connected components. BFS is called at most $n + 1$ times, each of which takes $\mathcal{O}(|\delta_P| + |Q_P|)$ time. \square

This bound simplifies to $\mathcal{O}(|\delta_P| + |Q_P|)$ if the number of soft constraints n is treated as a constant.

We can slightly improve the quality of solution by letting instead of the leader, the first state to which BFS reaches from the leader and who is either a final state or

it corresponds to a final state of the Büchi automata for a soft constraint, to be the midpoint of the lasso.

Though Algorithm 5 generates a lasso of minimum cost, it is not guaranteed to produce the shortest such lasso. In the next section, we compare our algorithm with an optimal brute-force algorithm, which finds the shortest lasso by letting any state within an optimal SCC to be the midpoint of the lasso, and synthesizes the suffix of the lasso by searching from the shortest ones, all cycles, simple or otherwise, that start from the midpoint until it finds a satisfactory cycle or the length will be longer than the length of a shortest lasso computed for other midpoints.

7.3 CASE STUDIES AND EXPERIMENTS

We have implemented Algorithms 4 and 5 in Java. The computed results were executed on an Ubuntu 16.04 computer with a 3.6GHz processor.

Case study: Hospital deliveries

Figure 7.2 shows a hospital which has an emergency department (e), a primary care department (p), a maintenance department (n), a pharmacy (h), and a warehouse (w). In this hospital, a robot delivers first aid items (f) and medicine (m) from the warehouse to the other departments. Each state of the transition system for this case study has an atomic proposition indicating a unit within the hospital, along with other propositions indicating which items the robot is caring at that unit. Those states are connected according to the connectivity of the units within the hospital and the items the robot can take or leave at a unit. The robot's primary tasks are to *deliver first aid items to the emergency department, deliver first aid items to primary care, deliver medicine to the pharmacy, and report for maintenance, each infinitely often*. In addition, suppose the robot is given these soft specifications, ordered from

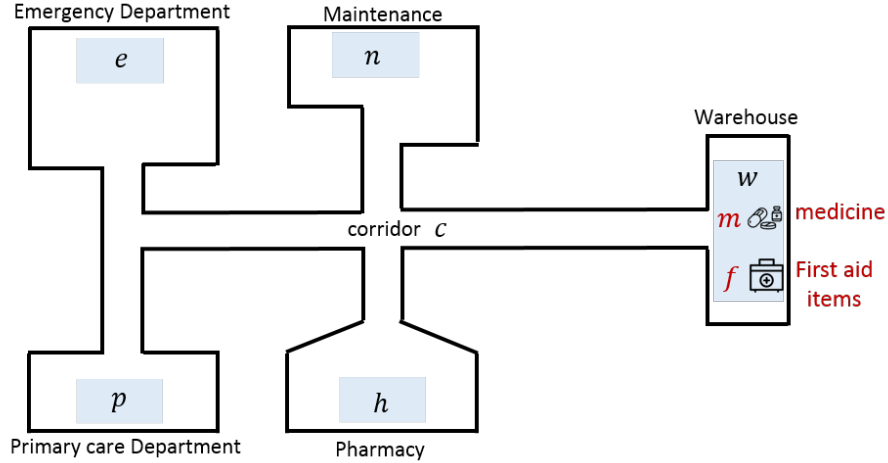


Figure 7.2: A hospital, in which a delivery robot is tasked to deliver first aid items to emergency and primary care departments, deliver medicine to pharmacy, and visit the maintenance section. The robot’s task is expressed by LTL formula $\varphi = \square\Diamond(p \wedge f) \wedge \square\Diamond(e \wedge f) \wedge \square\Diamond(h \wedge m) \wedge \square\Diamond n$

most to least important:

1. If first aid items are delivered to the primary care department, then do not deliver additional first aid items there until first aid items have been delivered to the emergency department, and vice versa.
2. If the first aid items are picked from the warehouse, then they must not be delivered to the primary care department until the emergency department receives the first aid items.
3. Do not carry first aid items and medicine together.
4. Always pick both first aid items and medicine when leaving warehouse.

Notice that, in particular, the first two constraints cannot be expressed in LDL_f . Thus, the algorithm from our prior work [107] cannot generate a plan for this instance. The box below shows how to formulate these constraints into an instance of OLPSC, along with the solution computed by our implementation.

Goal: $\Box\Diamond(p \wedge f) \wedge \Box\Diamond(e \wedge f) \wedge \Box\Diamond(h \wedge m) \wedge \Box\Diamond n$

Soft constraints:

- 1) $\Box((p \wedge f) \rightarrow \bigcirc(\neg p \mathcal{U}(e \wedge f)))$
 $\wedge \Box((e \wedge f) \rightarrow \bigcirc(\neg e \mathcal{U}(p \wedge f)))$
- 2) $\Box((c \wedge f) \rightarrow (\neg p \mathcal{U}e))$
- 3) $\Box(\neg w \rightarrow \neg(f \wedge m))$
- 4) $\Box((w \wedge \bigcirc c) \rightarrow \bigcirc(f \wedge m))$

Solution:

$wcp(cecw_f c_f p_f c w_f c_f e_f c w_m c_m h_m h_m h c n n c p c)^\omega$

Satisfied constraints: 1, 3

Computation time: 201.50s

In the sequence shown for the solution, a letter is the location of the robot, and the subscript of the letter, if any, is what the robot is carrying. Not that this optimal solution satisfies only the first and third soft constraints.

For comparison purposes, we also executed on this example, the brute-force algorithm to compute a shortest accepting lasso that minimizes the cost function f_ω . That algorithm failed to compute such a lasso in 15 hours.

Case study: Retirement home

In this section, we consider a social enrichment robot, capable of making animal balloons and juggling, visits the residents of a retirement home. See Figure 7.3. The robot's basic mission is to visit the two common rooms, each infinitely often. In addition to this basic mission, however, the robot is also charged with satisfying a collection of soft constraints, given in order of their relative importance. For example, we might prefer to maintain fairness by ensuring, if possible, that after making animal balloons in room 1, it should also do the same act in room 2. Or perhaps the manager

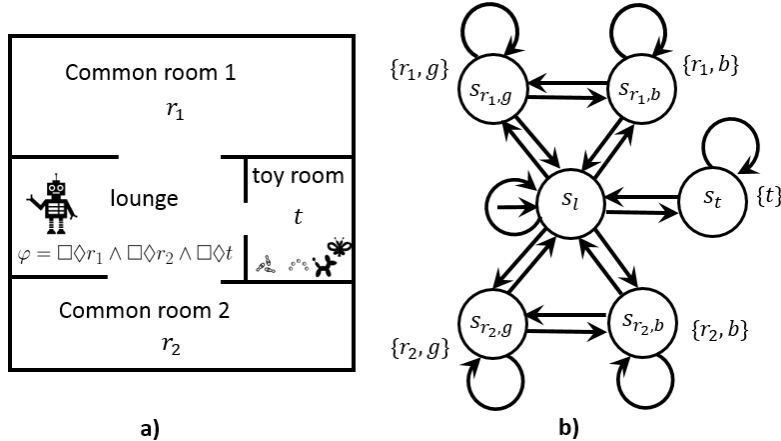


Figure 7.3: **a)** A retirement home in which a social enrichment robot visits each of the common rooms to perform juggling and to make animal balloons. Its primary mission is expressed by the LTL formula $\varphi = \square\Diamond r_1 \wedge \square\Diamond r_2 \wedge \square\Diamond t$. **b)** A transition system that models the robot’s state within this environment.

wants the robot to eventually perform juggling in room 2, if its current act in that room is making animal balloons. The residents of room 2 might even prefer not to see the balloon animal act at all. The essence of our problem is to determine how the robot can act, to satisfy its primary mission, along with some optimal subset of these kinds of soft constraints.

The transition system in that example has atomic propositions for locations — r_1 for common room 1, r_2 for common room 2, and t for toy room— and also for the robot acts — g for juggling, and b for making animal balloons. The robot is tasked to *visit r_1 , r_2 , and t , each one infinitely often*. The robot’s specification also includes 6 soft constraints:

1. After making animal balloons in room 1, the robot should immediately do the same act in room 2.
2. The robot should perform each of the acts in room 1 infinitely often.
3. The robot should not perform the act of making animal balloons in room 2.

4. If the current act in room 2 is making animal balloons, then the robot should eventually make animal balloons in that room.
5. The robot should not stay in a common room once it performed an act.
6. The robot should perform at least two acts in each common room once it has entered that room. Those two acts can be different or not.

We can formalize this scenario as an instance of OLPSC:

Goal: $\Box\Diamond r_1 \wedge \Box\Diamond r_2 \wedge \Box\Diamond t$

Soft constraints:

- 1) $\Box((r_1 \wedge b) \rightarrow \bigcirc((\neg b \wedge \neg g) \mathcal{U}(r_2 \wedge b)))$
- 2) $\Box\Diamond(r_1 \wedge g) \wedge \Box\Diamond(r_1 \wedge b)$
- 3) $\Box(r_2 \rightarrow \neg b)$
- 4) $\Box((r_2 \wedge b) \rightarrow \Diamond(r_2 \wedge g))$
- 5) $\Box(r_1 \rightarrow \bigcirc\neg r_1) \wedge \Box(r_2 \rightarrow \bigcirc\neg r_2)$
- 6) $\Box((\neg r_1 \wedge \bigcirc r_1) \rightarrow (\bigcirc \bigcirc r_1))$

Solution:

$$s_l s_{r_1, g} s_l s_{r_2, b} (s_l s_{r_2, b} s_l s_{r_2, g} s_l s_{r_2, b} s_l s_{r_1, g} s_l s_{r_1, b} s_l s_{r_2, b} s_l s_{r_1, b} s_l s_{r_2, b} s_l s_t s_t s_l s_{r_2, b} s_l)^\omega$$

Satisfied constraints: 1, 2, 4, 5

Computation time: 21.91s

For comparison, the brute-force algorithm computed $r_{\mathcal{P}}^* = s_l s_{r_1, g} s_{r_1, b} s_l s_{r_2, b} s_{r_2, g} s_l s_t s_l (s_{r_1, g} s_{r_1, b} s_l s_{r_2, b} s_{r_2, g} s_l s_t s_l)^\omega$ as a shortest accepting lasso minimizing f_ω in 3,254 seconds. The shortest accepting lasso has length 16, while the lasso generated by our algorithm has length 26.

Now suppose there is a change in the relative ordering of the soft constraints, in which the last two are swapped. This induces a change to the set of constraints that can be satisfied, but not to the basic structure of the product automaton.

Goal: $\Box\Diamond r_1 \wedge \Box\Diamond r_2 \wedge \Box\Diamond t$

Soft constraints:

- 1) – 4) Same as above.
- 5) $\Box((\neg r_1 \wedge \bigcirc r_1) \rightarrow \bigcirc \bigcirc r_1)$
- 6) $\Box(r_1 \rightarrow \bigcirc \neg r_1) \wedge \Box(r_2 \rightarrow \bigcirc \neg r_2)$

Solution:

$$s_l s_{r_2, b} (s_{r_2, g} s_{r_2, b} s_{r_2, g} s_l s_{r_1, g} s_{r_1, b} s_l s_{r_2, b} s_{r_2, g} s_l s_{r_1, g} s_{r_1, b} s_l s_{r_2, b} s_{r_2, g} s_l s_t s_t s_l s_{r_2, b} s_{r_2, g})^\omega$$

Satisfied constraints: 1, 2, 4, 5

Computation time: 0.02s (excluding product automaton construction)

In fact, if we have already computed the product automaton for the instance above, we need only to synthesize an accepting lasso, without any need to reconstruct the product automaton again. To synthesize the new lasso, for each state of the product automaton, we swap the elements of vector \mathbf{w} according to the new order of constraints, and then synthesize the lasso.

It took only 20 milliseconds to synthesize an optimal run on the constructed automaton, while for the first one it took 21.91 seconds, much of which was spent to form the product automaton. We also use the brute-force algorithm to compute the shortest accepting lasso $r_{\mathcal{P}}^*$ that minimizes the $f_\omega(r_{\mathcal{P}}^*)$. It took 110 seconds, excluding the time of product automaton construction, for the brute-force algorithm to compute the shortest accepting lasso, which was $s_l s_{r_1, g} s_{r_1, b} s_l s_{r_2, b} s_{r_2, g} s_l s_t s_l (s_{r_1, g} s_{r_1, b} s_l s_{r_2, b} s_{r_2, g} s_l s_t)^\omega$, with length 16. Observe that the length of the lasso generated by our greedy algorithm was 23 while the length of the shortest accepting lasso was 16. The product automaton for this problem had 1440 states.

States	Our algorithm			Brute-force algorithm			Approximation ratio		
	#Success	Avg Time	Avg lasso size	#Success	Avg Time	Avg lasso size	Min	Max	Avg
100	100	0.0001	17.86	100	12.404	9.27	1	4.33	2.08
200	100	0.0002	21.88	100	53.91	10.39	1	4.33	2.24
300	100	0.0001	23.68	96	170.151	10.93	1	4.4	2.26
500	00	0.0001	28.14	62	504.799	11.63	1	4.14	2.35

Figure 7.4: Results of our experiment comparing our algorithm with the brute-force algorithm. The average times are in seconds.

Experiments

In this section, we present results of our experiment comparing our (greedy) algorithm with the brute-force algorithm. We performed all those experiments on the same machine on which we executed our case studies. In this experiment, we execute both the greedy algorithm and the brute-force algorithm on a large number of graphs (product automata) of different sizes that we generated randomly by the Erdős-Rényi model of $G(n, p)$, according to which each edge will be included in the graph with probability p independent from any other edge.

Figure 7.4 shows results of our experiment. For each graph size —100, 200, 300, and 500— we generated 100 random graphs. The number of edges for each graph was approximately five times the number of vertices, and for each graph, approximately 20 percent of the states were final states. The number of soft constraints, the size of the Boolean vectors assigned to a state by \mathbf{w} , for each graph was 10. We report for each graph size, the average time to compute an accepting lasso with minimum cost by our algorithm, and also the average size of the generated lassos. For each test, the greedy algorithm had 20 minutes to find a shortest lasso. Figure 7.4 shows also for each graph size, the number of times the brute-force algorithm was able to compute a shortest lasso within the 20 minutes time window. Notice that for graph size 300, the brute-force algorithm failed four times to compute a minimal lasso within that time window, and for graph size 500, it was able to compute a shortest lasso only

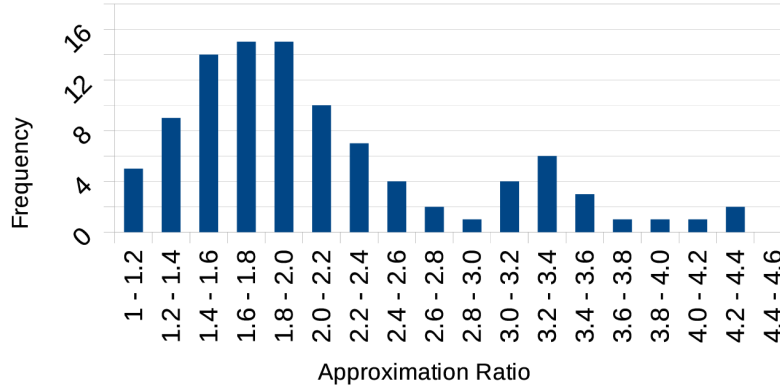


Figure 7.5: The distribution of approximation ratios of the lasso sizes generated by our greedy algorithm for 100 generated random graphs with 100 states.

for 62 tests. Accordingly, we considered in computing all those averages shown for the brute-force algorithm in that figure, only those tests for which the algorithm was able to compute a solution in 20 minutes. This means that the actual time averages for graph size 300 and 500 are higher, and probably much higher, than those shown in that figure. The average number of constraints satisfied were ranged from 6.30 to 6.81. f

For each test, we also computed approximation ratio, defined as the size of the lasso generated by the greedy algorithm over the size of the lasso generated by the brute-force algorithm. The minimum, maximum, and the average of those ratios for each graph size is shown in Figure 7.4. We observe from this experiment that the greedy algorithm generates a solution significantly faster than the brute-force algorithm while the quality of solution is still acceptable. For graph size 100, the distribution of the approximation ratios of the 100 tests we performed is shown in Figure 7.5.

We also executed a variant of our algorithm, in which we let any final state within an optimal SCC to be the midpoint and then chose a shortest one among all lassos generated for those midpoints. This algorithm increases the running time by the magnitude of the number of final states. We observed that the quality of solution is

slightly improved. For the graph sizes of 100, the average of approximation ratio was 2.01 for this new variant, compared to 2.08 to original algorithm. Because product automata are generally quite large, we may not need sacrifice computation time in favor of slightly improved quality.

CHAPTER 8

PLANNING TO CHRONICLE WITH SOFT CONSTRAINTS

In this chapter, we consider a problem in which a robot is tasked with recording a structured video from the events occurring in an unpredictable environment while it should optimally satisfy an ordered list of soft constraints on the movement of the robot and the story. This problem combines the problem settings in Chapter 5 and Chapter 6.

The organization of this chapter is as follows. In Section 8.1, we present preliminaries and problem definition. In Section 8.2, we present our algorithm. In Section 8.3, we present a case study.

8.1 PRELIMINARIES AND PROBLEM DEFINITION

We first introduce preliminary tools.

We again assume that the set of all possible events is a finite alphabet E . We model the environment using a transition system $\mathcal{T} = (W, R, w_0, AP, L)$ in which W is the state space, $R \subseteq W \times W$ is the transition relation, w_0 is the initial state, AP is a set of atomic propositions, and $L : W \rightarrow 2^{AP}$ is the labeling function. See Definition 23, in Chapter 6, for details about transition systems.

Each state of the transition system represents a physical location (region) in the world and the relation R is formed based on the connectivity of those locations (regions).

We extend Definition 13, the definition of event model in Chapter 5, as follows to

model the physical world along with the occurrences of events.

Definition 31. [Event-Location Model] An *event-location model* is a tuple $\mathcal{M} = (S, P, s_0, E, g)$ in which

- S , which is a nonempty finite set, is the *state space* of the model;
- $P : S \times S \rightarrow [0, 1]$ is the *transition probability function* of the model, such that for each state $s \in S$, $\sum_{s' \in S} P(s, s') = 1$;
- $s_0 \in S$ is the *initial state*;
- E is the set of all possible events;
- $g : S \times E \times W \rightarrow [0, 1]$ is the event-location occurrence function such that for each state $s \in S$, event $e \in E$, and location $w \in W$, $g(s, e, w)$ is the probability that event e happens at state s and location w . We assume that $g(s_0, e, w) = 0$ for any event e and state $w \in W$, meaning that no event happens at state s_0 .

In this chapter, we consider only the case where the current state of the event model is observable to the robot, that is, at each time step, the robot knows in which state the event model is, but it does not know to which state the event model transitions in the next time step.

We specify the story and the constraints on how to record a story using LDL_f . See Definition 27 and Definition 28 for syntax and semantics of LDL_f . Each LDL_f formula is defined over the extended set of atomic propositions $AP' = AP \cup E \cup W$. To illustrate the idea of LDL_f for specifying a story, consider again the environment in Figure 1.4. An example of LDL_f formula is $\varphi_1 = \langle \text{true}^* \rangle (b_i \wedge \langle \text{true}^* \rangle (l_b \wedge f)) \wedge \langle \text{true}^* \rangle (l_i \wedge j)$. Formula φ_1 specifies that the robot must capture a story that is a supersequence for both sequences $b_i l_b$ and l_i and that the event l_b is taken in the grass field while the event l_i is taken in the jungle. This formula specifies not only the story but also how the story is captured.

For another example, consider $\varphi_2 = [true^*] \neg(j \wedge l_i)$. This formula does not specify the story but it imposes the constraint that the robot must not record any event lion-eating-impala in the jungle. We use LDL_f for specifying the soft constraints as well.

The system evolves along a path $s_0 s_1 s_2 \dots$ and for each $k > 0$, when the system enters s_k , for each event $e \in E$ and location $w \in W$, event e happens with probability $g(s, e, w)$ at location w . The robot attempts to record some of the occurring events to form a story. At each time step $k \geq 0$, the robot chooses a location w , reachable from its current location in one step, to go to the next time step; it also either chooses an event $e \in E$ to record at w in the next step or decides to recording nothing at that time step. The former case applies, for example, to a situation where the robot needs to go through several locations without attempting to record anything at those locations to reach a desired location to record an event of interest at there. With these in mind, the robot action at each step is a tuple $(u, w) \in (E \cup \{\epsilon\}) \times W$, where w is a location and u is an event and the robot wants to record u at w in the next time step. If at step $k + 1$ event e happens at location w , then the robot successfully records e ; otherwise, it records nothing. In this chapter we again assume that the robot is aware of the success or failure of each of its attempts. If $u = \epsilon$, then it means the robot does not want to record any event at $k + 1$. The story and the hard constraints on how to record that story is specified by an LDL_f formula φ . The robot is given, apart from φ , a list of soft constraints $\psi_1, \psi_2, \dots, \psi_n$ ordered from the highest priority to the lowest priority. The aim of our problem is to compute a policy that guarantees to always satisfy the constraints with higher priorities, or equivalently, violates the soft constraints with lower priorities, if it has to violate any soft constraint. Note that by results of De Giacomo and Vardi [31], for the LDL_f formula φ , which is defined over the set of atomic propositions AP' , there is a DFA $\mathcal{D} = (Q, 2^{AP'}, q_0, \delta, F)$ such that $\mathcal{L}(\mathcal{D}) = \text{traces}(\varphi)$. As such, the robot computes the DFA \mathcal{D} from the formula specifying the story and maintains it. In a similar manner, it computes for each

soft constraint ψ_i for $i \in \{1, 2, \dots, n\}$, a DFA $\mathcal{D}_i = (Q_i, 2^{AP'}, q_{i,0}, \delta_i, F_i)$ such that $\mathcal{L}(\mathcal{D}_i) = \text{traces}(\psi_i)$, and maintains \mathcal{D}_i as well.

The robot stops its execution once it has completed its mission by recording a desirable story. For each time step k , the robot maintains several pieces of information: s_k , the current state of the event model; w_k , the location at which the robot is; $\eta_k \in E^*$, the story that has captured until time step k ; and $\xi_k \in ((E \cup \{\epsilon\}) \times W)^k$, the recorded *story-location* up to time step k . Note that $\xi_k = (d_0, w_0)(d_1, w_1) \cdots (d_k, w_k)$ and for each $0 \leq j \leq k$, d_j is an event the robot has recorded at location w_j . Again, if the robot recorded nothing at location w_j , then $d_j = \epsilon$.

For each time step $k \geq 0$ and story-location $\xi_k = (d_0, w_0)(d_1, w_1) \cdots (d_k, w_k) \in ((E \cup \{\epsilon\}) \times W)^k$, we let $\text{trace}(\xi_k) \in (AP')^*$ to be the trace of the story-location ξ_k such that $\text{trace}(\xi_k) = t_0 t_1 \cdots t_k$ and for each for each $0 \leq j \leq k$, $t_j = L(w_j) \cup \{w_j\}$ if $d_j = \epsilon$, and otherwise $t_j = L(w_j) \cup \{w_j, d_j\}$. Intuitively, for each location-event sequence, this function produces a trace where each element of the trace is a set of atomic propositions associated with the location and the event of that element.

Additionally, each story-location ξ_k is mapped to a unique state $q_k \in Q$, which is, in fact, the state to which the DFA \mathcal{D} reaches by tracking $\text{trace}(\xi_k)$ from the initial state q_0 . Also, for each DFA $\mathcal{D}_i = (Q_i, 2^{AP'}, q_{i,0}, \delta_i, F_i)$ for $i \in \{1, 2, \dots, n\}$, the story-location ξ_k is mapped to a unique state $q_{i,k} \in Q_i$, which is again the state to which \mathcal{D}_i reaches by tracking $\text{trace}(\xi_k)$ from the initial state $q_{i,0}$. As such, the robot maintains for each time step k , apart from the data mentioned above, the current state q_k of the DFA \mathcal{D} , and the current state $q_{i,k}$ of the DFA \mathcal{D}_i for each $i \in \{1, 2, \dots, n\}$ as well.

The robot uses a policy $\pi : S \times Q \times W \times Q_1 \times \cdots \times Q_n$ to choose which event-location to try at each time step.

Assuming $\pi(s_k, q_k, w_k, q_{1,k}, \dots, q_{n,k}) = (w_{k+1}, u_{k+1})$, the robot updates ξ_{k+1} from ξ_k based on the following formula:

$$\xi_{k+1} = \begin{cases} \xi_k \cdot \pi(s_k, q_k, w_{k+1}, q_{1,k}, \dots, q_{n,k}) & u_k \neq \epsilon \text{ and} \\ & g(s_{k+1}, u_{k+1}, w_{k+1}) > 0 \text{ and} \\ & u_k \text{ happened at } w_{k+1} \\ \xi_k \cdot (w_{k+1}, \epsilon) & \text{otherwise} \end{cases} \quad (8.1)$$

It computes η_{k+1} in a similar fashion.

Initially, $\xi_0 = \epsilon$ and $\eta_0 = \epsilon$. The robot changes the value of variable q_k only when the guessed event actually happened:

$$q_{k+1} = \begin{cases} \delta(q_k, L(w_{k+1}) \cup \{w_{k+1}, u_k\}) & u_k \neq \epsilon \text{ and} \\ & g(s_{k+1}, u_{k+1}, w_{k+1}) > 0 \text{ and} \\ & u_k \text{ happened at } w_{k+1} \\ \delta(q_k, L(w_{k+1}) \cup \{w_{k+1}\}) & \text{otherwise} \end{cases} \quad (8.2)$$

It uses a similar formula to compute $q_{i,k+1}$ from $q_{i,k}$ and result of its attempt for each $i \in \{1, 2, \dots, n\}$.

The robot stops when $q_k \in F$, i.e., when the robot has accomplished its task.

Let Π be the set of all policies. For each policy $\pi \in \Pi$ and soft constraint ψ , we use $Pr_\pi(\psi)$ to denote the probability that a recorded story-location under policy π satisfies ψ . Accordingly, $1 - Pr_\pi(\psi)$ is the probability that π violates ψ .

We define $\mathbf{J} : \Pi \rightarrow [0, 1]^n$ such that for each policy π , $\mathbf{J}(\pi) = [1 - Pr_\pi(\psi_1), 1 - Pr_\pi(\psi_2), \dots, 1 - Pr_\pi(\psi_n)]$, represents for each soft constraint ψ_i , the probability that ψ_i is violated by π . One purpose of our problem is to compute a policy that minimizes this function. To be able to tell when this function is minimized, one need to define a pre-order on the co-domain of this function. That pre-order can be simply defined as the lexicographical order over the co-domain. To see an example, that pre-order gives a higher priority to $[0.3, 0.9]$ over $[0.4, 0.1]$. Of course, one can define a pre-order

in which the probabilities that are smaller than a given threshold λ are converted to zero in certain situations. For example, if $\lambda = 0.05$, then $[0.04, 0.4, 0.9]$ has the better cost of violation over $[0, 0.6, 0.1]$, even though the later comes before the former in the standard lexicographical order. In this work, we define the pre-order by linearizing the values within the co-domain of \mathbf{J} using a function $f : [0, 1]^n \rightarrow \mathbb{R}_{\geq 0}$ such that for each policy π ,

$$f(\mathbf{J}(\pi)) = \mathbf{J}(\pi) \cdot [m^{n-1}, m^{n-2}, \dots, m^0] = \sum_{i=1}^n (1 - Pr_{\pi}(\psi_i)) m^{n-i}, \quad (8.3)$$

in which m is chosen suitably based on to what extent we want to discard small probabilities. Accordingly, a policy that minimizes this function is assumed to have the smallest cost of violation of the soft constraints.

Additionally, we are interested only in a *correct policy*, a policy that guarantees, with probability 1.0, a story-location that satisfies φ is captured.

We now define our problem.

Problem: Optimal Preference Planning to Chronicle (OPP2C)

Input: A transition system \mathcal{T} modeling the environment, an event model \mathcal{M} modeling the occurrences of events in the environment, an LDL_f formula φ specifying the story and hard constraints, and an ordered list of n LDL_f formulas $\psi_1, \psi_2, \dots, \psi_n$ that specify the soft constrains.

Output: The set of correct Pareto optimal policies for minimizing the expected number of steps k to record a story-location ξ_k that satisfies φ while minimizing the expected cost of violation of the soft constraints, defined by f , if a desired story-location can be captured almost-surely, NOPOLICYEXISTS otherwise.

8.2 ALGORITHM DESCRIPTION

This section presents our algorithm.

Converting the LDL_f formulas into DFAs

The first step of the algorithm uses the algorithm of De Giacomo and Vardi [31] to construct for the LDL_f formula φ , a DFA $\mathcal{D} = (Q, 2^{AP'}, \delta, q_0, F)$ such that $\mathcal{L}(\mathcal{D}) = \text{traces}(\varphi)$, and for each of the LDL_f formulas ψ_i for $i \in \{1, 2, \dots, n\}$, a DFA $\mathcal{D}_i = (Q_i, 2^{AP'}, \delta_i, v_{i,0}, F_i)$ such that $\mathcal{L}(\mathcal{D}_i) = \text{traces}(\psi_i)$.

The following sections discuss how to use these DFAs to solve the problem.

Cost filters

From the DFAs of the soft constraints, the algorithm constructs a filter $\mathcal{F} = (V, Y, C, \tau, v_0, c)$ in which $C = \{\text{F}, \text{T}\}^n$ that is used to check which ones of the soft constraints, a story-location is violating (satisfying). Note that for each state v , $c(v)$ is a Boolean vector of size n .

The following result shows how to construct the filter from the DFAs representing the soft constraints.

Definition 32. From the DFAs $\mathcal{D}_i = (Q_i, 2^{AP'}, \delta_i, q_{i,0}, F_i)$ for $i \in \{1, 2, \dots, n\}$, we construct filter $\mathcal{F} = (Q, 2^{AP'}, C, \tau, v_0, c)$ in which

- $V = Q_1 \times Q_2 \times \dots \times Q_n$,
- $q_0 = (q_{1,0}, q_{2,0}, \dots, q_{n,0})$,
- for any state $(q_1, q_2, \dots, q_n) \in V$ and any $A \in 2^{AP'}$,

$$\tau((q_1, q_2, \dots, q_n), A) = (\tau_1(q_1, A), \tau_2(q_2, A), \dots, \tau_n(q_n, A)),$$

and

- $c : V \mapsto \{\text{F}, \text{T}\}^n$ is an output function that assigns to each state $(q_1, q_2, \dots, q_n) \in V$, a Boolean vector $\mathbf{v} \in \{\text{F}, \text{T}\}^n$ such that for each $i \in \{1, 2, \dots, n\}$, $\mathbf{v}[i] = \text{T}$ if $q_i \in F_i$ and $\mathbf{v}[i] = \text{F}$ otherwise.

The output function tells a recorded story-location satisfies which ones of the soft constraints.

The next section uses this filter to make a product automaton.

Goal MDP

Now the algorithm constructs a Goal MDP as follows.

Definition 33. [Goal MDP] For the transition system $\mathcal{T} = (W, R, w_0, AP, L)$, the event model $\mathcal{M} = (S, P, s_0, E, g)$, the DFA $\mathcal{D} = (Q, E, \delta, q_0, F)$, and the filter $\mathcal{F} = (V, Y, C, \tau, v_0, c)$, the associated *Goal MDP* is a tuple $\mathcal{P}_{(\mathcal{T}, \mathcal{D}, \mathcal{M}, \mathcal{F})} = (X, A, x_0, \mathbf{T}, X_G, \mathbf{f})$, in which

1. $X = S \times Q \times W \times V$ is the state space;
2. $A = (E \cup \{\epsilon\}) \times W$ is the action space;
3. $x_0 = (s_0, q_0, w_0, v_0)$ is the initial state;
4. $\nu : X \rightarrow A$ is the *doable-actions* function that for each state $x \in X$, $\nu(x)$ is the set of actions the robot can do at state x , and for each $x = (s, q, w, v) \in X$,

$$\nu(x) = \bigcup_{w':(w,w') \in R} \left(\bigcup_{e \in E} \{(w', e)\} \cup \{(w', \epsilon)\} \right),$$

5. $\mathbf{T} : X \times A \times X \rightarrow [0, 1]$ is the transition probability function such that for each $x = (s, q, w, v), x' = (s', q', w', v') \in X$ and $A = (u, w'') \in \nu(x)$,

$$\mathbf{T}(x, a, x') = \begin{cases} \mathbf{P}(s, s') \cdot g(s', u, w') & \text{if } u \neq \epsilon, q \notin F, w'' = w', (w, w') \in R, \\ & q' = \delta(q, L(w') \cup \{w', u\}), \\ & v' = \tau(v, L(w') \cup \{w', u\}) \quad (5.a) \\ \mathbf{P}(s, s') \cdot (1 - g(s', u, w')) & \text{if } u \neq \epsilon, q \notin F, w'' = w', (w, w') \in R, \\ & q' = \delta(q, L(w') \cup \{w'\}), \\ & v' = \tau(v, L(w') \cup \{w'\}) \quad (5.b) \\ \mathbf{P}(s, s') & \text{if } u = \epsilon, q \notin F, w'' = w', (w, w') \in R, \\ & q' = \delta(q, L(w') \cup \{w'\}), \\ & v' = \tau(v, L(w') \cup \{w'\}) \quad (5.c) \\ 1 & \text{if } q \in F, q' = q, w'' = w', (w, w') \in R \quad (5.d) \\ 0 & \text{otherwise.} \end{cases}$$

6. $X_G = S \times F \times W \times V$ is the set of goal states;

7. $\mathbf{f} : X_G \rightarrow \{\mathbf{F}, \mathbf{T}\}^n$ is a goal-state-weighting function such that for each goal state $x = (s, q, w, v) \in X_G$, $\mathbf{f}(x) = c(v)$.

Each state of this MDP is a tuple (s, q, w, v) and when the MDP enters this state, the event model enters state s , the robot enters location w , the DFA enters state q , and the filter enters state v . For each pair of states $x = (s, q, w, v), x' = (s', q', w', v') \in X$ and action $a = (u, w'') \in \nu(A)$, $\mathbf{T}(x, a, x')$ is the probability that the MDP transitions from x to x' by doing action a . This transition corresponds to a situation where the robot chooses to record event u at location $w'' = w'$, the event model makes a transitions from s to s' , and then based on the result of success or failure of recording event u , the DFA transitions from q to q' , and the filter transitions from v to v' .

If $u \neq \epsilon$ and the robot successfully records u in location $w'' = w'$, then it changes the current state of the DFA to $\delta(q, L(w') \cup \{w', u\})$ and the current state of the filter

to $v' = \tau(v, L(w') \cup \{w', u\})$, otherwise, it changes the current state of the DFA to $\delta(q, L(w') \cup \{w'\})$ and the current state of the filter to $v' = \tau(v, L(w') \cup \{w'\})$. Cases (4.a) and (4.b) above respectively show these two situations. Case (4.c), makes all the goal states of the MDP into absorbing states.

We treat this MDP as a multi-objective MDP, where one objective is the number of steps to reach a goal state and the other objective is the violation cost, determined by function \mathbf{f} , when the MDP reaches the goal states.

After constructing the MDP, one needs to check if a correct policy exists or not. This can be done using the same discussions of Section 5.2 of Chapter 5. Based on those discussions, if the MDP has no unavoidable dead-end, then a correct policy exists. Otherwise, no correct policy exists.

The following section discusses how to solve the MDP if it has a correct policy.

Solving the Goal MDP

Recall that our problem concerning this multi-objective Goal MDP has two objectives; one is minimizing the expected number of steps to reach the goal states and the other is minimizing the expected cost of violation when ending in the goal states.

Given a policy $\pi : X \rightarrow A$, the value function of π is a function $\mathbf{V}^\pi : X \rightarrow \mathbb{R}_{\geq 0} \times [0, 1]^n$ such that for each state $x \in X$, $\mathbf{V}^\pi(x)$ is a two-dimensional vector $(V_1^\pi(x), \mathbf{V}_2^\pi(x))$ where $V_1^\pi(x)$ is the expected number of steps to reach the goal states and \mathbf{V}_2^π is a n -dimensional vector such that for each $i \in \{1, 2, \dots, n\}$, $\mathbf{V}_2^\pi[i]$ is the probability that soft constraints ψ_i is violated by the policy π . To compute the value function of a policy, if $x \in X_G$, then we set $V_1^\pi(x) = 0$, and for each $i \in \{1, 2, \dots, n\}$, $\mathbf{V}_2^\pi(x)[i] = 1$ if $\mathbf{f}(x)[i] = \text{F}$, and otherwise $\mathbf{V}_2^\pi(x)[i] = 0$. For $x \in X \setminus X_G$, we compute $\mathbf{V}^\pi(x)$ via the following recurrence

$$\mathbf{V}^\pi(x) = [1, \mathbf{0}_n] + \sum_{x' \in X} \mathbf{V}^\pi(x') \mathbf{T}(x, \pi(x), x'), \quad (8.4)$$

in which $\mathbf{0}_n$ is a zero vector of size n and $[1, \mathbf{0}_n]$ is also a vector.

Note that each $\mathbf{V}^\pi(x')$ is a vector, and thus, the summation produces a single vector, which when summed with $[1, \mathbf{0}_n]$ produces a single vector.

Let $\mathbf{V}^{\pi_1}(x) = (V_1^{\pi_1}(x), \mathbf{V}_2^{\pi_1}(x))$ and $\mathbf{V}^{\pi_2}(x) = (V_1^{\pi_2}(x), \mathbf{V}_2^{\pi_2}(x))$ be respectively the values of policies π_1 and π_2 at state x . Denoted $\mathbf{V}^{\pi_1}(x) \succ \mathbf{V}^{\pi_2}(x)$, it is said that $\mathbf{V}^{\pi_1}(x)$ *dominates* $\mathbf{V}^{\pi_2}(x)$ if

$$\begin{aligned} & \left(V_1^{\pi_1}(x) < V_1^{\pi_2}(x) \wedge f(\mathbf{V}_2^{\pi_1}(x)) \leq f(\mathbf{V}_2^{\pi_2}(x)) \right) \\ \vee & \left(V_1^{\pi_1}(x) \leq V_1^{\pi_2}(x) \wedge f(\mathbf{V}_2^{\pi_1}(x)) < f(\mathbf{V}_2^{\pi_2}(x)) \right), \end{aligned} \quad (8.5)$$

and otherwise it is said that $\mathbf{V}^{\pi_1}(x)$ does not dominate $\mathbf{V}^{\pi_2}(x)$, denoted $\mathbf{V}^{\pi_1}(x) \not\succeq \mathbf{V}^{\pi_2}(x)$. Intuitively, π dominates π_2 if it improves at least one of the objectives of expected number of steps or the expected cost of violation without losing the quality of any of those two objectives.

The set of all policies that cannot be dominated by any policy at state x is called the set of all Pareto optimal policies at state x , which is denoted Π_x° , and is formally defined,

$$\Pi_x^\circ = \{ \pi_1 \in \Pi \mid \forall \pi_2 \in \Pi : \mathbf{V}^{\pi_2}(x) \not\succeq \mathbf{V}^{\pi_1}(x) \}.$$

The purpose of our problem is to compute $\Pi_{x_0}^\circ$, also denoted Π° , the set of all Pareto optimal policies for the MDP. One can use a variety of methods to compute Π° . For a survey, see [119]. In this work, we use the *convex hull value iteration method* [6].

This method computes a subset of Π° , denoted $CH(\Pi^\circ)$, that contains the set of Pareto optimal policies that are optimal for a *linear scalarization*, which projects the multi-value vector $\mathbf{V}^\pi(x)$ into a scalar value using a weight vector $\mathbf{w} = (w_1, w_2)$ that specifies the relative importance of each of the objectives. All the elements of the weight vector are non-negative real numbers and they sum to 1. Formally, given a weight vector $\mathbf{w} = (w_1, w_2)$ and a value vector \mathbf{V}^π , the linear scalarization of

$\mathbf{V}^\pi = (V_1^\pi, \mathbf{V}_2^\pi)$ under \mathbf{w} is

$$\mathbf{V}_{\mathbf{w}}^\pi = w_1 V_1^\pi + w_2 \mathbf{V}_2^\pi.$$

The $CH(\Pi^o)$ contains all the Pareto optimal policies the linear scalarization of each of which is optimal for at least a weight vector. Formally,

$$CH(\Pi^o) = \{\pi_1 \in \Pi^o \mid \exists \mathbf{w} : \forall \pi_2 \in \Pi^o : \mathbf{V}_{\mathbf{w}}^{\pi_1} \leq \mathbf{V}_{\mathbf{w}}^{\pi_2}\}.$$

Note that computing $CH(\Pi^o)$ is useful when we know that our final decision to pick a Pareto optimal policy imposes a linear weight vector over the importance of the objectives, but we do not know priory which weight vector we will pick. Otherwise, if we initially know what will be our weight vector, then we need to compute only one policy, which is computable using the standard Bellman equation for single MDPs, in which the elements of weight vector has already applied to the terms of the equation as magnifiers.

To compute $CH(\Pi^o)$, the convex hull value iteration method uses the following recurrence

$$Q^o(x, a) = \mathbb{E}_{x'|x, a} \left[[1, \mathbf{0}_n] + \text{hull} \bigcup_{a' \in \nu(x')} Q^o(x', a') \right], \quad (8.6)$$

in which each Q^o is a set (rather than a single value) of non-dominated value vectors and the operation *hull* takes the convex hull of a set of value vectors. To implement this equation one needs to use a translation-and-scaling operation

$$\mathbf{r} + bQ^o = \{\mathbf{r} + b\mathbf{t} \mid \mathbf{t} \in Q^o\},$$

and a summing-convex-hulls operation

$$Q^o + U^o = \text{hull}\{\mathbf{r} + \mathbf{t} \mid \mathbf{r} \in Q^o, \mathbf{t} \in U^o\}.$$

Then, once the equations are solved and a weight vector \mathbf{w} is given, we extract the best Q -values as follows,

$$Q_{\mathbf{w}=(w_1, w_2)}(x, a) = \max_{\mathbf{r}=(r_1, \mathbf{r}_2) \in Q^o(x, a)} w_1 r_1 + w_2 f(\mathbf{r}_2).$$

Note that multi-objective MDPs usually have many Pareto optimal policies, whose number in the worst case is $\mathcal{O}(|X|^{|A|})$. As such it is not feasible to compute all of them. Additionally, when we compute the Q^o 's, there might be many vectors whose values are very close. Mandow et al. [86] propose an approximation approach in which each element of those vectors are rounded upto a limited precision. Using this approach, many of the vectors project to the same vector, reducing the sizes of the convex hulls. This significantly reduces the computation time, because computation time of the algorithm grows exponentially with the sizes of the convex hulls. Using this approach, however, may come at the cost of losing optimality. In our implementations, we round the values to the nearest 0.01.

Moreover, to reduce the number of times the equation is backed up for the states of the MDP, we use the idea of topological value iteration, presented in Section 5.4. Based on this idea, we choose a reverse order of the topological order of the set of strongly connected components of the MDP, and then we pick a strongly connected component (SCC) at a time, compute the Q^o -values for all the states within that SCC, and then proceed to the next SCC.

The next section shows how this algorithm performs on a small case study.

8.3 CASE STUDY

In this section, we present a case study, for which we implemented our algorithm in Python. We performed all the trials on an Ubuntu 16.04 computer with a 3.6 GHz processor.

A student life in University of South Carolina

The University of South Carolina wants to make a documentary introducing student life in their university. To do this, they want a student to volunteer to record her while she is in different buildings and doing different activities. Madelyn has volunteered to be the student in the documentary. The university has a multi-task robot that is capable of capturing videos, and they want the robot to do the task of recording such a documentary while it should also do other tasks as well.

Figure 8.1a shows locations of interest, including Madeline’s office, w_o ; the University’s gym; w_g ; the University’s dining hall, w_d ; and the University’s library, w_l . The transitions of the transition system are based on whether the robot is able to move from one location to another location in 10 minutes or not. For example, because the robot cannot reach the library from the gym in one time step, equivalent to 10 minutes, there is no transition from w_g to w_l .

The student’s activities in which the university is interested to make a documentary from include exercising, e ; working, r ; eating food, f ; and drinking coffee, c . Each activity is an event in our formulation and the occurrences of those events within the environment is modeled in the event-location model in Figure 8.1b. The university wants the documentary to contain Madelyn working, and then later either Madelyn eating food or Madelyn drinking coffee. They also want the documentary to contain Madelyn exercising, after which no activity of Madelyn eating food should appear in the video. This documentary is specified by the LDL_f formula $\langle true^* \rangle (r \wedge \langle true^* \rangle (f|c)) \wedge \langle true^* \rangle e \wedge [true^*](e \rightarrow \neg \langle true^* \rangle f)$. There are three soft constraints, ordered from most to least important:

1. After going to the dining hall, the robot should not go to office. This constraint is specified by:

$$[true^*](d \rightarrow \neg \langle true^* \rangle o)$$

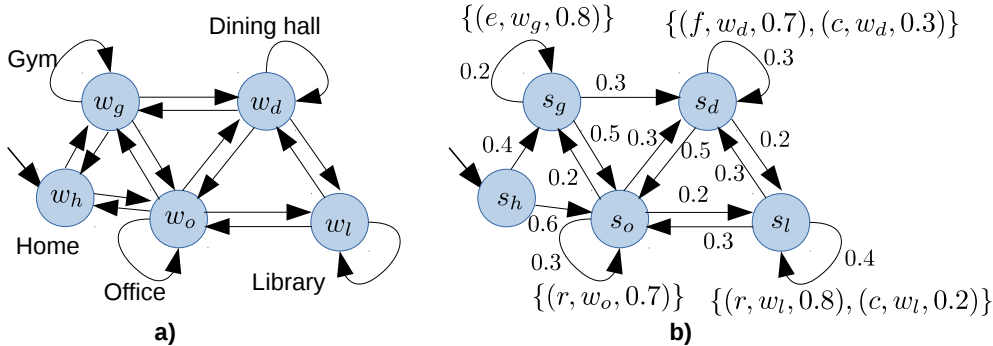


Figure 8.1: **a)** A transition system showing the locations of interest from the University of South Carolina. The transitions of this structures is based on the motion capabilities of the robot, that is, a transition from a location w_1 to location w_2 in this transition system exists only if the robot can move from w_1 to w_2 in 10 minutes. **b)** The event-location model that represents the occurrences of the events of interest in this environment. Those events include exercising, e ; working, r ; eating food, f ; and drinking coffee c .

2. If the robot enters the library, then in the next time step it should go to the office. This constraint is specified by:

$$[true^*](l \rightarrow \langle true \rangle o)$$

3. The robot should not stay two consecutive time steps in the dining hall. This constraint is specified by:

$$\neg \langle true^* \rangle (d \rightarrow \langle true \rangle d)$$

Note that if the robot wants to reduce the number of steps to record a desired sorty, then these soft constraints are conflicting because the robot enters dining hall and the current time step is not the last step of its execution, then at least one of these three constraints has to be violated.

We now present our results. We computed the Q^o values using the algorithm of the previous section with at most 120 iterations for states within a strongly connected component, and once those values are computed, we extracted 11 different policies

using 11 different weight vectors $\mathbf{w}_1 = [1.0, 0]$, $\mathbf{w}_2 = [0.9, 0.1]$, \dots , $\mathbf{w}_{11} = [0, 1.0]$, for each of which we computed its value function using Equation 8.4. For each policy, we generated 1000 simulations where in each simulation, Madelyn took a random path in the environment and did some of the activities drawn by the probabilities in the event-location model, and we let the robot to use the computed policy to record a story. For each of those 11 experiments, we computed the average number of steps to record a desirable story-location and the average number of times each of the soft constraints was violated.

Table 8.1 shows results of our experiment. Based on this table, when the importance of the expected number of steps increases over the importance of expected cost of violation (from the bottom of table to the top of it), the expected number of steps to record a desired story-location reduces. In contrast, when the importance of expected cost of violation over the importance of expected number of steps increases (from the top part of the table to the bottom of it), the expected cost of violation of soft constraints reduces.

We also compute a policy on the MDP for only minimizing the expected number of steps and based on this policy the expected number of step under the optimal policy was 27.67, which matches the first row of the table. We also compute a policy on the MDP but this time for only minimizing the expect cost of violation of the soft constrains, and based on that, the expected cost of violation of the soft constraints under an optimal policy was $[0.00, 0.00, 0.00]$, which matches the last row of the table. As for computation time, it took 7460 seconds for our algorithm to compute the polices.

Weight Vec.	Expc. #Steps	Expc. Violation	Avg. #Steps	Avg. Violation
[1, 0]	27.67	[0.89, 0.00, 0.00]	27.61	[0.90, 0, 0]
[0.9, 0.1]	28.13	[0.74, 0.18, 0.00]	28.41	[0.73, 0.19, 0]
[0.8, 0.2]	41.29	[0.00, 0.79, 0.00]	40.04	[0, 0.76, 0]
[0.7, 0.3]	41.30	[0.00, 0.79, 0.00]	41.08	[0, 0.77, 0]
[0.6, 0.4]	41.30	[0.00, 0.79, 0.00]	40.97	[0, 0.79, 0]
[0.5, 0.5]	41.30	[0.00, 0.79, 0.00]	42.45	[0, 0.80, 0]
[0.4, 0.6]	41.30	[0.00, 0.79, 0.00]	40.00	[0, 0.77, 0]
[0.3, 0.7]	41.47	[0.00, 0.79, 0.00]	41.74	[0, 0.79, 0]
[0.2, 0.8]	46.20	[0.00, 0.79, 0.00]	44.64	[0, 0.78, 0]
[0.1, 0.9]	62.79	[0.00, 0.56, 0.00]	62.00	[0, 0.57, 0]
[0, 1]	130.78	[0.00, 0.00, 0.00]	128.21	[0, 0, 0]

Table 8.1: Results of our experiment for computing the set of Pareto optimal policies for the case study of a student’s life in the University of South Carolina. We first computed the Q^o values using our version of the convex hull value iteration and then we extracted 11 optimal policies for 11 different weight vectors from those computed Q^o values. The first element of the vectors indicates the importance of number of steps and the second element indicates the importance of the cost of violation of soft constraints. The second column is the theoretical prediction for the number of steps to record a story-location and the third column is the theoretical prediction of cost of violation of the soft constraints. The fourth column represents the average number of steps to record a desired story-location using the computed policy for 1000 simulations, and the last column shows the average cost of violation over those 1000 simulations.

CHAPTER 9

SENSOR SELECTION FOR DETECTING DEVIATIONS FROM A PLANNED ITINERARY

In this chapter, we consider the problem of choosing a minimum number of sensors to turn on so that using the sequence of observations produced by those sensors the system could tell whether an agent deviated from its claimed itinerary or not.

The material in this chapter is based on our work Rahmani, Shell, and O’Kane [112], which appeared in IROS 2021.

The organization of this chapter is as follows. In Section 9.1, we formalize the problem of optimal sensor selection to detect deviations from a disclosed itinerary; in Section 9.3, we establish the computational hardness of the problem; in Section 9.4, we show how it may be treated via integer linear programming; and in Section 6.4, we present experimental results.

9.1 DEFINITIONS AND PROBLEM STATEMENT

This section formalizes our sensor selection problem.

Modeling the environment

In our approach, the environment is modeled using a discrete structure, called a world graph, defined as follows.

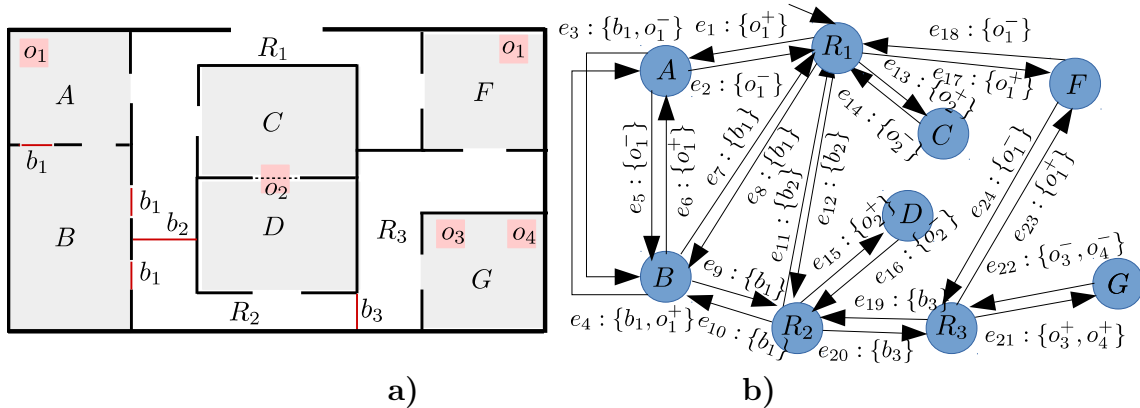


Figure 9.1: **a)** An example of an environment, which shows the floor map of a department. This environment is guarded by beam sensors b_1 , b_2 , and b_3 , and by occupancy sensors o_1 , o_2 , o_3 , and o_4 . **b)** A multigraph representing the environment. Each edge e is labeled by a world-observation, a set of events received when an agent moves from the region represented by the source vertex of e to the region represented by the target vertex of e .

Definition 34. [World graph] A *world graph* is an edge-labeled directed multigraph $\mathcal{G} = (V, E, \text{src}, \text{tgt}, v_0, S, \mathbb{Y}, \lambda)$ in which

- V is a nonempty vertex set,
- E is a set of edges,
- $\text{src} : E \rightarrow V$ and $\text{tgt} : E \rightarrow V$ are *source* and *target* functions, respectively, which identify the source vertex and target vertex of each edge,
- $v_0 \in V$ is an initial vertex,
- $S = \{s_1, s_2, \dots, s_k\}$ is a nonempty finite set of sensors,
- $\mathbb{Y} = \{Y_{s_1}, Y_{s_2}, \dots, Y_{s_k}\}$ is a collection of mutually disjoint event sets associated to each sensor, and
- $\lambda : E \rightarrow \mathcal{P}(Y_{s_1} \cup Y_{s_2} \cup \dots \cup Y_{s_k})$ is a labeling function, which assigns to each edge, a *world-observation*—a set of events.

(Here $\mathcal{P}(X)$ denotes the set of all subsets of X .)

The intuition is that a world graph describes an environment through which agent might move, along with the available sensors in the environment and also the sensor events that agent may trigger along its motion, provided those sensors are active. Vertices in the graph represent regions within the environment of interest. Edges represent feasible transitions between regions, each labeled with a set of sensor events that happen simultaneously when the system makes the transition corresponding to that edge.

In this model, for each sensor $s_i \in S$, Y_{s_i} is the set of all events produced by s_i . Notice that Definition 34 stipulates that distinct sensors produce disjoint events, that is, for each $s_i, s_j \in S$, if $s_i \neq s_j$, then $Y_{s_i} \cap Y_{s_j} = \emptyset$. The labeling function λ is used to indicate for each edge, the set of events produced by the sensors when the agent makes a transition corresponding to that edge in the environment.

Example: Beam and occupancy sensors

Though the technical results to follow apply for any world graph that satisfies Definition 34, to keep the description reasonably concrete, we present examples focused upon occupancy sensors (which detect the presence of the agent in a region) and beam sensors (which detect the passage of an agent between adjacent regions).

To illustrate, consider the simple environment in Fig. 9.1a, which is guarded by four occupancy sensors o_1, o_2, o_3 , and o_4 and three beam sensors b_1, b_2 , and b_3 . Fig. 9.1b shows the world graph corresponding to this environment, as constructed via the algorithm of Yu and LaValle [167]. Each state of this graph represents a room or a region within the environment guarded by the same set of sensors. Each edge shows a transition between two neighboring regions.

Notice that, in the example, some pairs of rooms are connected by multiple doors, each of which are guarded by different sensors. This explains why Definition 34 uses a multigraph structure for world graphs. Also note that in this environment, an agent

cannot directly move between rooms C and D because those two rooms are separated by a window rather than a door.

In this example, an occupancy sensor o , which detects the presence of an agent in a region X , is activated when the agent enters X and is deactivated once the agent exits X . Accordingly, each occupancy sensor o in this example produces two events, o^+ , which occurs when o is activated, and o^- , which happens when o is deactivated. A beam sensor is activated when a mobile agent physically crosses (or breaks) the beam and then instantly deactivated. Because the agent is mobile and a beam sensor is deactivated immediately after it was activated, we model each beam sensor b with a single event b in Y_b . When beam b is broken, the system knows that the physical line segment was crossed, but not the direction of crossing. For the environment in Fig. 9.1: for each of the occupancy sensors o_i , $i \in \{1, 2, 3, 4\}$, $Y_{o_i} = \{o_i^+, o_i^-\}$; for each of the beam sensors b_j , $j \in \{1, 2, 3\}$, $Y_{b_j} = \{b_j\}$.

When an agent makes a transition between two regions, it is possible that several events happen simultaneously, and in fact, the system observes all those events at the same time. For example, when an agent makes a transition from room A to room B from the left door, two events o_1^- and b_1 happen simultaneously, and thus, edge e_3 in the world graph is labeled with the world-observation $\{b_1, o_1^-\}$.

Finally, notice that a world graph can readily represent scenarios in which a single sensor guards multiple transitions. In the example, B has four doors, three of which are guarded. The beams that guard those doors are assumed to be a single beam sensor $b_1 \in S$. When an agent crosses any of those beams, the system knows that one of them is crossed but it does not know which one it was. Likewise, rooms C and D are guarded by a single occupancy sensor o_2 , which is located on the window between those two rooms. Thus, if an agent enters any of those two rooms, o_2 is activated, but by observing o_2^+ , the system cannot tell if the agent entered room C or room D .

Itinerary DFA

In our story validation problem, an agent takes a tour in the environment along a continuous path. This path is represented over the world graph \mathcal{G} by a *walk*, which is defined as a finite sequence of edges $e_1e_2\cdots e_n \in E^*$ in which $\text{src}(e_1) = v_0$ and for each $i \in \{1, 2, \dots, n-1\}$, $\text{tgt}(e_i) = \text{src}(e_{i+1})$. The set of all walks over \mathcal{G} —that is, the set of all not-necessarily-simple paths one can take in the environment—is denoted $\text{Walks}(\mathcal{G})$.

The agent claims that its tour will be one of those words specified by a deterministic finite automaton (DFA):

Definition 35. [Itinerary DFA] An *itinerary DFA* over a world graph $\mathcal{G} = (V, E, \text{src}, \text{tgt}, v_0, S, \mathbb{Y}, \lambda)$ is a DFA $\mathcal{I} = (Q, E, \delta, q_0, F)$ in which Q is a finite set of states; E is the alphabet; $\delta : Q \times E \rightarrow Q$ is the transition function; q_0 is the initial state; and F is the set of all accepting (final) states.

For each finite word $r = e_1e_2\cdots e_n \in E^*$, there is a unique sequence of states $q_0q_1\cdots q_n$ for which q_0 is the initial state and for each $i \in \{0, 1, \dots, n-1\}$, $\delta(q_i, e_i) = q_{i+1}$. Word r is *accepted* by the DFA if $q_n \in F$. The language of \mathcal{I} , denoted $L(\mathcal{I})$, is the set of all finite words accepted by \mathcal{I} , i.e., $L(\mathcal{I}) = \{r \in E^* \mid r \text{ is accepted by } \mathcal{I}\}$. The robot claims its tour will be one of the words $r \in L(\mathcal{I})$. Note that each word accepted by this DFA is a walk over the world graph, and accordingly, the robot’s itinerary not only specifies the sequence of locations the agent visits but it also identifies the specific transitions (i.e. the doors between the rooms) through which the agent moves.

Itinerary validation

We seek to enable a minimal subset $M \subseteq S$ of sensors such that, when the agent finishes its tour within the environment, the system can determine with full certainty whether the agent followed its itinerary or not. At the completion of the agent’s tour,

the system does not know the exact tour the agent took in the environment, instead receiving only a sequence of world-observations. Each item in the sequence is generated by the system when a set of sensors were activated or deactivated simultaneously as a result of the agent's moving in the environment.

Let us mildly abuse notation and use Y_M to denote the set of all events produced by sensors in some $M \subseteq S$, i.e., $Y_M = \bigcup_{s \in M} Y_s$. If, from all sensors S , only the sensors in M are turned on, then when the agent transitions across e in \mathcal{G} , the system receives world-observation $\lambda(e) \cap Y_M$. That is, when transitioning across an edge, the system observes precisely those events that are both associated with that edge and enabled by one of set M 's selected sensors. Where $\lambda(e) \cap Y_M = \emptyset$, this must be handled slightly differently: in this case, the system produces no symbol at all (not a symbol reporting that some un-sensed event occurred—which would itself be a tacit sort of information). We make this precise next.

The agent's walk over the world graph generates a sequence of non-empty world-observations. For a world graph $\mathcal{G} = (V, E, \text{src}, \text{tgt}, v_0, S, \mathbb{Y}, \lambda)$, we define function $\beta_{\mathcal{G}} : \text{Walks}(\mathcal{G}) \times \mathcal{P}(S) \rightarrow (\mathcal{P}(Y_S) \setminus \emptyset)^*$ in which for each $r \in \text{Walks}(\mathcal{G})$ and subset of sensors $M \subseteq S$, $\beta_{\mathcal{G}}(r, M)$ gives the sequence of world-observations the system receives when the agent takes walk r and precisely the sensors in M are turned on. Formally, for each $r = e_1 e_2 \cdots e_n \in \text{Walks}(\mathcal{G})$, $\beta_{\mathcal{G}}(r, M) = z_1 z_2 \cdots z_n$ in which for each $i \in \{1, \dots, n\}$, $z_i = \lambda(e_i) \cap Y_M$ if $(\lambda(e_i) \cap Y_M) \neq \emptyset$, and $z_i = \epsilon$ otherwise, where ϵ is the (standard) *empty symbol*.

Based on this function, we make a definition that formulates conditions under which a set of sensors are able to tell whether the agent adhered to its claimed itinerary or not.

Definition 36. [Certifying Sensor Selection] Let $M \subseteq S$ be a subset of sensors. We say M *certifies* itinerary \mathcal{I} on world graph \mathcal{G} if there exist no $r \in L(\mathcal{I}) \cap \text{Walks}(\mathcal{G})$ and $t \in \text{Walks}(\mathcal{G}) \setminus L(\mathcal{I})$ such that $\beta_{\mathcal{G}}(r, M) = \beta_{\mathcal{G}}(t, M)$.

Intuitively, if M is a certifying sensor selection for \mathcal{I} , then based on the sequence of world-observations $\beta_{\mathcal{G}}(r, M)$ the system perceives from the environment, the system can tell whether r was within the claimed itinerary or not, that is, whether $r \in L(\mathcal{I})$ or not. In fact, if for each $r \in L(\mathcal{I}) \cap \text{Walks}(\mathcal{G})$, there is no $t \in (\text{Walks}(\mathcal{G}) \setminus L(\mathcal{I}))$ such that $\beta_{\mathcal{G}}(t, M) = \beta_{\mathcal{G}}(r, M)$, then the system can tell with full certainty if r was within the claimed itinerary or not. Thus, the system must choose a sensor set M to turn on that is certifying for \mathcal{I} on \mathcal{G} .

We formalize our minimization problem as follows.

Problem: Minimal sensor selection to validate an itinerary (MSSVI)

Input: A world graph $\mathcal{G} = (V, E, \text{src}, \text{tgt}, v_0, S, \mathbb{Y}, \lambda)$ and an itinerary DFA $\mathcal{I} = (Q, V, \delta, q, F)$.

Output: A minimum size certifying sensor selection $M \subseteq S$ for \mathcal{I} on \mathcal{G} , or ‘INFEASIBLE’ if no such certifying sensor selection exists.

Before showing this problem to be NP-hard, we describe a construction that turns out to be useful in what follows.

9.2 WORLD GRAPH-ITINERARY PRODUCT AUTOMATA

In this section, we describe how to use the inputs of the MSSVI problem to construct a product automaton that captures the interactions between a world graph and an itinerary DFA. We use this construction for both a hardness result about MSSVI (in Section 9.3) and a practical solution of MSSVI via integer linear programming (in Section 9.4). This product automaton is defined as follows.

Definition 37. [Product automaton] Let $\mathcal{G} = (V, E, \text{src}, \text{tgt}, v_0, S, \mathbb{Y}, \lambda)$ be a world graph and $\mathcal{I} = (Q, E, \delta, q_0, F)$ be an itinerary DFA. The product automaton $\mathcal{P}_{\mathcal{G}, \mathcal{I}}$ is a partial DFA $\mathcal{P}_{\mathcal{G}, \mathcal{I}} = (Q_{\mathcal{P}}, E, \delta_{\mathcal{P}}, q_0^{\mathcal{P}}, F_{\mathcal{P}})$ with

- $Q_{\mathcal{P}} = Q \times V$,

- $\delta_{\mathcal{P}} : Q_{\mathcal{P}} \times E \rightharpoonup Q_{\mathcal{P}}$ is a partial function such that for each $(q, v) \in Q_{\mathcal{P}}$ and $e \in E$, $\delta_{\mathcal{P}}((q, v), e)$ is undefined if $\text{src}(e) \neq v$, otherwise, $\delta_{\mathcal{P}}((q, v), e) = (\delta(q, e), \text{tgt}(e))$,
- $q_0^{\mathcal{P}} = (q_0, v_0)$, and
- $F_{\mathcal{P}} = F \times V$.

Note that the transition function of this DFA is partial. We will write $\delta_{\mathcal{P}}(p, e) = \perp$ to mean that $\delta_{\mathcal{P}}$ is undefined for (p, e) . The extended transition function $\delta_{\mathcal{P}}^* : Q_{\mathcal{P}} \times E^* \rightharpoonup Q_{\mathcal{P}}$ —which for each $q \in Q_{\mathcal{P}}$ and $r \in E^*$, $\delta_{\mathcal{P}}^*(q, r)$ denotes the state to which the DFA reaches by tracing r from state q —is also partial. For a word $r = e_1 e_2 \cdots e_n \in E^*$, we use $\delta_{\mathcal{P}}^*(q_0^{\mathcal{P}}, r) = \perp$ to mean that this DFA crashes when it traces r from the initial state, that is, there is a unique state sequence $q_0 q_1 \cdots q_k$ for some $k < n$ such that $\delta_{\mathcal{P}}(q_{i-1}, e_i) = q_i$ for all $i \in \{1, 2, \dots, k-1\}$ but $\delta_{\mathcal{P}}(q_k, e_k) = \perp$. A word $r \in E^*$ is *trackable* by this DFA if $\delta_{\mathcal{P}}^*(q_0^{\mathcal{P}}, r) \neq \perp$.

Our purpose in constructing this product automaton is revealed by the following result.

Lemma 22. Let \mathcal{G} , \mathcal{I} , and $\mathcal{P}_{\mathcal{G}, \mathcal{I}}$ be the structures in Definition 37. A subset $M \subseteq S$ of sensors is a certifying sensor selection for \mathcal{I} if and only if for each $r, r' \in E^*$ such that $\delta_{\mathcal{P}}^*(q_0^{\mathcal{P}}, r) \in F_{\mathcal{P}}$ and $\delta_{\mathcal{P}}^*(q_0^{\mathcal{P}}, r') \in Q_{\mathcal{P}} \setminus F_{\mathcal{P}}$, it holds that $\beta_{\mathcal{G}}(r, M) \neq \beta_{\mathcal{G}}(r', M)$.

Proof. The construction yields two direct observations:

- (1) Every word trackable by \mathcal{P} is a walk over \mathcal{G} and vice versa, i.e., $\{r \in E^* \mid \delta_{\mathcal{P}}^*(q_0^{\mathcal{P}}, r) \neq \perp\} = \text{Walks}(\mathcal{G})$.
- (2) The accepting states of \mathcal{P} correspond to the accepting states of the itinerary DFA, so $L(\mathcal{P}_{\mathcal{G}, \mathcal{I}}) = L(\mathcal{I})$.

Taken together, (1) and (2) imply that $\text{Walks}(\mathcal{G}) \setminus L(\mathcal{I}) = \{r \in E^* \mid \delta_{\mathcal{P}}^*(q_0^{\mathcal{P}}, r) \in Q_{\mathcal{P}} \setminus F_{\mathcal{P}}\}$. This means that each walk over the world graph that is not within the language

of the itinerary DFA, reaches a non-accepting state in this DFA. But, because every word in the the language of the itinerary DFA reaches to an accepting state in this DFA, if there are two words r and r' such that r reaches to an accepting state, r' reaches to a non-accepting state, and r and r' both yield the same sequence of world-observations by $\beta_{\mathcal{G}}$ under M , i.e., $\beta_{\mathcal{G}}(r, M) = \beta_{\mathcal{G}}(r', M)$, then M is not a certifying sensor selection for \mathcal{I} . Contrariwise, if there are no such words r and r' , then M is a certifying sensor selection. This completes the proof. \square

As a result, given a sensor selection M as a feasible solution to MSSVI with inputs \mathcal{G} and \mathcal{I} , one can use the product automaton $\mathcal{P}_{\mathcal{G}, \mathcal{I}}$ to check if M is a certifying sensor selection for \mathcal{I} or not by testing whether such r and r' described in the proof of this lemma can be found or not. The next section makes this idea clear.

9.3 HARDNESS OF MSSVI

Next, we present a hardness result for minimal sensor selection, starting by casting MSSVI as a decision problem.

Decision Problem: Minimal sensor selection to validate an itinerary (MSSVI-DEC)

Input: A world graph $\mathcal{G} = (V, E, \text{src}, \text{tgt}, v_0, S, \mathbb{Y}, \lambda)$, an itinerary DFA $\mathcal{I} = (Q, V, \delta, q_0, F)$, and integer k .

Output: Yes if there is a certifying sensor selection $M \subseteq S$ such that $|M| \leq k$; No otherwise.

We prove that MSSVI-DEC is NP-complete, by showing that it is both in NP and NP-hard. First, we show that MSSVI-DEC can be verified in polynomial time.

Lemma 23. MSSVI-DEC \in NP.

Proof. We need to show that, using a given sensor selection $M \subseteq S$ as a certificate, we can verify in polynomial time both (1) whether $|M| \leq k$ and (2) whether M is a certifying sensor selection in the sense of Definition 36 or not. As (1) is trivially verifiable, we turn to (2).

Recall that by Lemma 22, if for any words $r, r' \in E^*$ for which $\delta_{\mathcal{P}}^*(v_0, r) \in F_{\mathcal{P}}$ and $\delta_{\mathcal{P}}^*(v_0, r') \in Q_{\mathcal{P}} \setminus F_{\mathcal{P}}$, it holds that $\beta_{\mathcal{G}}(r, M) \neq \beta_{\mathcal{G}}(r', M)$, then M is a certifying sensor selection for the given itinerary DFA \mathcal{I} , otherwise M is not a certifying sensor selection for \mathcal{I} . Thus, to check if M is certifying, we compute, using a fixed point algorithm described presently, a relation R that relates all pairs of states q and p that are reachable from the initial state by two words (walks) that yield the same sequence of world-observations by $\beta_{\mathcal{G}}$ under M . Then we check whether R relates any pairs of states q and p such that one of them is accepting while the other is non-accepting. If R relates such a pair, then M is not a certifying sensor selection for \mathcal{I} . Otherwise, M is certifying for \mathcal{I} . To construct R , begin with R initially assigned to $\{(q_0^{\mathcal{P}}, q_0^{\mathcal{P}})\}$. Then iteratively update R according to the following equation until the iteration reaches a fixed point, with no additional tuples added to R .

$$R \leftarrow R \cup \bigcup_{(q,p) \in R} \left(\bigcup_{\substack{e,d \in E: \\ (\lambda(e) \cap Y_M) = \\ (\lambda(d) \cap Y_M) \\ \text{and } \delta_{\mathcal{P}}(q,e) \neq \perp \\ \text{and } \delta_{\mathcal{P}}(p,d) \neq \perp}} (\delta_{\mathcal{P}}(q, e), \delta_{\mathcal{P}}(p, d)) \right) \\ \cup \bigcup_{(q,p) \in R} \left(\bigcup_{\substack{e \in E: \\ (\lambda(e) \cap Y_M) = \epsilon \\ \text{and } \delta_{\mathcal{P}}(q,e) \neq \perp}} (\delta_{\mathcal{P}}(q, e), p) \right).$$

To expand R , this equation uses two rules, one in the first line and the other in the second line. Fig. 9.2 illustrates those two rules. The first rule states that if $(q, p) \in R$, then for any edges $e, d \in E$ such that q has an outgoing transition for e and p has an outgoing transition for d , if e and d yield the same world-observation under M , then we must add $(\delta_{\mathcal{P}}(q), \delta_{\mathcal{P}}(p))$ to R as well, which means that states $\delta_{\mathcal{P}}(q, e)$ and $\delta_{\mathcal{P}}(p, d)$ are reachable from the initial state of $\mathcal{P}_{\mathcal{G}, \mathcal{I}}$ by two words (walks) that yield the same world observation by $\beta_{\mathcal{G}}$ under M . The second rule enforces that

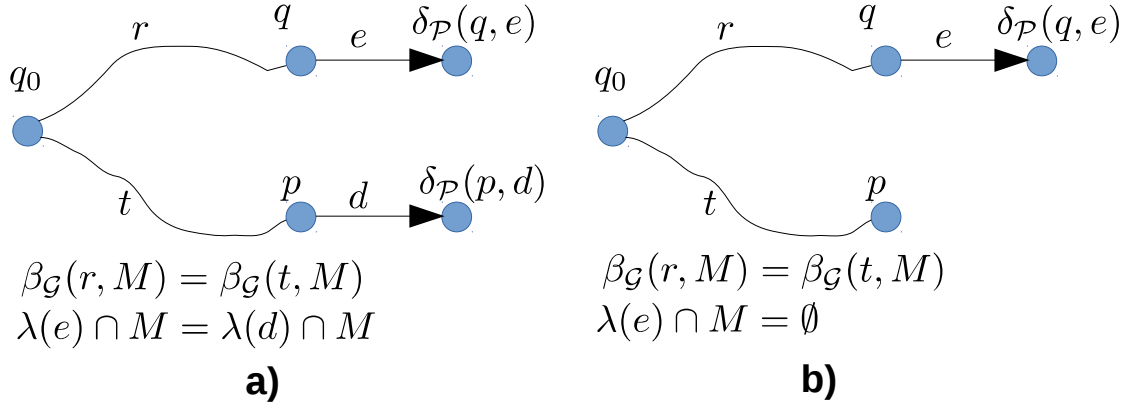


Figure 9.2: Two cases in the construction of the relation R in the proof of Lemma 23. In each case, R is expanded, given that states q and p are already related by R . **a)** In this case, pair $\delta_{\mathcal{P}}(q, e)$ and $\delta_{\mathcal{P}}(p, d)$ are added to R . **b)** In this case, pair $\delta_{\mathcal{P}}(q, e)$ and p are added to R .

if $(q, p) \in R$, then for any edge $e \in E$ such that q has an outgoing transition for e , if e yields the empty world-observation under M (that is, if e yields ϵ by $\beta_{\mathcal{G}}$ under M), then $(\delta_{\mathcal{P}}(q, e), p)$ must be added to R too. This is because here both states $\delta_{\mathcal{P}}(q, e)$ and p are reachable from the initial state, respectively, by a pair of words (walks) r and r' that, respectively, reached q and p , while yielding a single sequence of world-observations. Using an appropriate implementation, this algorithm takes time which is polynomial in the size of \mathcal{P} because there are $\mathcal{O}(|Q_{\mathcal{P}}|^2)$ pairs in R and thus $\mathcal{O}(|Q_{\mathcal{P}}|^2)$ stages of updates, each checking at most $|E|$ edges. This shows that $\text{MSSVI-DEC} \in \text{NP}$. □

The practical import of this lemma is that, in polynomial time, we can decide whether a sensor set is certifying for a given itinerary or not.

Next, we prove that MSSVI-DEC is computationally hard. To do so, we shall reduce from a well-known problem.

Decision Problem: Set cover (SETCOVER-DEC)

Input: A finite set U , called the universe, a collection of subsets $\mathbf{O} = \{O_1, O_2, \dots, O_m\}$ in which, for each $i \in \{1, 2, \dots, m\}$, $O_i \subseteq U$ and $\bigcup_{i \in \{1, 2, \dots, m\}} O_i = U$, and an integer k .

Output: Yes if there is a sub-collection $\mathbf{N} \subseteq \mathbf{O}$ such that $\bigcup_{O \in \mathbf{N}} O = U$ and $|\mathbf{N}| \leq k$, and No otherwise.

This problem is known to be NP-complete [65]. We reduce from SETCOVER-DEC to prove the following result.

Theorem 15. MSSVI-DEC is NP-hard.

Proof. We prove this result by a polynomial reduction from SETCOVER-DEC to MSSVI-DEC. Given a SETCOVER-DEC instance

$$x = \langle U = \{u_1, u_2, \dots, u_n\}, \mathbf{O} = \{O_1, O_2, \dots, O_m\}, k \rangle,$$

construct an MSSVI-DEC instance

$$f(x) = \langle \mathcal{G} = (V, E, \text{src}, \text{tgt}, v_0, S, \mathbb{Y}, \lambda), \mathcal{I}, k' \rangle,$$

as illustrated in Fig. 9.3 and detailed below.

- For the vertices of the world graph, create $2n+2$ states, denoted $V = \{C_0, \dots, C_{n+1}\} \cup \{u_1, \dots, u_n\}$. Note that the u_i elements correspond directly to the elements of the universe in the SETCOVER-DEC instance. The idea is that the u_i 's represent rooms arranged in sequence, each accessible from a shared corridor composed of the C_i 's.
- For the edges of the world graph, create $4n+2$ edges, denoted $E = \{e_0, e_1, \dots, e_n\} \cup \{e'_0, e'_1, \dots, e'_n\} \cup \{d_1, d_2, \dots, d_n\} \cup \{d'_1, d'_2, \dots, d'_n\}$. The src and tgt functions are defined so that each e_i connects C_i to C_{i+1} , each e'_i connects C_{i+1} to C_i , each d_i connects C_i to u_i , and each d'_i connects u_i to C_i .

- Create a set of $n + m + 1$ sensors, $S = \{b_1, b_2, \dots, b_{n+1}\} \cup \{O_1, O_2, \dots, O_m\}$, in which the b_i 's are beam sensors and the O_i 's are occupancy sensors. The event set corresponds to these sensors in the usual way, with one event for each beam sensor and two events for each occupancy sensor, so for each $j \in \{1, 2, \dots, n + 1\}$, $Y_{b_j} = \{b_j\}$, and for each $i \in \{1, 2, \dots, m\}$, $Y_{O_i} = \{O_i^+, O_i^-\}$, and then, $\mathbb{Y} = \{Y_{b_0}, Y_{b_1}, \dots, Y_{b_{n+1}}, Y_{O_1}, Y_{O_2}, \dots, Y_{O_m}\}$.
- For the events labeling each edge, define $\lambda(e_i) = \lambda(e'_i) = b_{i+1}$ for the e_i and e'_i edges, $\lambda(d_i) = \{O_j^+ \mid u_i \in O_j\}$ for the d_i edges, $\lambda(d'_i) = \{O_j^- \mid u_i \in O_j\}$ for the d'_i edges. This models one or more occupancy sensors in each of the u_i rooms, according to the subsets within the SETCOVER-DEC instance, and beam sensors along the corridor between each room.
- For the itinerary DFA \mathcal{I} , construct a DFA accepting the singleton language $L(\mathcal{I}) = \{e_0 d_1 d'_1 e_1 d_2 d'_2 e_2 \dots d_n d'_n e_n\}$ as a linear chain of states.
- For the bound on the number of sensors allowed, choose $k' = k + n + 1$.

In the MSSVI-DEC instance constructed in this way, notice that, for each subset $O \in \mathbf{O}$, the construction makes a corresponding occupancy sensor O and puts that sensor in all rooms u for which $u \in O$. Moreover, the itinerary specifies a single walk $e_0 d_1 d'_1 e_1 d_2 d'_2 e_2 \dots d_n d'_n e_n$, which indicates that the sequence of regions the agent intends to visit is $C_0 C_1 u_1 C_1 C_2 u_2 C_2 \dots C_{n+1}$. That is, the itinerary calls for the agent to travel down the corridor, visiting each room exactly once in the specific order u_1, u_2, \dots, u_n , without backtracking within the corridor. For the system to be able to tell that the agent has visited a room, at least one occupancy sensor in each room must be turned on. Also, each of the beam sensors must be turned on so that the system can assure that the agent did not vacillate back-and-forth between cells in the corridor.

The construction clearly takes polynomial time, so it remains only to show that

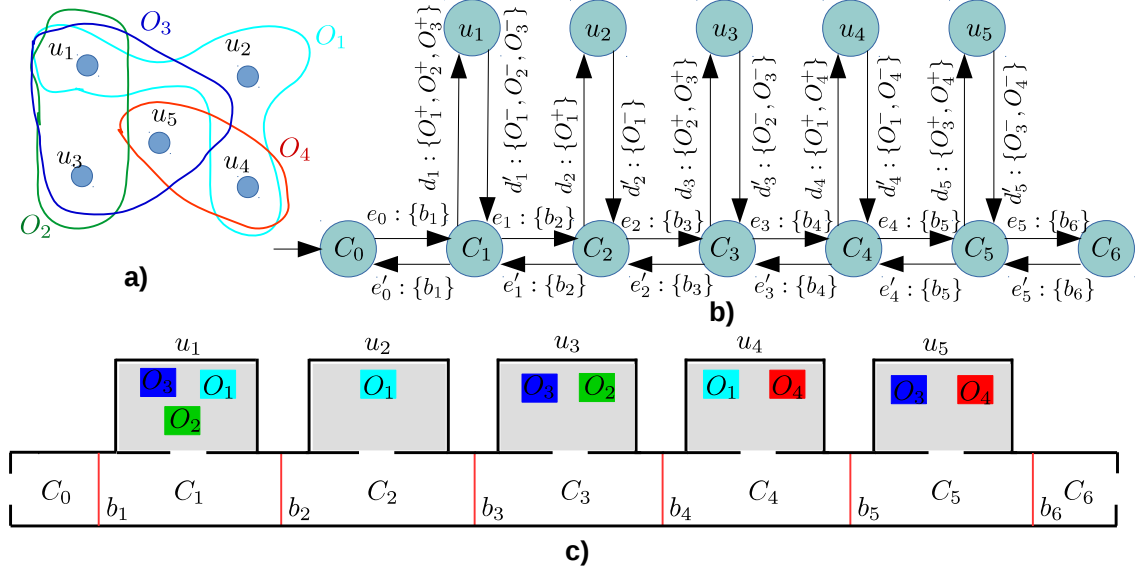


Figure 9.3: **a)** An instance of the set cover problem. **b)** The instance of the MSSVI problem for this set cover instance. **c)** A physical environment representing the MSSVI instance.

the reduction is correct, i.e. that the original SETCOVER-DEC instance has a set cover of size at most k if and only if the constructed MSSVI-DEC has a valid sensor set of size k' .

(\Rightarrow) Suppose there exists a set cover $\mathbf{N} \subseteq \mathbf{O}$ such that $|\mathbf{N}| \leq k$ and $\sum_{O \in \mathbf{N}} O = U$. In this case, based on our discussion, the sensor selection $M = \mathbf{N} \cup \{b_1, b_2, \dots, b_{n+1}\}$ is for itinerary \mathcal{I} , a certifying sensor selection of size $|M| = |\mathbf{N}| + n + 1 \leq k + n + 1 = k'$.

(\Leftarrow) Conversely, suppose there exists for \mathcal{I} , a certifying sensor selection $M \subseteq S$ for which $|M| \leq k'$. As argued above, because M is certifying, it must contain each of the $n + 1$ beam sensors. Thus, there are at most $k' - (n + 1) = k$ occupancy sensors in M . Recall, however, that for this construction, every certifying sensor selection includes at least one occupancy sensor within each room. Thus, the occupancy sensors in M form a set cover of size at most k for the original SETCOVER-DEC instance. \square

Finally, the following results follow immediately from Theorem 15 and Lemma 23.

Theorem 16. MSSVI-DEC is NP-complete.

Corollary 3. MSSVI is NP-hard.

As a result, assuming $P \neq NP$, one cannot find a certifying sensor selection with minimum size in polynomial time.

9.4 MSSVI VIA INTEGER LINEAR PROGRAMMING

In this section, we present an exact solution to MSSVI using an Integer Linear Programming formulation of the problem. First, we cast the MSSVI problem into mathematical programming form, and then linearize its constraints.

Mathematical programming formulation of MSSVI

For each sensor $s \in S$, we introduce a binary variable u_s , which receives value 1 if sensor s is chosen to be turned on, and receives 0 otherwise. For each tuple of states $(q, p) \in Q_{\mathcal{P}} \times Q_{\mathcal{P}}$, we introduce a binary variable $a_{q,p}$, which receives value 1 if and only if there exist two finite words $r, r' \in E^*$, such that $\delta_{\mathcal{P}}^*(q_0^{\mathcal{P}}, r) = q$, $\delta_{\mathcal{P}}^*(q_0^{\mathcal{P}}, r') = p$, and $\beta_{\mathcal{G}}(r, M) = \beta_{\mathcal{G}}(r', M)$. For each edge $e \in E$ and event $y \in Y$, we introduce a binary variable $b_{e,y}$ which is assigned a value 1 if and only if the label of e contains y and the sensor that produces y is chosen to be turned on. More precisely, if $y \in \lambda(e)$ and $u_{\eta(y)} = 1$, then $b_{e,y}$ receives value 1, otherwise it receives value 0, where $\eta(y)$ is the sensor that produces event y , i.e., $\eta(y) = s$ such that $y \in Y_s$. In terms of these variables, an MSSVI instance can be expressed as follows.

Minimize:	$\sum_{s \in S} u_s$	(9.1)
Subject to:	$a_{q_0^{\mathcal{P}}, q_0^{\mathcal{P}}} = 1$	(9.2)

For each $q \in F_{\mathcal{P}}, p \in Q_{\mathcal{P}} \setminus F_{\mathcal{P}}$,

$$a_{q,p} = 0 \tag{9.3}$$

For each $q \in Q_{\mathcal{P}} \setminus F_{\mathcal{P}}, p \in F_{\mathcal{P}}$,

$$a_{q,p} = 0 \tag{9.4}$$

For each $e \in E$ and $y \in Y$ s.t. $y \notin \lambda(e)$,

$$b_{e,y} = 0 \tag{9.5}$$

For each $e \in E$ and $y \in Y$ s.t. $y \in \lambda(e)$,

$$b_{e,y} = u_{\eta(y)} \tag{9.6}$$

For each $q, p \in Q_{\mathcal{P}}$ and $e, d \in E$ such that $\delta_{\mathcal{P}}(q, e) \neq \perp$ and $\delta_{\mathcal{P}}(p, d) \neq \perp$,

$$a_{q,p} = 1 \wedge (\forall y \in Y, b_{e,y} = b_{d,y}) \Rightarrow a_{\delta_{\mathcal{P}}(q,e), \delta_{\mathcal{P}}(p,d)} = 1 \tag{9.7}$$

For each $q, p \in Q_{\mathcal{P}}$ and $e \in E$ such that $\delta_{\mathcal{P}}(q, e) \neq \perp$,

$$a_{q,p} = 1 \text{ and } (\forall y \in Y, b_{e,y} = 0) \Rightarrow a_{\delta_{\mathcal{P}}(q,e), p} = 1 \tag{9.8}$$

For each $q, p \in Q_{\mathcal{P}}$ and $e \in E$ such that $\delta_{\mathcal{P}}(p, e) \neq \perp$,

$$a_{q,p} = 1 \text{ and } (\forall y \in Y, b_{e,y} = 0) \Rightarrow a_{q, \delta_{\mathcal{P}}(p,e)} = 1 \tag{9.9}$$

The objective (9.1) is to minimize the number of sensors turned on. Constraint (9.2) asserts that there exists a world-observation sequence, (the empty string, ϵ) by which both states of the tuple $(q_0^{\mathcal{P}}, q_0^{\mathcal{P}'})$ are reachable from the initial state. Constraint Sets (9.3) and (9.4) ensure that the sensor selection is certifying. Constraint Sets (9.5) and (9.6) encode which sensors affect the label of each edge. Constraint Set (9.7) asserts that if two states q and p are both reachable by a world-observation sequence, then for any edges e and d , if those two edges receive the same world-observation under the chosen sensors, then it means states $\delta_{\mathcal{P}}(q, e)$ and $\delta_{\mathcal{P}}(p, d)$ are also reachable

by at least one sequence of world-observations under the chosen sensors. In fact, these constraints implement the case shown in Fig. 9.2a. Similarly, Constraint Sets (9.8) and (9.9) implement the case in Fig. 9.2b.

This formulation would be a 0–1 integer linear programming model if Constraint Sets (9.7), (9.8), and (9.9) were linear. Thus, the next section shows how to linearize them.

Integer linear programming formulation of MSSVI

To linearize Constraint Set (9.7), first we introduce a binary variable $j_{e,d,y}$ for each $e, d \in E$ and $y \in Y$, which receives its value from the following constraints.

For all $e, d \in E$ and all $y \in Y$,

$$b_{e,y} - b_{d,y} \leq j_{e,d,y}, \quad (9.10)$$

$$b_{d,y} - b_{e,y} \leq j_{e,d,y}, \quad (9.11)$$

$$j_{e,d,y} \leq b_{e,y} + b_{d,y}, \text{ and} \quad (9.12)$$

$$j_{e,d,y} \leq 2 - b_{e,y} - b_{d,y}. \quad (9.13)$$

These linear constraints assign value 0 to $j_{e,d,y}$ if $b_{e,y} = b_{d,y}$; otherwise, they assign value 1 to $j_{e,d,y}$. Hence, Constraint Set (9.7) is replaced by the following linear constraints.

For each $q, p \in Q_{\mathcal{P}}$ and $e, d \in E$ s.t. $\delta_{\mathcal{P}}(q, e) \neq \perp$ and $\delta_{\mathcal{P}}(p, d) \neq \perp$,

$$(1 - a_{q,p}) + \sum_{y \in Y} (j_{e,d,y}) + a_{\delta_{\mathcal{P}}(q,e), \delta_{\mathcal{P}}(p,d)} \geq 1. \quad (9.14)$$

We also replace Constraint Set (9.8) by linearized forms:

For each $q, p \in Q_{\mathcal{P}}$ and $e \in E$ s.t. $\delta_{\mathcal{P}}(q, e) \neq \perp$,

$$(1 - a_{q,p}) + \sum_{y \in Y} b_{e,y} + a_{\delta_{\mathcal{P}}(q,e), p} \geq 1. \quad (9.15)$$

Similarly, we linearize Constraint Set (9.9) as follows.

For each $q, p \in Q_{\mathcal{P}}$ and $e \in E$ s.t. $\delta_{\mathcal{P}}(p, e) \neq \perp$,

$$(1 - a_{q,p}) + \sum_{y \in Y} b_{e,y} + a_{q, \delta_{\mathcal{P}}(p,e)} \geq 1. \quad (9.16)$$

Now, we have an ILP formulation of MSSVI, which can be solved directly by any of the many existing highly-optimized ILP solvers. This ILP formulation not only can be used for obtaining solutions to MSSVI but also to compute feasible (rather than optimal) solutions for large instances of world graphs for which exact solutions to MSSVI cannot be computed in a reasonable amount of time.

9.5 CASE STUDIES

In this section, we present case studies, using the ILP formulation from the previous section to solve some representative instances of MSSVI. All trials were performed on an Ubuntu 16.04 computer with a 3.6 GHz processor.

Case study 1: Computer Science Department

Recall Fig. 9.1, with a map of a small computer science department. For this world graph, we executed several instances of the MSSVI with different itineraries to verify the algorithm’s correctness. Table 9.1 shows results on those instances. The first two instances consider extreme itineraries as boundary test cases. For the first, the itinerary consists of all walks on the world graph, including the empty string; in the second instance, the itinerary does not have any walks over the world graph. In both of these instances, the optimal solution (which has size 0, i.e. no sensor is needed) was correctly found. The third scenario considers an itinerary moving any way, other than entering room G . To certify this itinerary, it suffices to turn on only one of the sensors o_3 and o_4 , both located in room G . Again, our implementation found this solution correctly. The fourth itinerary specifies a single sequence in which the

Table 9.1: Results of our experiment for the environment in Fig. 9.1, which shows the floor map of an environment.

Itinerary	Description	Solution	Comp. time (sec)
1 Walks(\mathcal{G})	All location sequences	\emptyset	2.7
2 $E^* \setminus Walks(\mathcal{G})$	No location sequence	\emptyset	2.3
3 $\{r \in Walks(\mathcal{G}) \mid e_{21} \notin r\}$	Do not enter G	$\{o_3\}$	2.8
4 $e_1e_5e_9e_{15}$	R_1ABR_2D	INFEASIBLE	2.5
5 $e_1e_5e_9e_{15} + e_1e_5e_7e_{13}$	R_1ABR_2D or R_1ABR_1C	$\{o_1, b_1, b_2, o_2, b_3\}$	2.9
6 $e_{12}(e_{11}e_{12} + e_{20}e_{19})^*e_{20}e_{21}$	$R_1R_2(R_1R_2 + R_3R_2)^*R_3G$	$\{o_1, o_2, o_3, b_1\}$	4.1

agent enters room A from R_1 , then it enters room B from the right door between A and B , then passes through R_2 to enter D . There is no certifying sensor selection for this itinerary because there is another walk, $e_1e_5e_7e_{13}$, that is not within the claimed itinerary but which produces the same sequence of world-observations, for any selection of sensors. In contrast, the fifth scenario, whose itinerary includes both the walk from the fourth scenario and additionally $e_1e_5e_7e_{13}$, can be solved with no need to turn on sensors o_3 and o_4 . The last itinerary specifies all walks in which the agent does not enter any room but only room G at the end after traveling along the corridor between any of regions R_1 , R_2 and R_3 . For this itinerary, it is required to turn on sensors o_1 , o_2 and b to verify that the robot did not enter any of rooms A , B , C , D , F . It also requires either o_3 or o_4 be turned on to ensure the robot enters room G at the end. Our program took less than 5 seconds to compute a minimal certifying sensor selection for each of these itineraries.

Though small, this case study suggests the correctness of our algorithm. The next section tests its scalability.

Case study 2: Where eagles soar

Ornithologists employ cellular-network devices to track migratory patterns of larger birds, allowing new insights to be gleaned [90]. Fig. 9.4a shows aggregated tracking data (from [138]) for journeys made by eagles over a year. The surprisingly infrequent

flights across open water might lead one to hypothesize that eagles circle the Caspian Sea (the region made visually salient in the figure). To validate this hypothesis would require purchasing data roaming capabilities from cellular-network operators across multiple countries in this region. Fig. 9.4b shows an approach to model the problem of minimizing these costs. The map is divided into subregions, each representing a vertex of a world graph. Edges of the world graph are between adjacent hexagons. When whole subregions fall substantially within a single country, they have been assigned a color representing the potential of purchasing data service for that country. There are 10 colors, representing the sensor set S . We model the hypothesis of circling the Caspian Sea by an itinerary containing walks that visit III-II-I-III, III-I-II-III, or their extra cyclic permutations. The DFA describing this itinerary consists of 7 states.

The observations provided by the cellular network are akin to the occupancy sensors in the previous example, with events triggered when the eagle enters or leaves each hexagonal cell. Our decomposition, shown in Fig. 9.4b, has 36 colored cells. Accordingly, there are 72 events. An additional 9 cells are uncolored. It took 734.83 seconds for our implementation to form the ILP model and then it took 270.17 seconds to find an optimal solution. In this solution, only 6 out of the 10 sensors (the colors listed in the caption of Fig. 9.4) were turned on. Before finding an optimal solution, the program found feasible solutions of sizes 10, 9, and 8 respectively in 83.07, 90.91, and 176.13 seconds.

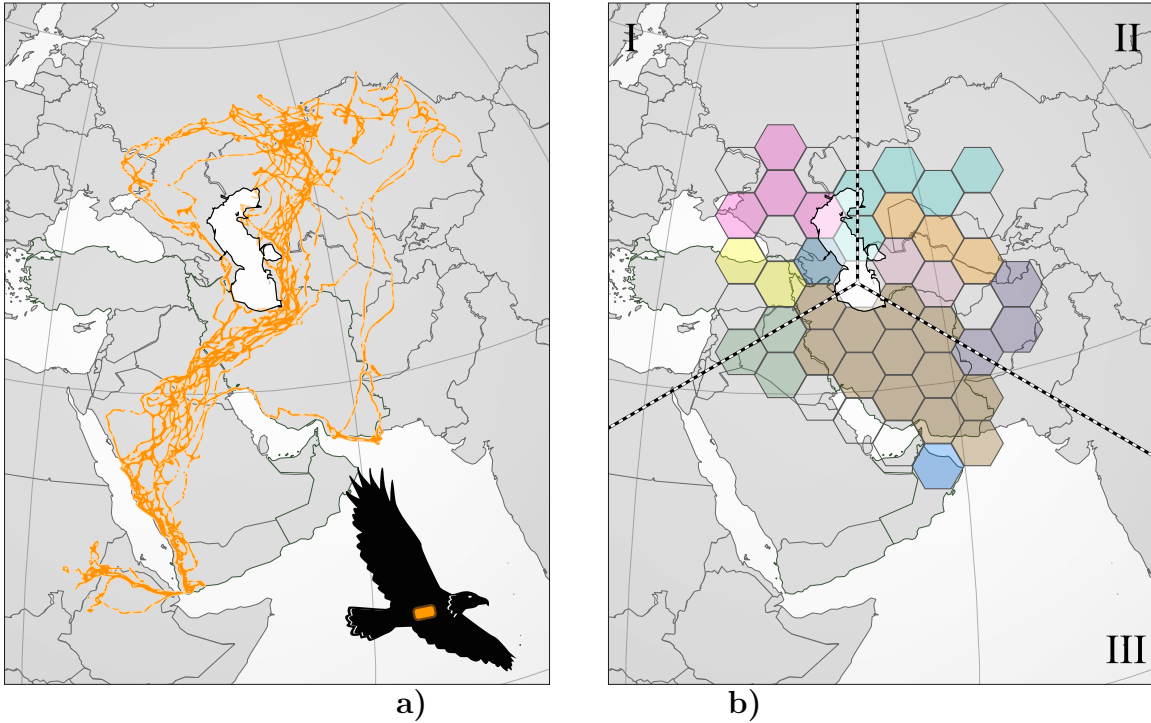








Figure 9.4: **a)** An aggregate of tracking data, reproduced from [138], showing the journeys across parts of North Africa and Central Asia made by eagles over a period of one year. Fig. **b)** The decomposition of the map into subregions. Subregions who fall substantially within a single country have been assigned a color representing the potential of purchasing data roaming service from that country (there are 10 colors, representing sensors set S). The optimal sensor selection to certify the hypothesis of circling the Caspian Sea has six sensors, consisting of the sets of hexagons of the following colors: , , , , , and .

CHAPTER 10

CONCLUSIONS AND FUTURE WORK

In this dissertation, we considered a collection of new, related planning problems that extend the classical point-to-point planning problem with complex goals.

10.1 STATE SPACE REDUCTION OF COMBINATORIAL FILTERS

Combinatorial filters are useful discrete structures for reasoning and filtering in systems that process discrete sensory data. They are also useful for planning, and especially for automata-theoretic approaches to planning. State space reduction of this kind of filters is important many because of several reasons.

1. First, reduced filters provide insight into the nature of the underlying problems that arises from identifying the information required to solve those problems, and this sometimes results to an optimal robotic design in terms of physical and computational resources.
2. Second, reducing combinatorial filters yields smaller memory usage and reduced combinatorial filters can be used on small devices, such as bee robots that are used for flower pollination, that have limited computational and physical resources.
3. Third, reduced filters are useful for communicating information and plans between multiple robots over communication channels that have low-bandwidth or

are noisy.

4. Finally, reducing the size of filters that are used in automata-theoretic approaches for planning leads to have smaller structures and product automata and sometimes reducing the sizes of those structures even by a small fraction significantly improves the performance and computation time to solve a problem.

In this dissertation, we first studied the filter minimization and the filter partitioning minimization problems from the lens of bisimulation. We showed that the bisimulation quotient, which is widely used for reducing the size of transition systems, is not always appropriate for optimally reducing the size of combinatorial filters. However, we also showed that it is useful when one needs to prevent expansion of the language of a filter under minimization. We conclude that both filter minimization and filter partitioning minimization problems can be done by constructing a quotient filter, but for filter partitioning minimization we need to look for an equivalence relation while for filter minimization we need to look for a covering. While any feasible solution to FPM is a feasible solution to FM, a feasible or even an optimal solution to FPM may not be a good feasible solution to FM.

If the union of all compatibility relations or the mergeability relation for a filter is an equivalence relation, then one can optimally reduce the size of that filter. By way of example, we identified several classes of filters for which this is the case.

Knowing that making the quotient of a filter under a compatibility equivalence relation (closed covering) with minimum number of classes produces an optimal solution to FPM (FM), future work might consider the design of efficient heuristic algorithms for finding a such relation (covering). It is also interesting to attempt to identify practical filters for which finding a compatibility equivalence relation with minimum number of classes can be done in polynomial time.

The filter partitioning minimization problem for filters for which the helper graph in Algorithm 2 has no edges is reduced to the problem of *vertex clique partitioning* for the graph of the mergeability relation, and several classes of graphs for which the vertex clique partitioning are in P have been recognized. This approach may provide a roadmap for finding additional classes of filters that can be minimized in polynomial time.

Our experiments show that by using the ILP technique, we can compute exact solutions for large filters for which the brute force algorithm is unable to compute exact solutions. Even for large filters for which we cannot compute an exact solution in a reasonable amount time, we can still resort to the ILP technique to compute feasible solutions that are smaller than those computed by the heuristic algorithm of O’Kane and Shell [20]. Those smaller feasible solutions are unlikely to be computed by randomized versions of the algorithm of O’Kane and Shell even if it is executed thousands of times. This is because forming the equivalence class must ‘globally’ find the ‘mergeable’ states while the algorithm of O’Kane and Shell finds them ‘locally.’ In fact, their algorithm iteratively colors a sequence of conflict graphs, at each step of which it is decided with which states, a state must not be merged, and that decision forces certain decisions to be made at later steps.

We also demonstrated using of this kind of filters in several planning problems with complex goals, beyond the classical point-to-point obstacle-avoidance path planning problem. We consider a variety of problems, including planning for what parts of the world to observe, where to monitor other agents’ movements, and how to achieve desired temporally-extended behaviors in spite of conflicting constraints.

10.2 PLANNING TO CHRONICLE

For planning what parts of the world to observe, we considered the idea of deploying robots to capture and chronicle events that happen in an unpredictable environment to form stories that meet specifications given by the user. We specifically considered the problem of minimizing the expected time to record an event sequence satisfying a set of specifications. This was posed as the problem of computing an optimal policy in an associated partially observable Markov decision problem or Markov decision problem, depending on the level of world-observability. Our implementation confirmed the fact that, by considering global implications, careful planning of observations can improve performance significantly. Also, having studied differing of levels of observability, the results support the intuition that as the robot’s ability to perceive the world improves, the expected number of steps to record a desired story decreases.

Gaps remain between the results presented in this dissertation and the eventual use of real sensors aboard real robots to chronicle sequences of events.

In particular, our approach relies heavily on the event model, which must abstract sufficient detail about the physical environment to model the occurrences of events. In addition, our approach abstracts the details of planning of the robot’s physical movements in its attempt to capture events, but of course those details of motions may have a major impact on the likelihood of successfully capturing events.

In Chapter 8, we tried to mitigate these two problems by modeling the physical environment using a transition system, separately from the event model. Using the transition system allows not only for planning for which events to try to capture but also for where to capture those events. In that chapter, we also considered a variant of the problem where the robot needs to optimally satisfy a set of prioritized soft constraints. Solution to that problem reduced to the problem of computing the set of Pareto optimal policies for a multi-objective Markov decision process.

The applicability of our results to real systems depends directly on the practica-

bility of resolving these abstract elements into fully-realized implementations within a complete system stack. Such a process seems likely to involve a number of challenges, including for example, the tradeoff between granularity of state (which, at some level, must model locations) and computational efficiency.

10.3 TEMPORAL LOGIC PLANNING WITH CONFLICTING SOFT CONSTRAINTS

We also studied planning problem in which high-level specification languages, LTL, LDL_f , and LTL_f , are used for specifying the complex goals and tasks. These languages are user friendly for humans to use, expressive enough for planning to specify complex goals, and precise for robots to manipulate algorithms. We considered temporal logic planning to achieve desired temporally-extended behaviors in spite of conflicting constraints while nevertheless satisfying given hard specifications. We considered this problem in two settings: in one the hard specifications are in LTL but the soft constraints are in LDL_f , and in the other, both the hard specifications and the soft specifications are in LTL. While the latter subsumes the former, they are solved using different algorithms and the algorithm for the former is more efficient compared to when the algorithm for the latter is used for the former. We also consider for the latter problem, synthesizing a shortest trajectory rather than any satisfying trajectory, which may lead to reduce energy consumption and to save resources. These two variants solved in deterministic environments, but we also studied a variant of it for stochastic environment, an extension of the planning to chronicle problem presented in Chapter 5. This problem is reduced to the problem of computing the set of Pareto optimal policies for a multi-objective MDP. In general it is not feasible to compute the set of all those policies even for small MDPs given in general there are many Pareto optimal policies. Accordingly, we combined an approximation method with the convex

hull value iteration method. Our implementation of this algorithm on an MDP with more than 200 state showed the applicability of the approach. For LTL planning, it is essential to make the size of the product automata as small as possible. To do so, one need to abstract away from the transition system those propositions that are irrelevant to the logical formulas of the mission and the soft constraints, to minimize those automata for the missions and the soft constraints, and also to minimize the other structures, such as the filter and the transition system, that participate in the product automaton construction. The well-known techniques of bisimulation and its variants are useful for these minimization processes.

10.4 SENSOR SELECTION FOR DETECTING DEVIATIONS FROM A PLANNED ITINERARY

We also examined the question of selecting the fewest sensors subject to the requirement that they have adequate distinguishing power to differentiate motions conforming to an itinerary from those that do not. This optimization question fits the resource minimization concern that underlies several useful applications. Our formulation of this problem allows for the possibility that when a sensor is selected, it can provide readings for events in potentially multiple places. We proved that this problem is NP-Complete and provided an ILP formulation to solve both exact and approximate solutions. We show the applicability of the approach on two case studies with moderate size.

10.5 OPEN PROBLEMS AND FUTURE WORK

There are several future directions to each of the problems we considered in this dissertation.

For filter reduction, there are several related problems that remain open.

Consider that any optimal solution to FPM can be obtained by finding within the mergeability relation, a compatibility equivalence relation that has a minimum number of equivalence classes as a subset of the mergeability relation. Also, one can show, by example, that there exist filters for which not all pairs related by the mergeability relation are related by a compatibility equivalence relation with minimum number of classes. Accordingly, there exists a relation that relates only those pairs that are related by a compatibility equivalence relation with minimum number of equivalence classes. This relation is a subset of the mergeability relation and one can always find an optimal solution to FPM, by searching for a compatibility equivalence relation within this relation. A question that remains open is whether that relation can be computed in time polynomial in the size of the input filter or not.

Another problem that remains open is the problem of whether a similar relation containing only pairs of states that are within a compatibility class of a closed covering with minimum number of compatibility classes can be computed in polynomial time. Such a relation can be used to search within it for a closed covering with minimum number of compatibility classes to make an optimal solution to FM. In addition, it is not known whether, for each of those two relations, the conditions that assure a pair is related by that relation can be posed in a natural way as conditions to define a variant of the notion of compatibility relation whose definition should be similar in nature to Definition 6.

Note that both closed coverings and compatibility equivalence relations are, in fact, sets of compatibility classes in which all the states within each class are compatible with each other. It is not known under what conditions a compatibility class is within a closed covering (a compatibility equivalence relation) that yields an optimal solution to FM (FPM).

Another interesting open problem is to establish more general conditions that determine for which classes of filters, FM can be solved in polynomial time. The approach utilized in Section 3.9 to identify some such classes was based on analyzing where the union of all compatibility relations becomes an equivalence relation. Another approach, however, would be to impose conditions that guarantee a filter has only a polynomial number of distinct closed coverings. Under such conditions, FM could be solved in polynomial time by enumerating the closed coverings. Additionally, notice that for each of those filters in Section 3.9 for which FM is solvable in polynomial time, there is a single unique closed covering with minimum number of classes. Accordingly, another interesting line of inquiry would be to find classes of filters that have more than one closed covering with minimum number of classes, but still one such closed covering can be constructed in polynomial time. Similar questions may be posed for FPM, considering the mergeability relation instead of the union all compatibility relations and also considering compatibility equivalence relations instead of closed coverings.

Future work can consider designing metrics to measure the level of difficulty of minimizing a given filter, which can be used for deciding an optimal stopping time and deciding which one of the three ILP formulations is more appropriate to use for a given filter. It can also consider computing strong lower bounds, similar to a recent work by van Hove [158], who uses an idea based on decision diagrams for computing lower bounds for the graph coloring problem. Being able to efficiently compute strong lower bounds not only assists accelerate proving optimally, but it also helps decide stopping time for filters that are hard to minimize. Another consideration would be using a ‘better’ relation than the union of all compatibility relations in the ILP formulations. One possibility would be the *mergeability relation*, a subset of the union of all compatibility relations, which consists of only those pairs of compatible states that are related by at least a compatibility equivalence relation. The mergeability

relation will serve a better mean to design a heuristic algorithm for estimating lower bounds, but experiments are required to see if it will affect the quality of solutions or the performance of the ILPs. Another direction would be to apply machine learning techniques to optimize the use of the current work, especially for computing lower and upper bounds and for learning an appropriate time at which the solver must stop in computing solutions for difficult instances of filters.

Future work on temporal logic planning can consider learning soft constraints. It can also consider the case where the environment is dynamic. In this case, the changes are reflected in the product automaton, for which one needs to maintain the SCCs of the automaton in a data structure that is able to quickly adapt to the changes. The problem of planning to chronicle with soft constraints reduced to the problem of computing the set of Pareto optimal policies for the multi-objective MDP. For the MDP, we combined an approximation algorithm with the convex hull value iteration method. This algorithm still needs to be improved both for computing better solutions and for having more efficient algorithms. Given that research is still ongoing on approximate solutions to MOMDPs [86, 130], future work can consider better algorithms, and perhaps algorithms for our specific MDP.

For future work on sensor selection, one can consider the steps that have become standard when dealing with NP-hard problems remain: seeking special-cases that possess some additional structure making them easier, understanding the problem using more nuanced parameterization (i.e., fixed parameter tractability approaches), and custom heuristics and approximation algorithms.

BIBLIOGRAPHY

- [1] Parosh Aziz Abdulla, Johanna Högberg, and Lisa Kaati, *Bisimulation minimization of tree automata*, International Journal of Foundations of Computer Science **18** (2007), no. 04, 699–713.
- [2] Tauhidul Alam, Leonardo Bobadilla, and Dylan A Shell, *Space-efficient filters for mobile robot localization from discrete limit cycles*, IEEE Robotics and Automation Letters **3** (2018), no. 1, 257–264.
- [3] Tomáš Babiak, Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček, *LTL to Büchi automata translation: Fast and more deterministic*, International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2012, pp. 95–109.
- [4] Christel Baier, Holger Hermanns, Joost-Pieter Katoen, and Verena Wolf, *Bisimulation and simulation relations for Markov chains*, Electronic Notes in Theoretical Computer Science **162** (2006), 73–78.
- [5] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen, *Principles of model checking*, MIT press, 2008.
- [6] Leon Barrett and Srinu Narayanan, *Learning all optimal policies with multiple criteria*, Proc. International Conference on Machine learning, 2008, pp. 41–47.
- [7] Patrick Beeson, Matt MacMahon, Joseph Modayil, Aniket Murarka, Benjamin Kuipers, and Brian Stankiewicz, *Integrating multiple representations of spatial knowledge for mapping, navigation, and communication*, Interaction Challenges for Intelligent Assistants, 2007, pp. 1–9.
- [8] Patrick Beeson, Joseph Modayil, and Benjamin Kuipers, *Factoring the mapping problem: Mobile robot map-building in the hybrid spatial semantic hierarchy*, The International Journal of Robotics Research **29** (2010), no. 4, 428–459.
- [9] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi, *Sampling-based motion planning with temporal goals*, Proc. IEEE International Conference on Robotics and Automation, 2010.

- [10] Leonardo Bobadilla, Oscar Sanchez, Justin Czarnowski, and Steven M LaValle, *Minimalist multiple target tracking using directional sensor beams*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011.
- [11] Blai Bonet and Hector Geffner, *Faster heuristic search algorithms for planning with uncertainty and full feedback*, Proc. International Joint Conference on Artificial Intelligence, 2003, pp. 1233–1238.
- [12] Blai Bonet and Héctor Geffner, *Solving POMDPs: RTDP-Bel versus point-based algorithms*, Proc. International Joint Conference on Artificial Intelligence, 2009.
- [13] Doron Bustan and Orna Grumberg, *Simulation-based minimization*, ACM Transactions on Computational Logic (TOCL) **4** (2003), no. 2, 181–206.
- [14] Mingyu Cai, Hao Peng, Zhijun Li, Hongbo Gao, and Zhen Kan, *Receding horizon control-based motion planning with partially infeasible LTL constraints*, IEEE Control Systems Letters **5** (2020), no. 4, 1279–1284.
- [15] Mingyu Cai, Hao Peng, Zhijun Li, and Zhen Kan, *Learning-based probabilistic ltl motion planning with environment and motion uncertainties*, IEEE Transactions on Automatic Control **66** (2020), no. 5, 2386–2392.
- [16] Mingyu Cai, Shaoping Xiao, Zhijun Li, and Zhen Kan, *Optimal probabilistic motion planning with potential infeasible ltl constraints*, IEEE Transactions on Automatic Control (2021).
- [17] Manoel Campêlo, Victor A Campos, and Ricardo C Corrêa, *On the asymmetric representatives formulation for the vertex coloring problem*, Discrete Applied Mathematics **156** (2008), no. 7, 1097–1111.
- [18] Manoel Campêlo, Ricardo Corrêa, and Yuri Frota, *Cliques, holes and the vertex coloring polytope*, Information Processing Letters **89** (2004), no. 4, 159–164.
- [19] Franck Cassez and Stavros Tripakis, *Fault diagnosis with static and dynamic observers*, Fundamenta Informaticae **88** (2008), no. 4, 497–540.
- [20] Luis I Reyes Castro, Pratik Chaudhari, Jana Tumová, Sertac Karaman, Emilio Frazzoli, and Daniela Rus, *Incremental sampling-based algorithm for minimum-violation motion planning*, Proc. IEEE Conference on Decision and Control, IEEE, 2013, pp. 3217–3224.

- [21] Diptanil Chaudhuri, Rhema Ike, Hazhar Rahmani, Dylan A Shell, Aaron T Becker, and Jason M O’Kane, *Conditioning style on substance: Plans for narrative observation*, Proc. IEEE International Conference on Robotics and Automation, IEEE, 2021, pp. 2687–2693.
- [22] Diptanil Chaudhuri, Hazhar Rahmani, Dylan A Shell, and Jason M O’Kane, *Tractable planning for coordinated story capture: Sequential stochastic decoupling*, Proc. International Symposium Distributed Autonomous Robotic Systems, Springer, 2021, pp. 256–268.
- [23] Zhe Chen, *Bayesian filtering: From Kalman filters to particle filters, and beyond*, Statistics **182** (2003), no. 1, 1–69.
- [24] Sandeep P Chinchali, Scott C Livingston, Marco Pavone, and Joel W Burdick, *Simultaneous model identification and task satisfaction in the presence of temporal logic constraints*, Proc. IEEE International Conference on Robotics and Automation, IEEE, 2016, pp. 3682–3689.
- [25] Edmund M Clarke, Orna Grumberg, Kenneth L McMillan, and Xudong Zhao, *Efficient generation of counterexamples and witnesses in symbolic model checking*, Proc. ACM/IEEE Design Automation Conference, 1995, pp. 427–432.
- [26] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen, *The concurrency workbench: A semantics-based tool for the verification of concurrent systems*, ACM Transactions on Programming Languages and Systems (TOPLAS) **15** (1993), no. 1, 36–72.
- [27] Rance Cleaveland and Oleg Sokolsky, *Equivalence and preorder checking for finite-state systems*, Handbook of Process Algebra (2001), 391–424.
- [28] David C Conner, Hadas Kress-Gazit, Howie Choset, Alfred A Rizzi, and George J Pappas, *Valet parking without a valet*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2007, pp. 572–577.
- [29] Costas Courcoubetis, Moshe Vardi, Pierre Wolper, and Mihalis Yannakakis, *Memory-efficient algorithms for the verification of temporal properties*, Formal Methods in System Design **1** (1992), no. 2-3, 275–288.
- [30] Peng Dai and Judy Goldsmith, *Topological value iteration algorithm for Markov decision processes*, Proc. International Joint Conference on Artificial Intelligence, 2007, pp. 1860–1865.

- [31] Giuseppe De Giacomo and Moshe Y Vardi, *Linear temporal logic and linear dynamic logic on finite traces*, Proc. International Joint Conference on Artificial Intelligence, Association for Computing Machinery, 2013, pp. 854–860.
- [32] Rocco De Nicola, *Behavioral equivalences*, Encyclopedia of Parallel Computing (2011), 120–127.
- [33] Jonathan DeCastro, Rüdiger Ehlers, Matthias Rungger, Ayca Balkan, and Hadas Kress-Gazit, *Automated generation of dynamics-based runtime certificates for high-level control*, Discrete Event Dynamic Systems **27** (2017), 371–405.
- [34] Jilles Steeve Dibangoye, Guy Shani, Brahim Chaib-Draa, and Abdell-illah Mouaddib, *Topological order planner for POMDPs*, Proc. International Joint Conference on Artificial Intelligence, 2009.
- [35] Rayna Dimitrova, Mahsa Ghasemi, and Ufuk Topcu, *Maximum realizability for linear temporal logic specifications*, Proc. International Symposium on Automated Technology for Verification and Analysis, 2018.
- [36] Rüdiger Ehlers, *Short witnesses and accepting lassos in ω -automata*, Proc. International Conference on Language and Automata Theory and Applications, Springer, 2010, pp. 261–272.
- [37] Michael A Erdmann and Matthew T Mason, *An exploration of sensorless manipulation*, IEEE Journal on Robotics and Automation **4** (1988), no. 4, 369–379.
- [38] Lawrence H Erickson, Jingjin Yu, Yaonan Huang, and Steven M LaValle, *Counting moving bodies using sparse sensor beams*, Proc. International Workshop on the Algorithmic Foundations of Robotics, 2012.
- [39] Georgios E Fainekos, *Revising temporal logic specifications for motion planning*, Proc. IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 40–45.
- [40] Georgios E Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J Pappas, *Temporal logic motion planning for dynamic robots*, Automatica **45** (2009), no. 2, 343–352.
- [41] Georgios E Fainekos, Antoine Girard, and George J Pappas, *Hierarchical synthesis of hybrid controllers from temporal logic specifications*, Proc. International Workshop on Hybrid Systems: Computation and Control, 2007.

- [42] Seyedshams Feyzabadi and Stefano Carpin, *Multi-objective planning with multiple high level task specifications*, Proc. IEEE International Conference on Robotics and Automation, IEEE, 2016, pp. 5483–5490.
- [43] Marcelo P Fiore, *A coinduction principle for recursive data types based on bisimulation*, Information and Computation **127** (1996), no. 2, 186–198.
- [44] Marco Forti and Furio Honsell, *Set theory with free construction principles*, Annali della Scuola Normale Superiore di Pisa-Classe di Scienze **10** (1983), no. 3, 493–522.
- [45] Jie Fu, *Probabilistic planning with preferences over temporal goals*, Proc. American Control Conference, IEEE, 2021, pp. 4854–4859.
- [46] Paul Gastin and Denis Oddoux, *Fast LTL to Büchi automata translation*, Proc. International Conference on Computer Aided Verification, Springer, 2001, pp. 53–65.
- [47] Raffaella Gentilini, Carla Piazza, and Alberto Policriti, *From bisimulation to simulation: Coarsest partition problems*, Journal of Automated Reasoning **31** (2003), no. 1, 73–103.
- [48] Yogesh Girdhar and Gregory Dudek, *Efficient on-line data summarization using extremum summaries*, Proc. IEEE International Conference on Robotics and Automation, 2012, pp. 3490–3496.
- [49] Robert Givan, Thomas Dean, and Matthew Greig, *Equivalence notions and model minimization in Markov decision processes*, Artificial Intelligence **147** (2003), no. 1-2, 163–223.
- [50] Kenneth Y Goldberg, *Orienting polygonal parts without sensors*, Algorithmica **10** (1993), no. 2, 201–225.
- [51] Boqing Gong, Wei-Lun Chao, Kristen Grauman, and Fei Sha, *Diverse sequential subset selection for supervised video summarization*, Proc. Advances in Neural Information Processing Systems, 2014, pp. 2069–2077.
- [52] Roberto Gorrieri and Cristian Versari, *Transition systems and behavioral equivalences*, Introduction to Concurrency Theory, Springer, 2015, pp. 21–79.

- [53] Michael Gygli, Helmut Grabner, Hayko Riemenschneider, and Luc Van Gool, *Creating summaries from user videos*, Proc. European Conference on Computer Vision, 2014, pp. 505–520.
- [54] Esfandiar Haghverdi, Paulo Tabuada, and George J Pappas, *Bisimulation relations for dynamical, control, and hybrid systems*, Theoretical Computer Science **342** (2005), no. 2-3, 229–261.
- [55] Alireza Haji-Valizadeh and Kenneth A Loparo, *Minimizing the cardinality of an events set for supervisors of discrete-event dynamical systems*, IEEE Transactions on Automatic Control **41** (1996), no. 11, 1579–1593.
- [56] Mohammad Hekmatnejad and Georgios Fainekos, *Optimal multi-valued ltl planning for systems with access right levels*, Proc. American Control Conference, IEEE, 2018, pp. 2363–2370.
- [57] Monika Rauch Henzinger, Thomas A Henzinger, and Peter W Kopke, *Computing simulations on finite and infinite graphs*, Proc. IEEE 36th Annual Foundations of Computer Science, IEEE, 1995, pp. 453–462.
- [58] Yu-Chi Ho and Robert CK Lee, *A Bayesian approach to problems in stochastic estimation and control*, IEEE Transactions on Automatic Control **9** (1964), no. 4, 333–339.
- [59] Johanna Högberg, Andreas Maletti, and Jonathan May, *Backward and forward bisimulation minimization of tree automata*, Theoretical Computer Science **410** (2009), no. 37, 3539–3552.
- [60] Adalat Jabrayilov and Petra Mutzel, *New integer linear programming models for the vertex coloring problem*, Proc. Latin American Symposium on Theoretical Informatics, Springer, 2018, pp. 640–652.
- [61] Zhong Ji, Kailin Xiong, Yanwei Pang, and Xuelong Li, *Video summarization with attention-based encoder–decoder networks*, IEEE Transactions on Circuits and Systems for Video Technology **30** (2019), no. 6, 1709–1717.
- [62] Sebastian Junges, Nils Jansen, and Sanjit A Seshia, *Enforcing almost-sure reachability in pomdps*, Proc. International Conference on Computer Aided Verification, Springer, 2021, pp. 602–625.
- [63] Rudolph Emil Kalman, *A new approach to linear filtering and prediction problems*, Transaction of the ASME, Journal of Basic Engineering **82** (1960), 34–45.

- [64] Paris C Kanellakis and Scott A Smolka, *CCS expressions, finite state processes, and three problems of equivalence*, Information and Computation **86** (1990), no. 1, 43–68.
- [65] Richard M Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations, Springer, 1972, pp. 85–103.
- [66] Emil Keyder and Hector Geffner, *The HMDPP planner for planning with probabilities*, Sixth International Planning Competition at ICAPS **8** (2008).
- [67] Kangjin Kim, Georgios Fainekos, and Sriram Sankaranarayanan, *On the minimal revision problem of specification automata*, The International Journal of Robotics Research **34** (2015), no. 12, 1515–1535.
- [68] Kangjin Kim, Georgios E Fainekos, and Sriram Sankaranarayanan, *On the revision problem of specification automata*, Proc. IEEE International Conference on Robotics and Automation, IEEE, 2012, pp. 5171–5176.
- [69] Andrey Kolobov, Mausam, and Daniel S, *A theory of goal-oriented MDPs with dead ends*, Proc. Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2012, p. 438–447.
- [70] Andrey Kolobov, Mausam, and Daniel S Weld, *Sixthsense: Fast and reliable recognition of dead ends in MDPs*, Proc. Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010.
- [71] Stavros Konstantinidis, *Computing the edit distance of a regular language*, Information and Computation **205** (2007), no. 9, 1307–1316.
- [72] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman, *Synthesis for robots: Guarantees and feedback for robot behavior*, Annual Review of Control, Robotics, and Autonomous Systems **1** (2018), no. 1, 211–236.
- [73] Shawn M Kristek and Dylan A Shell, *Orienting deformable polygonal parts without sensors*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.
- [74] Guillermo Laguna, Rafael Murrieta-Cid, Hector M Becerra, Rigoberto Lopez-Padilla, and Steven M LaValle, *Exploration of an unknown environment with a differential drive disc robot*, Proc. IEEE International Conference on Robotics and Automation, 2014.

- [75] Morteza Lahijanian, Shaull Almagor, Dror Fried, Lydia E Kavraki, and Moshe Y Vardi, *This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction*, Proc. AAAI Conference on Artificial Intelligence, 2015.
- [76] Morteza Lahijanian and Marta Kwiatkowska, *Specification revision for Markov decision processes with optimal trade-off*, Proc. Conference on Decision and Control, IEEE, 2016, pp. 7411–7418.
- [77] Steven M LaValle, *Planning algorithms*, Cambridge University Press, Cambridge, U.K., 2006, Available at <http://planning.cs.uiuc.edu/>.
- [78] ———, *Sensing and filtering: A fresh perspective based on preimages and information spaces*, Foundations and Trends® in Robotics **1** (2012), no. 4, 253–372.
- [79] Yong Jae Lee, Joydeep Ghosh, and Kristen Grauman, *Discovering important people and objects for egocentric video summarization*, Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 1346–1353.
- [80] Vladimir I Levenshtein, *Binary codes capable of correcting deletions, insertions, and reversals*, Soviet Physics Doklady **10** (1966), no. 8, 707–710.
- [81] Meilun Li, Zhikun She, Andrea Turrini, and Lijun Zhang, *Preference planning for Markov decision processes*, Proc. Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
- [82] Iain Little and Sylvie Thiébaux, *Probabilistic planning vs. replanning*, ICAPS Workshop on International Planning Competition: Past, Present and Future, 2007.
- [83] Rigoberto Lopez-Padilla, Rafael Murrieta-Cid, and Steven M LaValle, *Optimal gap navigation for a disc robot*, Proc. International Workshop on the Algorithmic Foundations of Robotics, Springer, 2013, pp. 123–138.
- [84] Zheng Lu and Kristen Grauman, *Story-driven summarization for egocentric video*, Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 2714–2721.
- [85] Behrooz Mahasseni, Michael Lam, and Sinisa Todorovic, *Unsupervised video summarization with adversarial lstm networks*, Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 202–211.

- [86] Lawrence Mandow, José-Luis Perez-de-la Cruz, and N Pozas, *Multi-objective dynamic programming with limited precision*, Journal of Global Optimization (2021), 1–20.
- [87] Matthew T Mason, *Kicking the sensing habit*, AI magazine **14** (1993), no. 1, 58–58.
- [88] Cl Masreliez and R Martin, *Robust Bayesian estimation for the linear model and robustifying the Kalman filter*, IEEE Transactions on Automatic Control **22** (1977), no. 3, 361–371.
- [89] Isabel Méndez-Díaz and Paula Zabala, *A cutting plane algorithm for graph coloring*, Discrete Applied Mathematics **156** (2008), no. 2, 159–179.
- [90] Bernd-U Meyburg, Patrick Paillat, and Christiane Meyburg, *Migration routes of steppe eagles between asia and africa: a study by means of satellite telemetry*, The Condor **105** (2003), no. 2, 219–227.
- [91] Robin Milner, *A formal notion of simulation between programs*, Memo 14, Computers and Logic Research Group, University College of Swansea, UK, 1970.
- [92] ———, *An algebraic definition of simulation between programs*, Proc. International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1971, p. 481–489.
- [93] ———, *Program simulation: An extended formal notion*, Memo 17, Computers and Logic Research Group, University College of Swansea, UK, 1971.
- [94] ———, *A calculus of communicating systems*, Lecture Notes in Computer Science 92 (1980).
- [95] Salar Moarref and Hadas Kress-Gazit, *Decentralized control of robotic swarms from high-level temporal logic specifications*, Proc. International Symposium on Multi-robot and Multi-agent Systems (MRS), IEEE, 2017, pp. 17–23.
- [96] Joseph Modayil, Patrick Beeson, and Benjamin Kuipers, *Using the topological skeleton for scalable global metrical map-building*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 2, IEEE, 2004, pp. 1530–1536.

- [97] Lorenzo Nardi and Cyrill Stachniss, *Experience-based path planning for mobile robots exploiting user preferences*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2016, pp. 1170–1176.
- [98] Jason M O’Kane and Dylan A Shell, *Automatic reduction of combinatorial filters*, Proc. IEEE International Conference on Robotics and Automation, 2013.
- [99] Joseph O’Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, 1987.
- [100] Jason M O’Kane and Dylan A Shell, *Concise planning and filtering: hardness and algorithms*, IEEE Transactions on Automation Science and Engineering **14** (2017), no. 4, 1666–1681.
- [101] Robert Paige and Robert E Tarjan, *Three partition refinement algorithms*, SIAM Journal on Computing **16** (1987), no. 6, 973–989.
- [102] David Park, *Concurrency and automata on infinite sequences*, Theoretical Computer Science, Springer, 1981, pp. 167–183.
- [103] Marco Peressotti and Tomasz Brengos, *Behavioural equivalences for timed systems*, Logical Methods in Computer Science **15** (2019).
- [104] Bryan A Plummer, Matthew Brown, and Svetlana Lazebnik, *Enhancing video summarization via vision-language embedding*, Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5781–5789.
- [105] Michael O Rabin and Dana Scott, *Finite automata and their decision problems*, IBM Journal of Research and Development **3** (1959), no. 2, 114–125.
- [106] Hazhar Rahmani and Jason M O’Kane, *On the relationship between bisimulation and combinatorial filter reduction*, Proc. IEEE International Conference on Robotics and Automation, IEEE, 2018, pp. 7314–7321.
- [107] Hazhar Rahmani and Jason M O’Kane, *Optimal temporal logic planning with cascading soft constraints*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2019, pp. 2524–2531.
- [108] ———, *Integer linear programming formulations of the filter partitioning minimization problem*, Journal of Combinatorial Optimization (2020), 1–23.

- [109] ———, *What to do when you can't do it all: Temporal logic planning with soft temporal logic constraints*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2020, pp. 6619–6626.
- [110] ———, *Equivalence notions for state-space minimization of combinatorial filters*, IEEE Transactions on Robotics **37** (2021), no. 6, 2117–2136.
- [111] Hazhar Rahmani, Dylan A Shell, and Jason M O’Kane, *Planning to chronicle*, Algorithmic Foundations of Robotics XIV, Springer International Publishing, 2021, pp. 277–293.
- [112] Hazhar Rahmani, Dylan A Shell, and Jason M O’Kane, *Sensor selection for detecting deviations from a planned itinerary*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2021, pp. 6511–6518.
- [113] ———, *Planning to chronicle: Optimal policies for narrative observation of unpredictable events*, International Journal of Robotics Research (2022).
- [114] Vasumathi Raman and Hadas Kress-Gazit, *Explaining impossible high-level robot behaviors*, IEEE Transactions on Robotics **29** (2013), no. 1, 94–104.
- [115] Francesco Ranzato, *An efficient simulation algorithm on kripke structures*, Acta Informatica **51** (2014), no. 2, 107–125.
- [116] Francesco Ranzato and Francesco Tapparo, *An efficient simulation algorithm based on abstract interpretation*, Information and Computation **208** (2010), no. 1, 1–22.
- [117] Mark O Riedl and Robert Michael Young, *Narrative planning: Balancing plot and character*, Journal of Artificial Intelligence Research **39** (2010), 217–268.
- [118] Justus Robertson and R Michael Young, *Narrative mediation as probabilistic planning*, Proc. Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference, 2017.
- [119] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley, *A survey of multi-objective sequential decision-making*, Journal of Artificial Intelligence Research **48** (2013), 67–113.
- [120] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa, *Online planning algorithms for POMDPs*, Journal of Artificial Intelligence Research **32** (2008), 663–704.

- [121] Jurriaan Rot, Marcello Bonsangue, and Jan Rutten, *Proving language inclusion and equivalence by coinduction*, Information and Computation **246** (2016), 62–76.
- [122] ———, *Proving language inclusion and equivalence by coinduction*, Information and Computation **246** (2016), 62–76.
- [123] Jan JMM Rutten, *Automata and coinduction (an exercise in coalgebra)*, Proc. International Conference on Concurrency Theory, Springer, 1998, pp. 194–218.
- [124] ———, *Universal coalgebra: a theory of systems*, Theoretical computer science **249** (2000), no. 1, 3–80.
- [125] Fatemeh Zahra Saberifar, Ali Mohades, Mohammadreza Razzazi, and Jason M O’Kane, *Combinatorial filter reduction: Special cases, approximation, and fixed-parameter tractability*, Journal of Computer and System Sciences **85** (2017), 74–92.
- [126] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis, *Diagnosability of discrete-event systems*, IEEE Transactions on Automatic Control **40** (1995), no. 9, 1555–1575.
- [127] Davide Sangiorgi, *On the origins of bisimulation and coinduction*, ACM Transactions on Programming Languages and Systems (TOPLAS) **31** (2009), no. 4, 15.
- [128] Klaus U Schulz and Stoyan Mihov, *Fast string correction with Levenshtein automata*, International Journal on Document Analysis and Recognition **5** (2002), no. 1, 67–85.
- [129] David Sears and Karen Rudie, *Minimal sensor activation and minimal communication in discrete-event systems*, Discrete Event Dynamic Systems **26** (2016), no. 2, 295–349.
- [130] Ekaterina Sedova, Lawrence Mandow, and José-Luis Pérez-de-la Cruz, *Asynchronous vector iteration in multi-objective Markov decision processes*, Conference of the Spanish Association for Artificial Intelligence, Springer, 2021, pp. 129–138.
- [131] Guy Shani, Joelle Pineau, and Robert Kaplow, *A survey of point-based POMDP solvers*, Autonomous Agents and Multi-Agent Systems **27** (2013), no. 1, 1–51.

- [132] Asaf Shapira, Raphael Yuster, and Uri Zwick, *All-pairs bottleneck paths in vertex weighted graphs*, *Algorithmica* **59** (2011), no. 4, 621–633.
- [133] Dylan A Shell, Li Huang, Aaron T Becker, and Jason M O’Kane, *Planning coordinated event observation for structured narratives*, Proc. IEEE International Conference on Robotics and Automation, 2019, pp. 7632–7638.
- [134] Thomas R Shiple, *Survey of equivalences for transition systems*, University of California, Berkeley, EE 290A Class Report.
- [135] Christoffer Sloth, George J Pappas, and Rafael Wisniewski, *Compositional safety analysis using barrier certificates*, Proc. International Conference on Hybrid Systems: Computation and Control, 2012, pp. 15–24.
- [136] Stephen L Smith, Jana Tumova, Calin Belta, and Daniela Rus, *Optimal path planning under temporal logic constraints*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp. 3288–3293.
- [137] Fabio Somenzi and Roderick Bloem, *Efficient Büchi automata from LTL formulae*, Proc. International Conference on Computer Aided Verification, Springer, 2000, pp. 248–263.
- [138] Jessica Stewart, *Incredible Map Shows Flight Routes of Eagles Over the Course of One Year*, March 6, 2019, Last access: February 27, 2021.
- [139] Subhash Suri, Elias Vicari, and Peter Widmayer, *Simple robots with minimal sensing: From local visibility to global geometry*, *International Journal of Robotics Research* **27** (2008), no. 9, 1055–1067.
- [140] Tae-Sic Yoo and S. Lafortune, *NP-completeness of sensor selection problems arising in partially observed discrete-event systems*, *IEEE Transactions on Automatic Control* **47** (2002), no. 9, 1495–1499.
- [141] Li Tan and Rance Cleaveland, *Simulation revisited*, International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2001, pp. 480–495.
- [142] Robert Tarjan, *Depth-first search and linear graph algorithms*, *SIAM Journal on Computing* **1** (1972), no. 2, 146–160.

- [143] Takashi Tomita, Atsushi Ueno, Masaya Shimakawa, Shigeki Hagihara, and Naoki Yonezaki, *Safraless LTL synthesis considering maximal realizability*, Acta Informatica **54** (2017), no. 7, 655–692.
- [144] B. Tovar, F. Cohen, and S. M. LaValle, *Sensor beams, obstacles, and possible paths*, Algorithmic Foundations of Robotics VIII, Springer, 2009, pp. 317–332.
- [145] Benjamin Tovar, *Minimalist models and methods for visibility-based tasks*, University of Illinois at Urbana-Champaign, 2009.
- [146] Benjamin Tovar, Fred Cohen, Leonardo Bobadilla, Justin Czarnowski, and Steven M LaValle, *Combinatorial filters: Sensor beams, obstacles, and possible paths*, ACM Transactions on Sensor Networks (TOSN) **10** (2014), no. 3, 47.
- [147] Benjamin Tovar, Fred Cohen, and Steven LaValle, *Sensor beams, obstacles, and possible paths*, Proc. International Workshop on the Algorithmic Foundations of Robotics (2009), 317–332.
- [148] Benjamín Tovar, Luigi Freda, and Steven M LaValle, *Learning combinatorial map information from permutations of landmarks*, The International Journal of Robotics Research **30** (2011), no. 9, 1143–1156.
- [149] Benjamn Tovar, Rafael Murrieta-Cid, and Steven M LaValle, *Distance-optimal navigation in an unknown environment without sensing distances*, IEEE Transactions on Robotics **23** (2007), no. 3, 506–518.
- [150] Ba Tu Truong and Svetha Venkatesh, *Video abstraction: A systematic review and classification*, ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) **3** (2007), no. 1, 3–es.
- [151] Jana Tumova, Luis I Reyes Castro, Sertac Karaman, Emilio Frazzoli, and Daniela Rus, *Minimum-violation LTL planning with conflicting specifications*, Proc. American Control Conference, IEEE, 2013, pp. 200–205.
- [152] Jana Tumova, Gavin C Hall, Sertac Karaman, Emilio Frazzoli, and Daniela Rus, *Least-violating control strategy synthesis with safety rules*, Proc. International Conference on Hybrid Systems: Computation and Control, 2013.
- [153] Johan van Benthem, *Modal correspondence theory*, Ph.D. thesis, University of Amsterdam, Netherlands, 1976.

- [154] Arjan van der Schaft, *Equivalence of dynamical systems by bisimulation*, IEEE Transactions on Automatic Control **49** (2004), no. 12, 2160–2172.
- [155] Rob van Glabbeek and Bas Ploeger, *Correcting a space-efficient simulation algorithm*, Proc. International Conference on Computer Aided Verification, Springer, 2008, pp. 517–529.
- [156] Rob J van Glabbeek, *The linear time—branching time spectrum ii*, Proc. International Conference on Concurrency Theory, Springer, 1993, pp. 66–81.
- [157] ———, *The linear time-branching time spectrum i. the semantics of concrete, sequential processes*, Handbook of Process Algebra, Elsevier, 2001, pp. 3–99.
- [158] Willem-Jan van Hoeve, *Graph coloring lower bounds from decision diagrams*, Proc. International Conference on Integer Programming and Combinatorial Optimization, Springer, 2020, pp. 405–418.
- [159] Moshe Y Vardi and Pierre Wolper, *An automata-theoretic approach to automatic program verification*, Proc. First Symposium on Logic in Computer Science, IEEE Computer Society, 1986.
- [160] ———, *Reasoning about infinite computations*, Information and Computation **115** (1994), no. 1, 1–37.
- [161] Weilin Wang, Chaohui Gong, and Di Wang, *Optimizing sensor activation in a language domain for fault diagnosis*, IEEE Transactions on Automatic Control **64** (2018), no. 2, 743–750.
- [162] Andrew M. Wells, Morteza Lahijanian, Lydia E. Kavrakı, and Moshe Y. Vardi, *LTLf synthesis on probabilistic systems*, Proc. Eleventh International Symposium on Games, Automata, Logics, and Formal Verification (Brussels, Belgium), Elsevier, Sep. 2020.
- [163] Nils Wilde, Dana Kulić, and Stephen L Smith, *Learning user preferences in robot motion planning through interaction*, Proc. IEEE International Conference on Robotics and Automation, 2018.
- [164] Kyle Hollins Wray, Shlomo Zilberstein, and Abdel-İllah Mouaddib, *Multi-objective mdps with conditional lexicographic reward preferences*, Proc. Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.

- [165] Hengyang Wu and Yuxin Deng, *Logical characterizations of simulation and bisimulation for fuzzy transition systems*, Fuzzy Sets and Systems **301** (2016), 19–36.
- [166] Xiang Yin and Stéphane Lafortune, *A general approach for optimizing dynamic sensor activation for discrete event systems*, Automatica **105** (2019), 376–383.
- [167] Jingjin Yu and Steven M LaValle, *Cyber detectives: Determining when robots or people misbehave*, Proc. International Workshop on the Algorithmic Foundations of Robotics, Springer, 2010, pp. 391–407.
- [168] ———, *Story validation and approximate path inference with a sparse network of heterogeneous sensors*, Proc. IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 4980–4985.
- [169] ———, *Shadow information spaces: Combinatorial filters for tracking targets*, IEEE Transactions on Robotics **28** (2012), no. 2, 440–456.
- [170] Ke Zhang, Kristen Grauman, and Fei Sha, *Retrospective encoders for video summarization*, Proc. European Conference on Computer Vision, 2018, pp. 383–399.
- [171] Yulin Zhang, Hazhar Rahmani, Dylan A Shell, and Jason M O’Kane, *Accelerating combinatorial filter reduction through constraints*, Proc. IEEE International Conference on Robotics and Automation, IEEE, 2021, pp. 9703–9709.
- [172] Yulin Zhang and Dylan A Shell, *Cover combinatorial filters and their minimization problem*, Proc. International Workshop on the Algorithmic Foundations of Robotics, 2020.