

Dynamic Execution of Temporal Plans with Sensing Actions and Bounded Risk

by

Pedro Henrique de Rodrigues Quemel e Assis Santana

Submitted to the Department of Aeronautics and Astronautics

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Aerospace Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

© Massachusetts Institute of Technology 2016. All rights reserved.

Author
Department of Aeronautics and Astronautics
August 15th, 2016

Certified by.....
Brian C. Williams
Professor of Aeronautics and Astronautics, MIT, Thesis Supervisor

Certified by.....
Tomás Lozano-Pérez
Professor of Computer Science & Engineering, MIT

Certified by.....
Sertac Karaman
Associate Professor of Aeronautics and Astronautics, MIT

Certified by.....
Sylvie Thiébaux
Professor of Computer Science, ANU, and Research Leader, NICTA

Certified by.....
Shlomo Zilberstein
Professor of Computer Science, UMass Amherst

Accepted by.....
Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Dynamic Execution of Temporal Plans with Sensing Actions and Bounded Risk

by

Pedro Henrique de Rodrigues Quemel e Assis Santana

Submitted to the Department of Aeronautics and Astronautics
on August 15th, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aerospace Engineering

Abstract

A special report on the cover of the June 2016 issue of the IEEE Spectrum magazine reads: “can we trust robots?” In a world that has been experiencing a seemingly irreversible process by which autonomous systems have been given increasingly more space in strategic areas such as transportation, manufacturing, energy supply, planetary exploration, and even medical surgeries, it is natural that we start asking ourselves if these systems could be held at the same or even higher levels of safety than we expect from humans. In an effort to make a contribution towards a world of autonomy that we can trust, this thesis argues that one necessary step in this direction is the endowment of autonomous agents with the ability to dynamically adapt to their environment while meeting strict safety guarantees.

From a technical standpoint, we propose that autonomous agents in safety-critical applications be able to execute *conditional plans* (or *policies*) within *risk bounds* (also referred to as *chance constraints*). By being conditional, the plan allows the autonomous agent to adapt to its environment in real-time by conditioning the choice of activity to be executed on the agent’s current level of knowledge, or belief, about the true state of world. This belief state is, in turn, a function of the history of potentially noisy sensor observations gathered by the agent from the environment. With respect to *bounded risk*, it refers to the fact that executing such conditional plans should guarantee to keep the agent “safe” - as defined by sets of state constraints - with high probability, while moving away from the conservatism of *minimum risk* approaches.

In this thesis, we propose Chance-Constrained Partially Observable Markov Decision Processes (CC-POMDP’s) as a formalism for conditional risk-bounded planning under uncertainty. Moreover, we present Risk-bounded AO* (RAO*), a heuristic forward search-based algorithm that searches for solutions to a CC-POMDP by leveraging admissible utility and risk heuristics to simultaneously guide the search and perform early pruning of overly-risky policy branches. In an effort to facilitate the specification of risk-bounded behavior by human modelers, we also present the Chance-constrained Reactive Model-based Programming Language (cRMPL), a novel variant of RMPL that incorporates chance constraints as part of its syntax. Finally,

in support of the temporal planning applications with duration uncertainty that this thesis is concerned about, we present the Polynomial-time Algorithm for Risk-aware Scheduling (PARIS) and its extension to conditional scheduling of Probabilistic Temporal Plan Networks (PTPN's).

The different tools and algorithms developed in the context of this thesis are combined to form the Conditional Planning for Autonomy with Risk (CLARK) system, a risk-aware conditional planning system that can generate chance-constrained, dynamic temporal plans for autonomous agents that must operate under uncertainty. With respect to our empirical validation, each component of CLARK is benchmarked against the relevant state of the art throughout the chapters, followed by several demonstrations of the whole CLARK system working in tandem with other building blocks of an architecture for autonomy.

Thesis Supervisor: Brian C. Williams

Title: Professor of Aeronautics and Astronautics, MIT

Thesis Committee Member: Tomás Lozano-Pérez

Title: Professor of Computer Science & Engineering, MIT

Thesis Committee Member: Sertac Karaman

Title: Associate Professor of Aeronautics and Astronautics, MIT

Thesis Committee Member: Sylvie Thiébaux

Title: Professor of Computer Science, ANU, and Research Leader, NICTA

Thesis Committee Member: Shlomo Zilberstein

Title: Professor of Computer Science, UMass Amherst

Acknowledgments

First and foremost, I would like to thank my advisor, Brian Williams, for his patience, support, brilliant insights, and many opportunities during my time as his student in the MERS group. *It is the case* that you gave me the opportunity of a lifetime by accepting me into MIT, and provided me with the tools to fulfill dreams that I never thought would be possible. *Does that make sense*, or should I clarify with some *grounded examples*?

I would also like to thank my thesis committee members, Tomás Lozano-Pérez, Sertac Karaman, Sylvie Thiébaux, and Shlomo Zilberstein, for their exceptionally constructive and accurate feedback throughout the process of making this thesis, and for being such a thoughtful sounding board for my ideas and future goals. For all her guidance during my time at NICTA, our productive collaboration, and her hatred of my semicolons, Sylvie deserves an extra round of gratitude and appreciation. Moreover, I would like to thank Julie Shah, the external evaluator in my thesis proposal defense, whose invaluable remarks turned into a whole chapter in my thesis.

To my family, my eternal love and gratitude for their Herculean effort and sacrifices to turn me into a well-functioning adult, and for their understanding that the process did not go as planned. To my father, Paulo, the engineer that I will always want to be; to my mother, Angélica, for packing more love per square inch than any other person on the planet; to my second mother, Maria José, for making me feel as her own son, even though this son is... well, me. To my siblings, Patrícia and Fernando, for their companionship and supernatural knack for remembering every embarrassing moment of my life.

To my beautiful wife, Luiza, who gave me strength every time I felt I could no longer bear the weight. Thank you for accepting my imperfect self in the warm comfort of your love, and for our countless moments of shared joy. For that, I owe you my lifelong gratitude and love, and a couple of those annoying little dogs that will not stop barking!

Navigating the winding and steep road of a thesis would have been much harder,

or even impossible, without the daily guidance of postdocs, research scientists, and visiting scholars with whom I had the privilege to interact. Among those, I would like to thank Andreas Hofmann for his constant support throughout this thesis, and his unusual ability to remain calm and point us in the right direction when things seemed too hard to accomplish; Tiago Vaquero, Erez Karpas, and Christian Muise, for all the enlightening and productive discussions permeated by moments of sheer silliness; Geovany Borges and Bruno Adorno, who mentored me at the University of Brasília, and shone a bit more of their brilliance upon me during their visits to MIT; and Cláudio Toledo, whose excitement about his work and open-minded approach to research catapulted my thesis forward.

My sincere gratitude also goes to Beth Marois and Jason McKnight, for their unwavering support in reducing the entropy of my sometimes chaotic life as a Ph.D. student. I would also like to thank Jonathan Smith from UPenn, who is deserving of my deepest admiration and appreciation, despite our short period of interaction.

I am thankful to my dear labmates for sharing their genius with me, and making my time in MERS a memorable period of my life. After a long discussion with Andrew about whether the order of appearance in the acknowledgements means anything, I decided to list people alphabetically. Therefore, thank you Ameya, for clubbing like a boss; Andrew, for being an extraordinarily special Texas snowflake; Ben, for being half British, half Yeeeeeehaw!; Bobby, for inspiring this thesis and teaching me about socially acceptable chocolate; Dan, for teaching me how to properly say “robot”, and for winning the contest of how long it would take Steve to sit after he adopted a standing desk; David, for always knowing how to do things; Enrique, for his impeccable professionalism; Eric, for the outing adventures and keeping our lab running like a clock; Hiro, for the impromptu Pink Floyd performances at 2 AM in the lab; James, for showing us that rugby is, indeed, a gentleman’s sport; Johannes, for shining both figuratively and literally; Jonathan, for keeping MERS classy; Larry, for milkshakes that bring all the boys to the yard; Luis (a.k.a. Postinho), for being as awesome of a friend in the Northern, as in the Southern hemisphere; Peng, for never being too busy to help or in a bad mood, ever; Shannon, for the amazing

tips about graduate life on my very first day at MIT; Simon, for saying the most inappropriate things at the most appropriate times; Spencer, for being an example of resilience; Steve, for accepting me into the Jewish faith, and never really charging me any licensing fees; Szymon, for teaching me so much about Poland; and Yuki, for making me feel at home in Japan.

My sincere gratitude also goes to Alban, Menkes, Pascal, Patrik, Phil, Scott, Frank, Philip, Karsten, Boon Ping, Jing, and all the other remarkable people I had the privilege of working with during my time at NICTA (or should I say Data61?); to my wonderful students and co-instructors at SUTD and MEET, who showed me how much I enjoy being in a classroom; and the CSAIL community, which never ceases to amaze me.

Finally, this thesis would not have been possible without the gracious support of various institutions. Therefore, I would like to acknowledge the initial financial support provided by the Fulbright S&T Award, and the various funding agencies throughout my Ph.D.: the Boeing Company, the Air Force Office of Scientific Research, the SUTD-MIT Graduate Fellows Program, and the Mitsubishi Electric Corporation.

Contents

List of Figures	13
List of Tables	23
1 Introduction	25
1.1 Minimal <i>vs</i> bounded risk: key thesis principles	30
1.2 Desiderata	33
1.3 Problem statement	35
1.4 Approach in a nutshell	39
1.5 Thesis contributions	46
1.6 Thesis roadmap	47
2 Related work	51
2.1 (Constrained) MDP's and POMDP's	52
2.2 Scheduling under uncertainty	56
2.3 Temporal and hybrid planning	60
2.4 Programming languages for autonomy	65
3 Generating chance-constrained, conditional plans	69
3.1 Introduction	70
3.2 Problem formulation	72
3.2.1 Managing belief states	72
3.2.2 Computing mission risk dynamically	75
3.2.3 Chance-constrained POMDP's	80

3.2.4	Enforcing safe behavior at all times	82
3.3	Relation to constrained POMDP's	83
3.4	Solving CC-POMDP's through RAO*	86
3.4.1	Propagating risk bounds forward	87
3.4.2	Algorithm	91
3.4.3	Grounded example	94
3.4.4	Properties	100
3.5	Experiments	102
3.6	Conclusions	105
4	Programming risk-aware missions with cRMPL	107
4.1	Introduction	108
4.2	Motivation: programming high level missions	110
4.3	Design desiderata for cRMPL	113
4.4	Syntax	115
4.4.1	Episodes	115
4.4.2	Episode constraints	118
4.4.3	Composing episodes in cRMPL	120
4.5	Execution semantics	124
4.5.1	Valid executions of a cRMPL program	125
4.5.2	Execution of cRMPL programs as CC-POMDP	133
4.6	Conclusions	144
5	Risk-sensitive unconditional scheduling under uncertainty	147
5.1	Introduction	148
5.1.1	Motivation: planetary rover coordination	150
5.2	Background & PSTNU's	151
5.3	Problem formulation	154
5.3.1	Computing strong schedules	155
5.3.2	Computing scheduling risk	156
5.4	Polynomial-time, risk-aware scheduling	157

5.4.1	Assumptions and walk-through	157
5.4.2	A linear scheduling risk bound	159
5.4.3	The risk of “squeezing” contingent durations	160
5.4.4	Improving piecewise approximations	163
5.4.5	From minimum risk to other linear objectives	165
5.4.6	Algorithm properties	167
5.5	Experiments	168
5.6	Conclusions	171
6	Risk-sensitive scheduling of PTPN’s	175
6.1	Introduction	176
6.2	Approach in a nutshell	178
6.3	Problem statement	181
6.4	Chance-constrained consistency of PTPN’s	183
6.4.1	Chance-constrained weak consistency	184
6.4.2	Chance-constrained strong consistency	185
6.5	Numerical chance constraint evaluation	189
6.6	Conclusions	192
7	Integrated CLARK experiments	193
7.1	The CLARK system	194
7.1.1	Inputs	194
7.1.2	Outputs	202
7.1.3	Execution on Enterprise	204
7.1.4	Chance-constrained path planning	207
7.2	Collaborative manufacturing	213
7.3	Data retrieval missions	223
7.3.1	Extending RSS with risk-bounded path planning	234
7.4	Conclusions	237

8	Conclusions	239
8.1	Summary of contributions	239
8.2	Future work	241
A	Extending RAO* to partially-enumerated beliefs and policies	247
A.1	Partial enumeration of belief states	248
A.1.1	Predicting PEBS's	250
A.1.2	Updating partially-enumerated beliefs	252
A.1.3	Computing approximate execution risks	255
A.1.4	Approximate forward-propagation of execution risks	260
A.1.5	Computing approximate utilities	260
A.2	Trial-based bounds	262
B	A SAT model for power supply restoration	267
B.1	Modeling circuit-breakers	268
B.2	Simulating the network	270
C	PTPN XML schema	271
D	RSS model	283
	Bibliography	289

List of Figures

1-1	Curiosity’s self-portrait on Mars. Photo source: http://photojournal.jpl.nasa.gov/jpeg/PIA16239.jpg	26
1-2	Examples of underwater vehicles operated by WHOI. Photo 1-2a by Ben Allsup, Teledyne Webb Research. Photos 1-2b and 1-2c by WHOI’s Autonomous Underwater Vehicle Application Center.	26
1-3	Current safety-critical robotic applications. Source for photo 1-3a: https://www.google.com/selfdrivingcar/where/ . Photo 1-3b by John F. Williams, U.S. Navy.	27
1-4	Rural power supply network from [Thiébaux and Cordier, 2001].	28
1-5	Collaborative manufacturing cell in the MERS group at MIT.	29
1-6	PSTNU used in the <i>sleep maximization</i> problem. Circles represent temporal events that are under our control, while squares are temporal events whose occurrence is governed by external forces (“Nature”). A solid arrow with an interval $[l, u]$ represents the constraint $l \leq e_j - e_i \leq u$, where e_j and e_i are, respectively, the events at the end and at the start of the arrow. A dashed arrow represents an uncontrollable duration between two events. The unit of time is minutes here.	31
1-7	A risk-minimal solution to the sleep maximization problem in Figure 1-6. If one sleeps for 420 minutes and leaves to work at 7:30 AM, the risk of being late is no greater than 0.00068%.	32

1-8	A chance-constrained solution to the sleep maximization problem in Figure 1-6. If one sleeps for 444 minutes and leaves to work at 7:54 AM, the risk of being late is no greater than 1.78%.	32
1-9	Architecture diagram of the different elements composing CLARK. Inputs are shown to the left of CLARK’s core (surrounded by the dotted line), while outputs are placed on the right.	39
1-10	Depiction of how the <i>CLARK executive</i> , a combination of CLARK and Pike, can be used in closed-loop control applications. The conditional temporal policy generated by CLARK is sent to Pike in the form of a Probabilistic Temporal Plan Network (PTPN), which Pikes then takes care of dispatching through the physical hardware interface while performing execution monitoring.	41
1-11	Simple morning commute problem from [Santana and Williams, 2014] written in cRMPL.	42
1-12	Probabilistic Temporal Plan Network obtained from the cRMPL program shown in Figure 1-11. Temporal constraints are represented as in Section 1.1; double circles with solid lines represent controllable choices (decisions) that the agent can make (in this case, the means of transportation); and double circles with dashed lines represent uncontrollable choices (observations) obtained from the environment.	43
1-13	Temporal network entailed by choosing to ride a bike to work. A consistent schedule exists with probability 94.9% according to the cRMPL program in Figure 1-11, corresponding to the probability of not slipping and falling.	44
1-14	Temporal network entailed by choosing to drive to work. A consistent schedule exists with probability 98.7% according to the cRMPL program in Figure 1-11, corresponding to the probability of not being involved in an accident.	45
1-15	Temporal network entailed by choosing to stay at home and telecommuting. This option has a consistent schedule with probability 100%.	45

1-16	Block diagram illustrating the relationship between different components developed in this thesis, its goal (blue block on second level from the top), and the real-world need that motivates it (top block).	48
3-1	Depiction of a discrete belief state over a state space \mathcal{S}	73
3-2	Simple graphical example of how safe belief states and observation probabilities are computed. Black circles represent belief states, and squares with probabilities shown next to them are belief state particles. Arrows emanating from particles represent stochastic transitions triggered by action $\pi(b_k)$ at the belief state b_k . Particles shown in red are those whose states s violate constraints, i.e., $c_v(s, C) = 1$, while white particles are <i>safe</i> . For this example, assume that the observation o_{k+1} shown in blue is generated with probability 1 by the particles next to it, and with probability 0 everywhere else.	79
3-3	Modeling chance constraints via unit costs may yield incorrect results when constraint-violating states (dashed outline) are not terminal. Numbers within states are constraint violation probabilities. Numbers over arrows are probabilities for a non-deterministic action.	83
3-4	Impact on belief states of assuming terminal constraint violations, with squares representing belief state particles. White squares represent particles for which c_v indicates no constraint violation, while red squares denote particles on constraint-violating paths.	84
3-5	Relationship between spaces explored by heuristic forward search. . .	86
3-6	Hypergraph node containing a belief state.	87
3-7	Segment of an AND-OR search tree.	87
3-8	Hypergraph representation of an AND-OR tree.	88
3-9	Visual relationship between (3.30) and the portion of RAO*'s search hypergraph associated with executing action $a_k = \pi(b_k)$ at belief b_k . .	90

3-10	From left to right: node in g , the greedy graph; node in G , the explicit graph, but not in g ; node with $r_b = 1$ (guaranteed to violate constraints); color used to represent heuristic estimates. In opposition to nodes with red outlines, we assume in this particular example that nodes with black outlines have $r_b = 0$	95
3-11	Initial state of the search for a CC-POMDP policy featuring a chance constraint $er(b_0, C \pi) \leq \Delta = 5\%$	95
3-12	State of the search after expanding the initial belief state and propagating execution risk bounds forward.	96
3-13	Outcome of updating the policy graph based on the numbers shown in Figure 3-12. The highlighted portion of the explicit graph G corresponds to the best available estimate of the optimal policy graph g	97
3-14	Result of expanding node b_1^2 in Figure 3-13. Notice the child node with red outline and $r_b = 1$, which causes the left hyperedge of b_1^2 to be pruned on the grounds of being too risky.	97
3-15	Result of pruning the search graph in Figure 3-14 due to violations of execution risk bounds.	99
3-16	Estimate of the best policy g after two full iterations of RAO* on this simple example.	99
4-1	Mars rover scenario where a robotic scout must explore different regions of the map before driving back to a relay location and communicating with an orbiting satellite.	110
4-2	Simple rover control program expressed in cRMPL.	112
4-3	Example route between two arbitrary locations A and B on a map. The intermediate dots connecting path segments are intermediate waypoints that the robot should visit in its traversal in order to maintain a safe distance from obstacles.	113

4-4	Result of the CLARK executive dispatching the cRMPL program in Figure 4-2.	114
4-5	Extended Backus-Naur Form (EBNF) grammar for cRMPL.	116
4-6	Episode specifying that an unmanned aerial vehicle (UAV) should scan an area for a period between 1 and 10 time units, while making sure that it maintains itself in a healthy state through the state constraint <code>Healthy=True</code> . If <code>uav-scan</code> can be directly executed by the UAV, this would be a primitive episode. Otherwise, if <code>uav-scan</code> requires a combination of more fundamental episodes, then this episode would be composite.	117
4-7	Composite episode generated by the <code>sequence</code> operator, which enforces sequential temporal execution by means of $[0, \infty]$ STC's.	121
4-8	Composite episode generated by the <code>parallel</code> operator. Different from <code>sequence</code> , component episodes in a parallel composition can be scheduled to happen at the same time.	122
4-9	Composite episodes generated by two instances of the <code>choose</code> operator. The composite episode on the left corresponds to a controllable choice (decision) u_d , while the one on the right corresponds to an uncontrollable choice (observation) u_o	123
4-11	Roller coaster-riding scenario described in cRMPL.	134
4-12	Incremental unraveling of the cRMPL program from Figure 4-11. Temporal consistency is checked by PARIS, the probabilistic scheduling algorithm described in Chapter 5.	135
4-13	Fully unraveled PTPN for the cRMPL program in Figure 4-11 when the number of loop iterations is limited to be no more than 5.	140
4-14	Elements of the constraint store at b_0 , the initial (deterministic) belief state for the execution of the cRMPL program in Figure 4-11.	141
4-15	Temporal constraints tested for consistency with PARIS after the first (<code>ride</code>) primitive is unraveled.	142

4-16	Temporal constraints tested for consistency with PARIS after the second (ride) primitive is unraveled.	142
4-17	Temporal constraints tested for consistency with PARIS after the third (ride) primitive is unraveled. Unlike the previous cases, here PARIS returns that no strongly consistent schedule exists.	143
4-18	Temporal constraints tested for consistency at the halting state in the RAO* policy shown in Figure 4-12. It corresponds to two runs of the loop, followed by the decision to stop the iteration. As required by valid cRMPL executions in Definition 4.4, these constraints are jointly feasible with high probability (100%, in this example).	143
5-1	Rover coordination under temporal uncertainty. (a) Scenario representation as PSTNU. (b) Strong activity schedule for the PSTNU in (a) with scheduling risk bound of 6.7% (i.e., all temporal requirements met with probability of at least 93.3%).	150
5-2	Elements of a PSTNU, where $[l, u]$ is a given interval and f is a known probability density function (pdf). From left to right: controllable event; contingent (uncontrollable) event; Simple Temporal Constraint (STC); STC with Uncertainty (STCU); Probabilistic STC (PSTC).	152
5-3	Piecewise-constant approximation of a Gaussian pdf allowing $\Phi(l_i)$ and $(1 - \Phi(u_i))$ in (5.6) to be upper bounded by a piecewise-linear function. The p_i 's are given partition points.	162
5-4	Performance of Rubato and PARIS on CAR-SHARING dataset.	173
5-5	Performance of PARIS on ROVERS dataset.	174
6-2	Partition of the constraints induced by an assignment to the controllable choices.	186
6-3	Average time to solution for CDA* <i>versus</i> CS.	191
6-4	Average time complexity growth for CDA*.	191

7-2	Simple scenario that can be modeled as a CC-POMDP: a robot with unreliable movements and noisy position sensors that must move around a grid to get to its goal G . Execution should be carried with bounded risk of colliding against obstacles (black squares).	196
7-3	Durative action in PDDL2.1 representing a traversal between two locations by a rover.	200
7-4	Grounded version of the durative action from Figure 7-3 written in cRMPL. Unlike PDDL, the cRMPL version supports probabilistic uncontrollable durations.	201
7-5	Depiction of the episode generated by the cRMPL code in Figure 7-4.	202
7-6	Example of a PTPN obtained from a CC-POMDP policy.	204
7-8	CLARK executive as part of Enterprise. The diagram is a courtesy of Catharine McGhan and Tiago Vaquero.	206
7-9	A chance-constrained traversal generated by pSulu for a robot with linear dynamics and Gaussian noise. The 3σ ellipses are shown in red.	208
7-10	Process of converting traversals generated by pSulu into chance-constrained cRMPL episodes.	210
7-11	First two waypoint episodes for the traversal in Figure 7-9.	210
7-12	Solution quality as a function of the chance constraint Δ for traversals between the two locations in Figure 7-9. The ProOFFull model contains the complete set of constraints for collision avoidance in pSulu, while ProOFFCSA is an approximation of ProOFFull that can be computed faster.	211
7-13	A closer look at the low risk portion of Figure 7-12.	212
7-14	Baxter and pieces of a mock electronic component assembly task constituting the <i>ECA scenario</i> for collaborative manufacturing demonstrations. Small boxes represent electronic components; the elongated yellow box is a circuit board cleaner; and the elongated green box is a soldering iron.	214

7-15	Simple collaborative pick-and-place task between the Baxter and a human coworker.	215
7-16	Top level function defining the cRMPL control program for the collaborative pick-and-place task.	216
7-17	Recursive function modeling the process of the Baxter observing whether the human modified the environment, choosing the next block to move, and repeating the process until no more blocks are left.	216
7-18	The <code>say</code> activity is primitive for the Baxter, and causes the string passed as an argument to be read by a text-to-speech module. The <code>pick_and_place_block</code> activity, on the other hand, is represented by a composite <code>sequence</code> episode composed of the primitive activities <code>pick</code> and <code>place</code>	217
7-19	PTPN for the collaborative pick-and-place task featuring two blocks.	218
7-20	Scheduling risk and time to compute a temporally feasible execution of the collaborative pick-and-place task with 3 blocks, as a function of the width of the time window.	222
7-21	Resilient Spacecraft Executive (RSE) architecture. In the RSS demonstration, the role of the risk-aware <i>deliberative layer</i> was performed by the CLARK executive within Enterprise (Figure 7-8). The diagram is a courtesy of Catharine McGhan and Tiago Vaquero.	224
7-22	Snapshot of the RSS demonstration environment.	225
7-23	Different stage of the RSS demonstration available at the aforementioned video link. The “nominal temporal plan” shown in 7-23a is the same one depicted on the bottom right quadrant of Figure 7-22. The figures are a courtesy of Tiago Vaquero and Catharine McGhan. . . .	228
7-24	PTPN generated by CLARK and sent to Pike for the simple case where only a picture at location 3 is needed. The actual scheduling of activities is performed by Pike in real time as it dispatches the plan. . . .	230

7-25	CLARK policy for the RSS scenario with picture requests at locations 3 and 5, and two types of move actions with bounded risk of collision: move-low (risk bound of 0.01%) and move-high (risk bound of 0.04%).	235
7-26	Comparison between RSS scenarios with (label <i>Collision</i>) and without (label <i>No collision</i>) collision handling. The order of goal combinations on the horizontal axes follows the first column of Table 7.6.	236
A-1	Example PEBS. Squares on the tree represent enumerated particles with non-zero probabilities; clouds represent unenumerated (lumped) particles; and layers represent belief states.	248
A-2	Situation in which an interval of width $2t$ around the sample mean \bar{X} does not contain the true mean μ	264
A-3	Sensitivity of t with respect to n for a constant term multiplying $1/(n\sqrt{n})$ equals to -1 . The more negative the value, the better. . . .	265
A-4	Ambiguity for selecting between actions a_1 and a_2 , which generated, respectively, the sample means \bar{X}_1 and \bar{X}_2	266
B-1	Small instance of a power network with fault sensors.	268
B-2	Electrical model of a feeder.	269

List of Tables

3.1	SA results for various time windows and risk levels. The <i>Window</i> column refers to the time window for the SA agent to gather information, not a runtime limit for RAO*.	105
3.2	PSR results for various numbers of faults (#) and risk levels. Top: avg. of 7 single faults. Middle: avg. of 3 double faults. Bottom: avg. of 2 triple faults. Left (right) numbers correspond to 12 (16) network sensors.	105
4.1	Relationship between the pRMPL variant of [Effinger, 2012], cRMPL, and HCA [Williams et al., 2003].	126
4.2	Relationship between cRMPL and PTPN constructs.	131
7.1	Number of elements involved in the scheduling of the collaborative pick-and-place task as a function of the number of blocks. <i>Decisions</i> are controllable choices; <i>observations</i> are probabilistic uncontrollable choices; CD are controllable durations represented as simple temporal constraints; and UD are uncontrollable probabilistic durations.	217
7.2	Number of elements in the PTPN's given to Pike, which contain no decisions (controllable choices) and only simple temporal constraints (controllable durations). Refer to the numbers in Table 7.1 for a comparison of relative complexity.	219

7.3	Performance comparison between CLARK and Pike in the collaborative pick-and-place scenario. The complexity of the scheduling problems for CLARK and Pike are described, respectively, in Tables 7.1 and 7.2. The CD columns are used for cRMPL programs containing only controllable durations, while UD columns are for programs containing both controllable and uncontrollable (probabilistic) durations. We use <i>NA</i> to denote that Pike cannot handle uncontrollable durations, and <i>TO</i> to represent a compilation timeout (ran beyond 1 hour without returning a result). Numbers are averages over ten runs.	220
7.4	Duration models used in the RSS demonstration. Set-bounded durations $u[a, b]$ are those found in STNU's and represent random variables that take values within the interval $[a, b]$ with <i>unknown</i> probability distribution. Similar to set-bounded durations, a uniform duration $U(a, b)$ also takes values in the interval $[a, b]$, but with known density $(b - a)^{-1}$ anywhere within the interval. Finally, a Gaussian duration $N(\mu, \sigma^2)$ has mean μ and variance σ^2	227
7.5	Possible schedule for the risk-aware "nominal" temporal plan generated by CLARK for the RSS demonstration scenario.	229
7.6	CLARK's performance on the RSS demonstration for different combinations of goals. PL is the <i>Plan Length</i> (number of actions); TTS is the <i>Time to Solution</i> , i.e., the amount of time to generate a temporal plan with a schedule that meets the chance constraint; ER is RAO* <i>execution risk</i> , which corresponds to the scheduling risk for these plans; MR-MS is the <i>Minimum Risk Makespan</i> , i.e., the temporal span to the temporal plan when scheduling risk is minimized; and CC-MS is the <i>Chance-Constrained Makespan</i> , where the risk bound $\Delta = 0.1\%$ is exploited to reduce the total execution time.	232

Chapter 1

Introduction

“Space: the final frontier. These are the voyages of the starship Enterprise. Its five-year mission: to explore strange new worlds, to seek out new life and new civilizations, to boldly go where no man has gone before.”

Captain Kirk, Starship Enterprise

We hold these truths to be self-evident: warp drives would be amazingly useful tools to have at our disposal; and it is remarkably difficult to prevent our delicate human bodies from freezing, burning, choking, starving, disintegrating, or some other painful combination thereof as we try to visit increasingly distant corners of our galaxy. Unfortunately, by the time of this thesis’ writing, humankind had not yet found acceptable solutions to any of these problems, thus forcing us earthlings to rely on robotic proxies to physically explore the immensity of space beyond our blue planet’s immediate vicinity.

When scientists envision robotics as a means to conquer the skies, a fundamental question comes to mind: “can we trust these robots to do the right thing?” As we move towards trusting robots and other autonomous systems with increasingly important missions, one must be able to provide a decisive answer to this question. Evidently, a thorough discussion of “do the right thing” would require us to sift through centuries of human lucubrations on moral and ethics, a journey as wonderful as it is out of the

scope of this thesis. Therefore, here, we circumscribe ourselves to the much narrower field of applications in which “do the right thing” means “complete their tasks well and safely”. In particular, this thesis delivers a series of methods that have been demonstrated to enable autonomous agents to plan missions and its contingencies for stochastic outcomes, while guaranteeing user-specified levels of risk. We justify this thesis focus below.



Figure 1-1: Curiosity’s self-portrait on Mars. Photo source: <http://photojournal.jpl.nasa.gov/jpeg/PIA16239.jpg>.

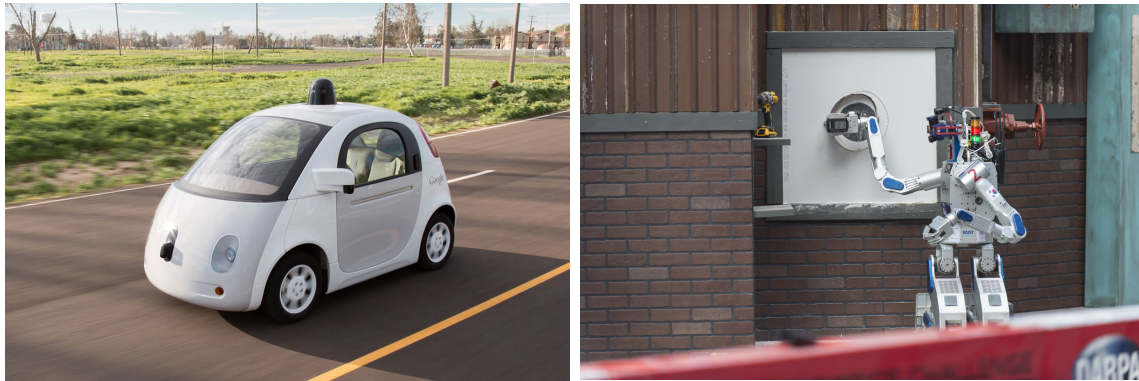


(a) Slocum glider. (b) Sentry autonomous underwater vehicle. (c) Nereus remotely-operated vehicle.

Figure 1-2: Examples of underwater vehicles operated by WHOI. Photo 1-2a by Ben Allsup, Teledyne Webb Research. Photos 1-2b and 1-2c by WHOI’s Autonomous Underwater Vehicle Application Center.

Fortunately, one does not necessarily have to go to outer space, as in the robotic

space missions operated by NASA’s Jet Propulsion Laboratory (JPL) (Figure 1-1), to find situations in which autonomous agents must be trusted with safety-critical missions. Within our own planet, one could mention, for instance, the sea exploration missions undertaken by the Woods Hole Oceanographic Institution (WHOI) and its suite of autonomous and remotely-operated underwater vehicles, some of which are shown in Figure 1-2. Closer to our daily lives, there has been a tremendous push in recent years by traditional automakers, Silicon Valley giants, universities, and start-ups in the United States and abroad towards the development of semi and fully autonomous vehicles to be operated on roads alongside human drivers (Figure 1-3a). In disaster relief and search-and-rescue domains, where safety is also of paramount importance, one could mention the recently completed DARPA Robotics Challenge (Figure 1-3b) and automated power supply restoration (Figure 1-4).



(a) One of Google’s self-driving vehicles.

(b) HUBO, from the Korea Advanced Institute of Science and Technology (KAIST), winner of the DARPA Robotics Challenge.

Figure 1-3: Current safety-critical robotic applications. Source for photo 1-3a: <https://www.google.com/selfdrivingcar/where/>. Photo 1-3b by John F. Williams, U.S. Navy.

Finally, outside the streets and inside modern factories and warehouses, one can observe a clear progression from the traditional manufacturing cell model, in which humans and robots work in complete physical separation, to a more dynamic, collaborative production environment in which teams of humans and robots must coordinate amongst themselves in order to achieve common goals. As indication of such a trend, consider Amazon’s recent purchase of Kiva System’s robotic warehouse management

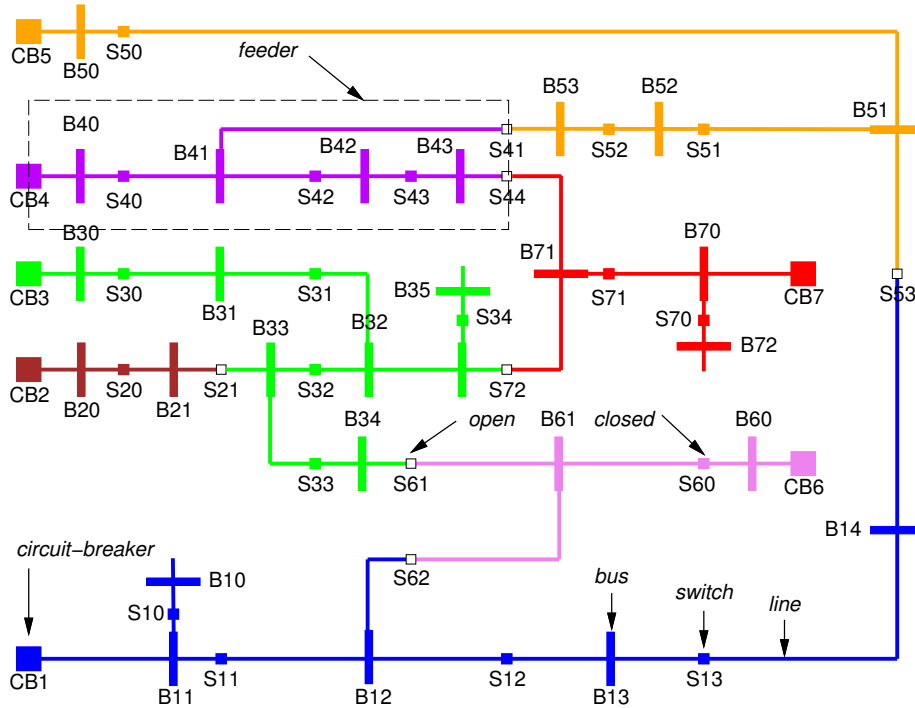


Figure 1-4: Rural power supply network from [Thiébaux and Cordier, 2001].

technology and sponsorship of the Amazon picking challenge; and the Boeing Company’s multi-year alliance with MIT to allow robots to be seamlessly integrated within their intrinsically unstructured and fluid aircraft manufacturing environments (Figure 1-5).

In this thesis, we assert that trusting autonomous agents with real-world, high-stakes missions requires that these agents develop a keen sensitivity to risk and incorporate uncertainty into their decision-making. By risk, we mean the commonly accepted notion of the probability of some failure event taking place, such as a mobile agent crashing against obstacles, missing communication windows, running out of battery, crossing no-fly zones, etc. Due to the lack of guarantees that plans will be carried out within their very stringent safety requirements, the current practice for ensuring mission safety generally requires groups of engineers to reason over a very large number of potential decisions and scenarios that might unfold during execution, which is a challenging, time-consuming, and error-prone process. Given the overwhelming number of possible scenarios, one opts, in many cases, to follow the

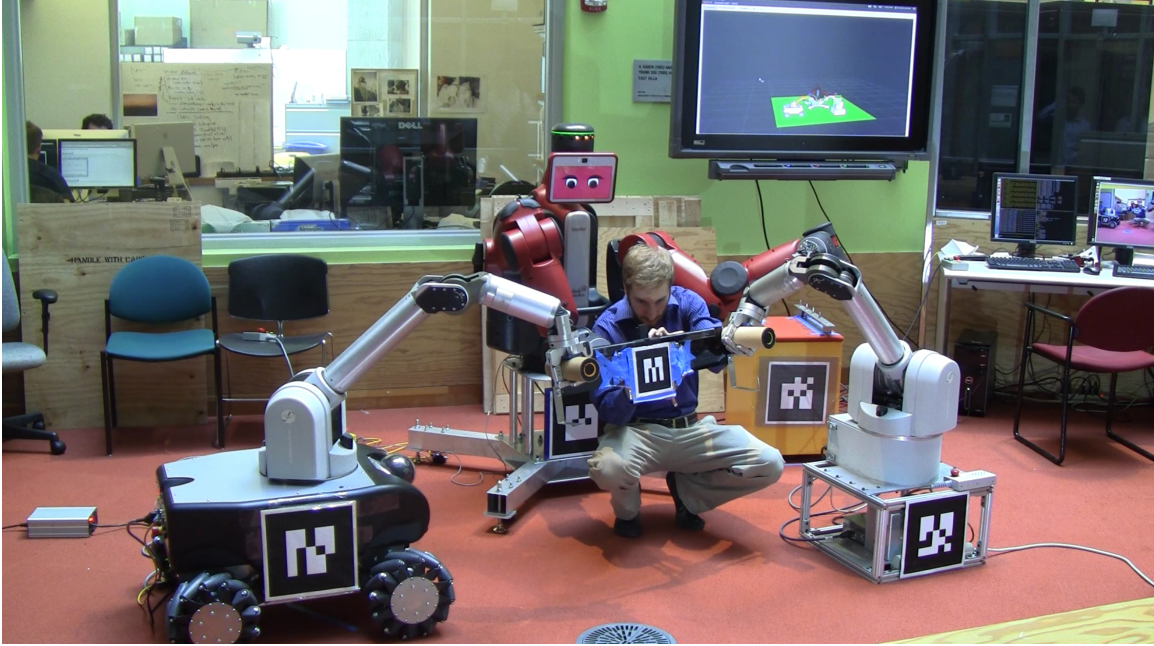


Figure 1-5: Collaborative manufacturing cell in the MERS group at MIT.

“safest”, most predictable strategy, in which the impact of uncertainty in the plan is limited. Such “safe” fixed sequences of actions, however, tend to be far from ideal in terms of utility or brittle to disturbances due to their inability to adapt to the environment. For example, the authors in [Benazera et al., 2005a] report that “it has been estimated that the 1997 Mars Pathfinder rover spent between 40% and 75% of its time doing nothing because plans did not execute as expected”, a clear underutilization of such an invaluable resource. Extensions of this planning paradigm have been proposed in order to improve robustness, such as conditioning execution on the state of the world and dynamic task scheduling. However, mission operators resist to incorporate those improvements, due to a lack of explicit guarantees on the risk of mission failure.

Suppose we have a meaningful way of measuring “risk of failure” for an agent executing a plan dynamically in an uncertain environment, a concept developed in Chapter 3. The next question is what the agent should do with this new quantity. By means of a simple example of scheduling under uncertainty, a subproblem within the risk-aware conditional temporal planning theme of this thesis, the next section provides insights on the principles behind our choice for risk-bounded autonomy.

Throughout this thesis, we leverage *chance constraints* [Birge and Louveaux, 1997] to impose risk bounds on a mission.

1.1 Minimal *vs* bounded risk: key thesis principles

Risk-aware decision-making is not privy to autonomous agents. On the contrary, handling different notions and levels of “risk” is an integral part of everyone’s daily routines. A recurring concept in this thesis is the notion of *risk-bounded* plan execution [Blackmore, 2007, Undurti, 2011, Ono et al., 2012a], and its contrast with previous risk-minimal strategies [Effinger, 2012]. Therefore, in an effort to develop an intuitive understanding of this distinction, this section brings a simple *sleep maximization* problem, a pedagogical example of risk-aware scheduling that seeks to provide a convincing answer to the question: “why should I opt for a higher risk solution to a problem, should a lower risk one exist?” Its description follows:

“I go to bed at midnight and must sleep for at least 5 hours (300 minutes) every night. Once I wake up, it takes me at least 30 minutes to get ready to leave home and go to work. Historically, the duration of my commute can be well approximated by a Gaussian random variable with mean of 45 minutes and standard deviation of 10 minutes. I would like to sleep as much as possible, but still make sure I arrive at work by 9 AM with high certainty. ”

Figure 1-6 shows the Probabilistic Simple Temporal Network with Uncertainty (PSTNU) used for solving the *sleep maximization* problem: pick times to wake up (event e_1) and leave to work (event e_2) so that the sleeping period is maximized, while making sure to arrive at work (or school) on time. Even though PSTNU’s and risk-aware scheduling are formally introduced only in Chapter 5, the example in Figure 1-6 is simple enough to allow us to quickly understand the reasoning process.

The first thing we should notice about Figure 1-6 is that *no risk-free solution to the sleep maximization problem exists!* The reason for that is simple: even if we

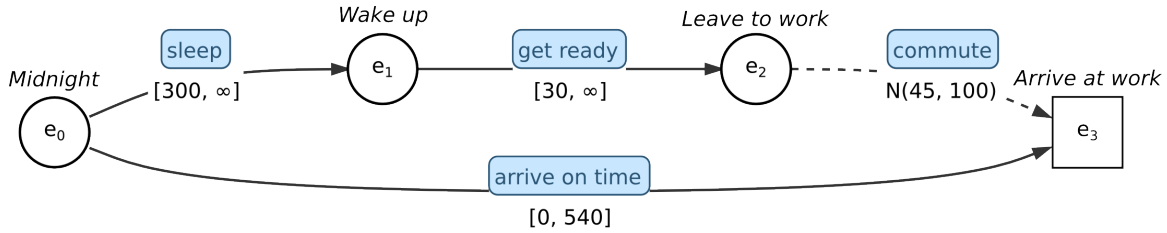


Figure 1-6: PSTNU used in the *sleep maximization* problem. Circles represent temporal events that are under our control, while squares are temporal events whose occurrence is governed by external forces (“Nature”). A solid arrow with an interval $[l, u]$ represents the constraint $l \leq e_j - e_i \leq u$, where e_j and e_i are, respectively, the events at the end and at the start of the arrow. A dashed arrow represents an uncontrollable duration between two events. The unit of time is minutes here.

choose to wake up at 5:00 AM ($e_1 = 300$) and leave to work at 5:30 AM ($e_2 = 330$), there is negligible, albeit nonzero, probability that the duration of the commute will be greater than 210 minutes according to the Gaussian duration model, therefore causing us to be late for work. This is a rather trivial example of a more general notion that pervades the algorithms in this thesis:

Principle 1.1. *When designing dynamic temporal plans to be executed by autonomous agents in real-world, uncertain environments, allowing for nonzero risk of failure is often required to be able to generate any solution at all.*

With Principle 1.1 in mind, let us make the following approximation in the sleep maximization problem: instead of considering the complete, unbounded support of the commute duration model, let us focus on the restricted $\pm 4.5\sigma$ interval around the mean, i.e., the interval $[0, 90]$. Assuming that the Gaussian duration model was, indeed, a good approximation¹ of our morning commute’s uncertain duration, such an assumption would exclude 0.00068% of the total probability mass. If we agree that the risk yielded by such an approximation is justifiable, we arrive at the second important principle motivating the algorithms in this thesis:

Principle 1.2. *Allowing autonomous agents to execute plans with nonzero risk of failure allows us to focus our attention (i.e., computation) on the most likely scenarios*

¹It most certainly cannot be exact, given that the negative support of a Gaussian random variable cannot represent the duration of any physical process.

for the task at hand.

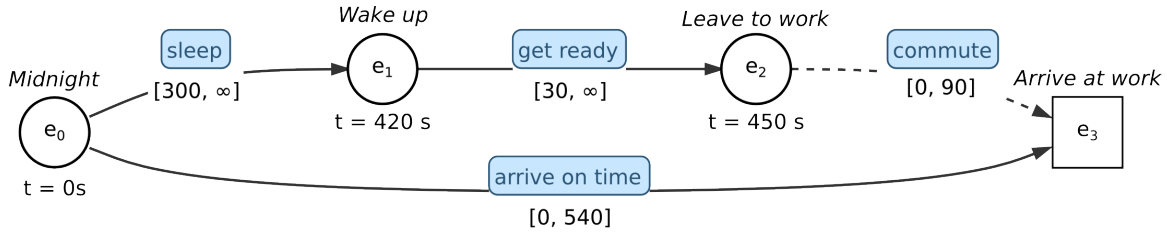


Figure 1-7: A risk-minimal solution to the sleep maximization problem in Figure 1-6. If one sleeps for 420 minutes and leaves to work at 7:30 AM, the risk of being late is no greater than 0.00068%.

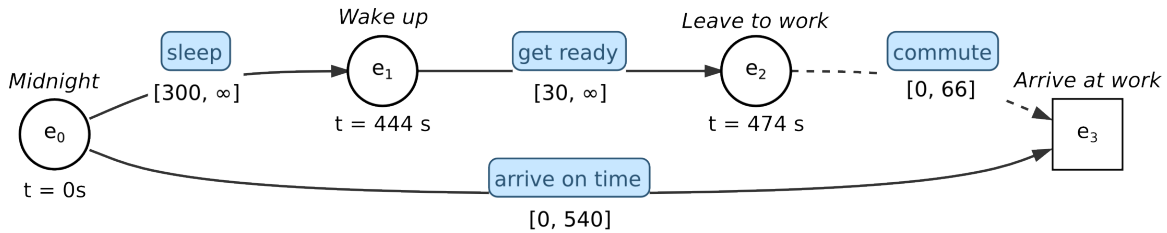


Figure 1-8: A chance-constrained solution to the sleep maximization problem in Figure 1-6. If one sleeps for 444 minutes and leaves to work at 7:54 AM, the risk of being late is no greater than 1.78%.

With this approximation of the commute model, we are ready to go back to our original goal of developing an intuitive understanding of risk-minimal *vs* risk-bounded strategies. In Figure 1-7, we see what a risk-minimal solution to the sleep maximization problem looks like: instead of prolonging our sleep, this solution forces us to sleep as little as possible and be ready to leave to work by 7:30 AM, so that we can accommodate the worst-case scenario where our commute takes 1:30 hours (90 minutes). On the other hand, Figure 1-8 shows what our schedule would look like if we allowed our risk of arriving late to work to be no higher than 2%: instead of getting up as early as possible in the fear that the worst commute scenario might happen every day, we allow ourselves to sleep for 24 additional minutes every morning and leave to work at 7:54 AM. For a commute that is expected to take 45 minutes, the latter strategy would have us arrive at work, on average, at 8:39 AM (instead of 8:15 AM), and is arguably much closer to what we would expect the average person to do. Once again, despite the simplicity of this example, the comparison between

the risk-minimal and the risk-bounded solutions in Figures 1-7 and 1-8 is effective at communicating another important algorithm design principle in this thesis:

Principle 1.3. *While risk-minimal (including risk-free) execution strategies tend to be overly conservative and yield poor performance, risk-bounded, also referred to as chance-constrained, strategies allow autonomous agents to operate strictly within the bounds of acceptable safety while performing at the best of their capacity.*

1.2 Desiderata

Recalling the principles and motivating scenarios from previous sections, we argue that a path towards risk-aware autonomy for safety-critical applications must go through the endowment of autonomous agents with the ability to

I Handle state uncertainty and its impact on risk: when autonomous agents leave controlled laboratory environments and meet the real world, it is rarely, if ever, the case that decisions will be made under complete information about the agent's state and the state of its environment. As examples, consider the planetary and underwater robotic explorers described before, or an automated medical advice system that must suggest a series of treatments for a patient with an unknown ailment based on results of imperfect exams. In those and many other applications, one cannot expect more than the availability of a *belief state*, or some approximation thereof, that captures our uncertainty about the true underlying state of the system. At the same time, the impact of this uncertainty on mission risk should be properly quantified and handled dynamically during execution. For instance, an autonomous Mars rover that is unsure about its energy levels should refrain from further exploring the environment until its internal state can be diagnosed due to the risk of damaging its batteries.

II Dynamically leverage real-time (noisy) sensor information: related to the previous point, an autonomous agent operating under uncertainty must constantly adapt its behavior in response to real-time measurements that it receives

from its sensors. Otherwise, safety can only be ensured by following static, and often exceedingly conservative, conformant [Smith and Weld, 1998] strategies that must “fit all cases”. For example, a manufacturing robot that follows a conformant strategy in order not to negatively interfere with its human counterpart would probably settle for the policy of not doing anything at all. On the other hand, if the robot could observe the human through its sensors and use that to infer, up to some level of uncertainty, what were the human’s goals and intended next steps, it could dynamically adapt its behavior and promote a more synergistic collaboration.

III **Provide strict safety guarantees over rich sets of mission constraints:**

consider again the example of a robot that must work in tandem with humans to complete manufacturing tasks. Ensuring safe behavior in this environment requires the robot not only to decide what activities to perform and how to react to their outcomes, but also how it should move its joints so as to not hit obstacles, and schedule activities under temporal uncertainty to avoid violating any temporal deadlines. In order to be able to dynamically manage risk stemming from diverse sets of mission requirements in hybrid (continuous and discrete) domains, autonomous agents must be able to efficiently quantify, or at least estimate, how likely it is that they will find themselves trapped in “dangerous” situations in which one or more safety requirements might get violated, and incorporate these risk estimates into their decision-making.

IV **Allow mission operators to specify risk-aware autonomous behavior**

at a high level of abstraction: skilled humans communicate goals and task requirements at a high level of abstraction, while relying on each other’s reasoning capabilities to “unravel” these high level requirements into sequences of more elementary activities that achieve these goals. For instance, take the example of two people working collaboratively to assemble a piece of furniture: there might be different ways of dividing up the task between them, with each possible method potentially entailing different requirements in terms on how the two

should coordinate in order to have the pieces ready to be put together. Once this high level plan is laid out, it is up to the individuals to devise a course of action that will ensure that they meet their own subgoals while ensuring proper team coordination. Therefore, if we are to have autonomous agents serve one or even both roles in such a scenario, it becomes necessary that the process of communicating the “mission” to them happens at a level of detail that feels natural from a person’s perspective.

In light of these key enabling capabilities for risk-aware autonomy, we are now ready to delve into this thesis’ problem statement.

1.3 Problem statement

Towards the fulfillment of the desiderata, we propose that risk-bounded, dynamic plans (or policies) with sensing actions for safety-critical applications be generated as solutions to Chance-Constrained Partially Observable Markov Decision Process (CC-POMDP) [Santana et al., 2016b] instances, a formalism explained in detail in Chapter 3 that extends traditional POMDP’s [Smallwood and Sondik, 1973] with a notion of probabilistic *execution risk*. In simple words, a CC-POMDP is a model of decision-making under uncertainty in which an autonomous agent seeks to optimize some expected measure of performance, such as minimize expected energy consumption or maximize expected science return, by continuously executing actions in its environment. As it does so, we assume that the agent may also be able to collect potentially noisy or ambiguous observations from its sensors in order to mitigate uncertainty about the environment and its own state. Finally, safety is ensured in a CC-POMDP by means of *conditional state constraints*, which define feasible regions for the state of the underlying system under control (the *Plant*), and their associated *chance constraints*, i.e., user-defined upper bounds on the probability of subsets of these conditional state constraints being violated during execution. We use the term *conditional* when referring to state constraints to make it clear that, in a CC-POMDP, we allow the state constraints that a plan must fulfill to depend on real-time

observations acquired during execution. This is in contrast to only supporting *global* constraints, a particular type of conditional constraint that must hold regardless of the execution scenario. More formally, a CC-POMDP is a tuple

$$H = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R, b_0, \mathcal{C}, c_v, \Delta \rangle,$$

where

- \mathcal{S} , \mathcal{A} , and \mathcal{O} are, respectively, discrete sets of planning states, actions, and observations;
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a stochastic state transition function such that

$$T(s_k, a_k, s_{k+1}) = \Pr(s_{k+1} | s_k, a_k);$$

- $O : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ is a stochastic observation function such that

$$O(s_k, o_k) = \Pr(o_k | s_k);$$

- $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function;
- b_0 is the initial belief state;
- \mathcal{C} is a set of conditional constraints defined over \mathcal{S} ;
- $c_v = [c_v^1, \dots, c_v^q]$ is a vector of constraint violation indicators $c_v^i : \mathcal{S} \times 2^{\mathcal{C}} \rightarrow \{0, 1\}$, $i = 1, 2, \dots, q$, such that $c_v(s, C^i) = 1$ if, and only if, s violates constraints in a subset C^i of \mathcal{C} ;
- $\Delta = [\Delta^1, \dots, \Delta^q]$ is a vector of q execution risk bounds used to define q chance constraints

$$er(b_k, C^i | \pi) \leq \Delta^i, i = 1, 2, \dots, q, k \geq 0, \quad (1.1)$$

where

$$er(b_k, C^i | \pi) = 1 - \Pr \left(\bigwedge_{i=k}^h Sa_i(C^i) \middle| b_k, \pi \right)$$

is the *execution risk* of a policy π ; $Sa_k(C^i)$ (for “safe at step k ”) is a Bernoulli random variable denoting whether the system *has not violated* any constraints in C^i at planning step k ; and h is the planning horizon.

A solution to a CC-POMDP is an optimal *policy* (or *plan*) $\pi^* : \mathcal{B} \rightarrow \mathcal{A}$ mapping *belief states* (or just *beliefs*) in \mathcal{B} to actions in \mathcal{A} such that

$$\pi^* = \arg \max_{\pi} \mathbb{E} \left[\sum_{t=0}^h R(s_t, a_t) \middle| \pi \right]$$

and (1.1) holds. By enforcing (1.1) and conditioning action choices on belief states, which in turn depend on real-time (noisy) observations received by the agent, we see that CC-POMDP policies meet desiderata I and II.

With respect to ensuring safety over rich sets of constraints in desideratum III, CC-POMDP’s incorporate a hierarchical constraint satisfaction structure akin to Planning Modulo Theories (PMT) [Gregory et al., 2012], semantic attachment [Dornhege et al., 2012], and the approach in [Ivankovic et al., 2014], through constraint violation functions c_v that are theory-dependent. As in general SMT-like [Nieuwenhuis et al., 2006] hierarchical problem decompositions, CC-POMDP solution scalability is strongly tied to how efficiently these risk-aware constraint checkers run. Therefore, in close connection with the temporal applications that this thesis is concerned about, Chapters 5 and 6 present the current fastest algorithm for risk-aware scheduling of Probabilistic Simple Temporal Networks (PSTN’s) and PSTN’s with Uncertainty (PSTNU’s), as well as the first algorithms for risk-aware scheduling of Probabilistic Temporal Plan Networks (PTPN’s). The reason why these scheduling algorithms are needed is because this thesis pursues a *continuous time* approach to the generation of temporal plans with duration uncertainty, as opposed to resorting to *time discretization*. Thus, instead of folding discrete increments of time directly into the CC-POMDP’s discrete

state space \mathcal{S} , our models collect within \mathcal{C} temporal requirements (e.g., “this plan has to finish within 10 minutes”), as well as uncertain duration models for durative CC-POMDP actions (e.g. “driving from A to B takes anything between 10 to 20 minutes”), as conditional and unconditional temporal constraints from the scheduling literature, as reviewed in Section 2.2.

In lieu of providing general solution methods for CC-POMDP models, this thesis focuses on a *temporal* subset that is relevant to the risk-aware applications that this thesis is concerned about, in which autonomous agents must complete their tasks under time pressure. More specifically, we consider CC-POMDP models in which actions have non-instantaneous durations - also referred to as *activities* - and where one or more elements of \mathcal{C} force the execution of policy π^* to eventually terminate. This, in turn, causes the planning horizon h to be limited by $\lceil t_{\max}/d_{\min} \rceil$, where t_{\max} is the maximum temporal plan length, and d_{\min} is the shortest activity duration.

Furthermore, even though optimal CC-(PO)MDPs policies may, in general, require some limited amount of randomization [Altman, 1999], we follow [Dolgov and Duffie, 2005] and focus on optimal *deterministic* policies for technical and application-dependent reasons. On the technical side, while deterministic policies can be effectively solved using heuristic forward search (HFS), computing exact randomized policies for Constrained POMDP’s (C-POMDP’s) generally involves intractable formulations over reachable beliefs, and current approximate methods [Kim et al., 2011, Poupart et al., 2015] do not guarantee solution feasibility. In this context, it is worthwhile to mention [Trevizan et al., 2016] as a recent advancement in the constrained MDP literature combining HFS and C-MDP policy optimization in dual space to efficiently compute randomized policies over infinite horizons. On the application side, preference for agent predictability in safety-critical systems causes operators to rarely trust stochastic autonomous behavior.

Finally, for the fulfillment of desideratum IV, we introduce the Chance-constrained Reactive Model-based Programming Language (cRMPL) in Chapter 4, where we show that the problem of extracting optimal execution policies for this decision-theoretic language can also be framed as a CC-POMDP instance.

The reasoning tools developed in this thesis are combined to form CLARK², whose overview is given in the next section. CLARK is a risk-aware conditional planning system that can generate chance-constrained, dynamic temporal plans for autonomous agents that must operate under uncertainty.

1.4 Approach in a nutshell

The different tools and algorithms developed in the context of this thesis are combined to form the **C**onditional **P**lanning for **A**utonomy with **R**isk (CLARK) system, whose block diagram is shown in Figure 1-9.

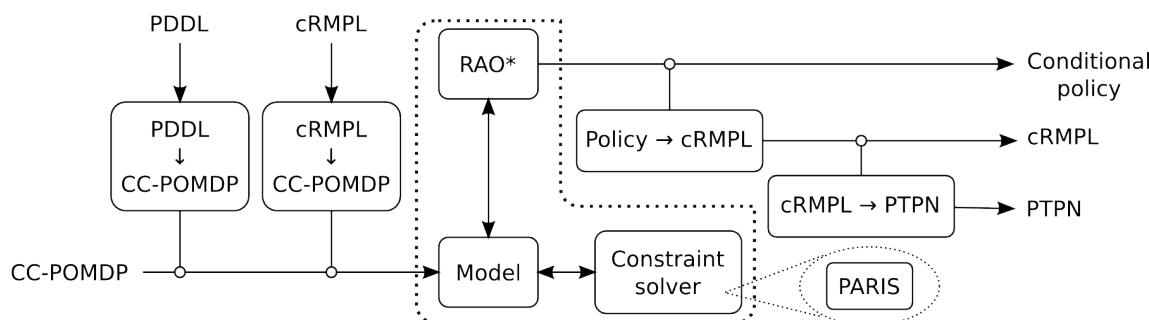


Figure 1-9: Architecture diagram of the different elements composing CLARK. Inputs are shown to the left of CLARK’s core (surrounded by the dotted line), while outputs are placed on the right.

The input to CLARK is always a CC-POMDP, either directly supplied to the system, or generated from some other type of user-provided input. This thesis focuses on the forms of input to CLARK shown in Figure 1-9. In the *generative* conditional planning discussion in Chapter 3, we consider inputs given directly as CC-POMDP’s instances. Alternatively, in the *decision-theoretic* programming setting of Chapter 4 and Section 7.2, the input is a program in the Chance-constrained Reactive Model-based Programming Language (cRMPL), a novel variant of RMPL [Ingham et al., 2001, Williams et al., 2001, Williams and Ingham, 2002, Williams et al., 2003, Ingham,

²This acronym a desperate attempt at evoking the Lewis and Clark Expedition towards the Pacific Ocean, a mission undertaken under strong uncertainty and numerous risks. The fact that Lewis and Clark leveraged Sacagawea’s expert knowledge to navigate uncharted territory also creates an interesting parallel with the heuristic-guided search methods developed in this thesis.

2003, Effinger, 2012] developed in this thesis that adds support to chance-constrained execution. The mapping from optimal cRMPL execution to CC-POMDP depicted in Figure 1-9 is given in Chapter 4. Finally, in the experimental demonstration of CLARK described in Section 7.3, we show how initial planning models in the Planning Domain Definition Language (PDDL) [McDermott et al., 1998] are used to bootstrap CC-POMDP models for a real-world application involving resilient planetary rovers.

The flow of information that maps inputs to outputs within CLARK starts with the CC-POMDP instance given as input becoming part of the *Model* in CLARK’s core, shown in Figure 1-9 surrounded by the dotted line. Within the core, the RAO* algorithm (Chapter 3) accesses the different functions available as part of the CC-POMDP model to incrementally construct an optimal deterministic conditional plan with bounded risk. It is important to stress the terms *conditional* and *bounded risk*. By being conditional, the plan allows the autonomous agent to adapt to its environment in real-time by conditioning the choice of activity to be executed on its current level of knowledge, or belief, about the true state of world. This belief state is, in turn, a function of the history of observations gathered by the agent from the environment according to the CC-POMDP’s observation model. With respect to *bounded risk*, it refers to the fact that the execution of such optimal conditional policies by the autonomous agent is guaranteed to keep the agent safe by ensuring that “good behavior”, as defined by a set of conditional state constraints, can be expected from the autonomous agent with high probability. In order to assess the level of safety entailed by such conditional policies, RAO* leverages a potentially diverse suite of constraint solvers to quantify the likelihood of one or more such state constraints being violated during execution. Such calls to the constraint solvers are not done directly by RAO*; instead, they are routed through general risk-measuring interfaces attached to the *Model* block of CLARK. Among these constraint solvers, the PARIS algorithm (Chapter 5) is highlighted in Figure 1-9 due to its key role in efficiently computing risk-aware schedules in the presence of temporal uncertainty.

As previously mentioned, the solution to the input CC-POMDP given to CLARK is an optimal conditional policy that entails a consistent conditional constraint system

with high probability. Then, according to Figure 1-9, this policy can optionally be converted into an output cRMPL program representing the optimal behavior for the autonomous agent (the process for performing this conversion is given in Chapter 7). An advantage of representing CLARK’s outputs as cRMPL programs is the fact that these can later be used as *optimal subroutines* within a hierarchical composition of cRMPL programs, as defined by the cRMPL composition operators presented in Chapter 4.

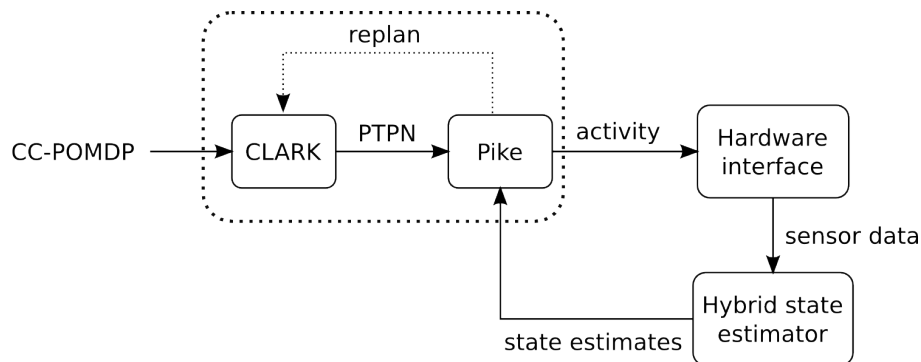


Figure 1-10: Depiction of how the *CLARK executive*, a combination of CLARK and Pike, can be used in closed-loop control applications. The conditional temporal policy generated by CLARK is sent to Pike in the form of a Probabilistic Temporal Plan Network (PTPN), which Pike then takes care of dispatching through the physical hardware interface while performing execution monitoring.

When combined with the *Pike dispatcher* from [Levine and Williams, 2014], the CLARK system from Figure 1-9 becomes the *CLARK executive*, shown in Figure 1-10 surrounded by the dotted line. The input to Pike is a Probabilistic Temporal Plan Network (PTPN) [Levine and Williams, 2014, Santana and Williams, 2014], which Figure 1-9 shows can be obtained from the cRMPL program generated from CLARK’s output. In our goal of developing risk-aware autonomous agents that can quickly react to their environment in order to perform their tasks well and safely, Pike offers yet another line of defense: as it schedules activities and dispatches them through the physical hardware interface, Pike constantly monitors the true state of system in the hopes of detecting failure conditions that may not have been included in the CC-POMDP model used to generate CLARK’s conditional policy. Should such failure conditions be detected, Pike immediately triggers a replanning signal to CLARK using

the current state of the system as the initial belief. The *hybrid state estimator* in Figure 1-10 translates sensor measurements into discrete logical predicates that Pike uses to monitor execution, and has been investigated in this thesis and elsewhere [Timmons, 2013, Santana et al., 2014, Santana et al., 2015, Santana et al., 2016a, Lane, 2016]. However, for the sake of focusing our attention on the core contributions of this thesis, we refer the reader to these references for details on how the hybrid estimation block works, and to the video at <https://www.youtube.com/watch?v=Fz1s5jAgEew> for a hardware demonstration.

```
def rmpyl_icaps14():
    """
    Example from (Santana & Williams, ICAPS14).
    """
    prog = RMPyL()
    prog *= prog.decide(
        {'name': 'transport-choice', 'domain': ['Bike', 'Car', 'Stay'],
         'utility': [100, 70, 0]},
        prog.observe(
            {'name': 'slip', 'domain': [True, False],
             'ctype': 'probabilistic', 'probability': [0.051, 1.0-0.051]},
            prog.sequence(Episode(action='ride-bike'),
                          duration={'ctype': 'controllable', 'lb': 15, 'ub': 25}),
                          Episode(action='change'),
                          duration={'ctype': 'controllable', 'lb': 20, 'ub': 30}),
                          Episode(action='ride-bike', duration={'ctype': 'controllable', 'lb': 15, 'ub': 25})),
            prog.observe(
                {'name': 'accident', 'domain': [True, False],
                 'ctype': 'probabilistic', 'probability': [0.013, 1.0-0.013]},
                prog.sequence(Episode(action='tow-vehicle'),
                              duration={'ctype': 'controllable', 'lb': 30, 'ub': 90}),
                              Episode(action='cab-ride'),
                              duration={'ctype': 'controllable', 'lb': 10, 'ub': 20}),
                              Episode(action='drive'), duration={'ctype': 'controllable', 'lb': 10, 'ub': 20}),
                              Episode(action='stay'))
        )
    prog.add_overall_temporal_constraint(ctype='controllable', lb=0.0, ub=30.0)
    return prog
```

Figure 1-11: Simple morning commute problem from [Santana and Williams, 2014] written in cRMPL.

Seeking to provide an intuitive understanding of CLARK’s inner workings, consider the cRMPL program in Figure 1-11 modeling the simple morning commute problem³ from [Santana and Williams, 2014]. In this problem, a person must decide

³The apparent obsession with commute problems in the examples given so far is intended to show that our tools should be broadly applicable to decision-making under uncertainty, as opposed to being restricted to niche robotics applications.

between riding a bike to work, driving, or staying at home and “telecommuting” in order to attend an important meeting starting in 30 minutes. Due to health and cost concerns, the person places higher value on riding a bike (reward of 100 units), followed by driving (reward of 70 units) and telecommuting (no reward, since being present at the meeting is very important). In this simple example, we assume that the person has enough command over the means of transportation that the duration of each activity is fully within their control. However, there are two ways in which the environment can impact the schedule. If the person chooses to ride a bike, slippery road conditions may cause them to fall and get dirty, forcing them to change clothes before attending the meeting. Alternatively, if the person chooses to drive, there is the possibility of being involved in an accident, forcing them to first tow the vehicle and then catch a cab to get to work. The last and least preferred, albeit the safest, is staying home and telecommuting.

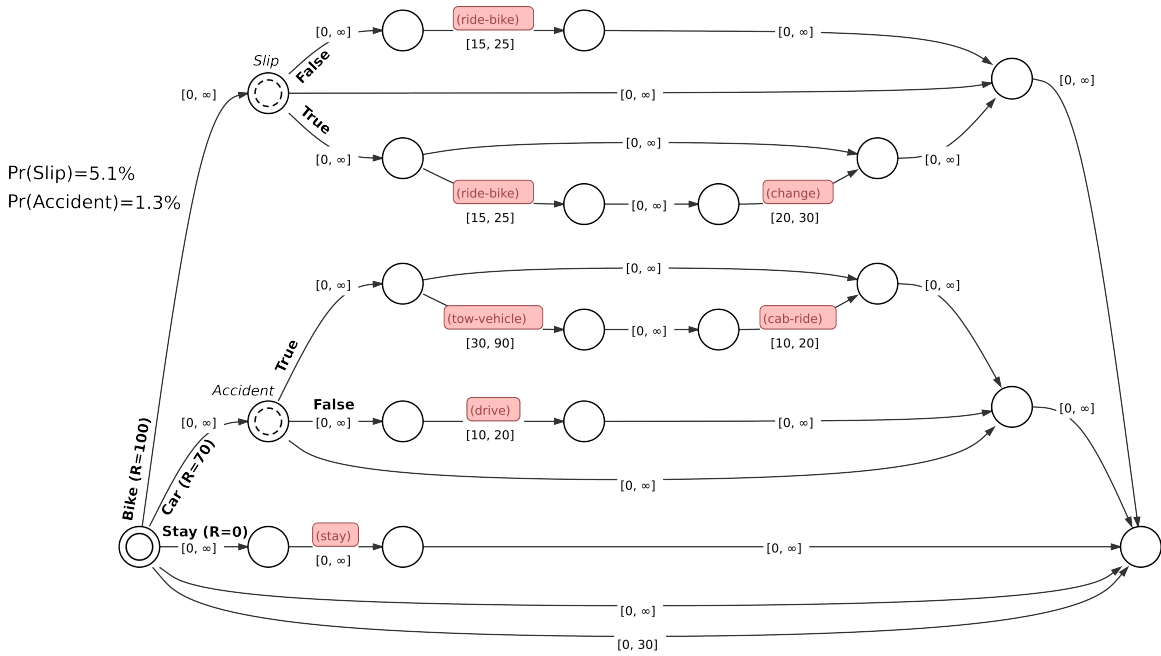


Figure 1-12: Probabilistic Temporal Plan Network obtained from the cRMPL program shown in Figure 1-11. Temporal constraints are represented as in Section 1.1; double circles with solid lines represent controllable choices (decisions) that the agent can make (in this case, the means of transportation); and double circles with dashed lines represent uncontrollable choices (observations) obtained from the environment.

When mapped to a CC-POMDP, different executions of the cRMPL program

shown in Figure 1-11 can be represented as the PTPN shown in Figure 1-12. The goal is to choose the means of transportation that yields the highest reward, while making sure that the probability of being late for the meeting is no greater than $\Delta = 2\%$. In this pedagogical example, it should be evident that the only decision that the person (playing the role of the autonomous agent) can make is how to get to the meeting, which is represented by the double circle with solid lines in Figure 1-12. If the person chooses to ride a bike (top branch of the choice node), the schedule will depend on the environment’s “choice” of having them slip and fall or not (dashed double circle at the top). Similarly, if the person chooses to drive (middle branch of the choice node), the schedule will depend on the environment “choosing” whether to have them be involved in an accident or not (dashed double circle in the middle). Finally, if the person chooses to stay home and telecommute, the environment has no impact on the schedule⁴.

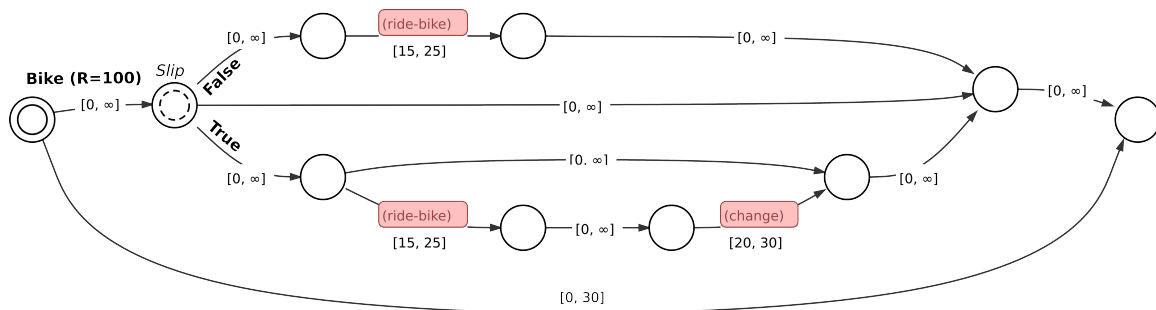


Figure 1-13: Temporal network entailed by choosing to ride a bike to work. A consistent schedule exists with probability 94.9% according to the cRMPL program in Figure 1-11, corresponding to the probability of not slipping and falling.

In order to understand the requirement of a policy π entailing a conditional constraint system for which a solution exists with high probability, consider the temporal networks in Figures 1-13, 1-14, and 1-15, which are entailed by the different commute choices that the person can make. Figures 1-13 and 1-14 are examples of *conditional* constraint systems, since the temporal constraints ultimately used for scheduling depend on the assignment to uncontrollable choices made by the environment, while Figure 1-15 is *unconditional*. Focusing on the conditions under which the temporal

⁴In an ideal world where Internet service providers are perfectly reliable.

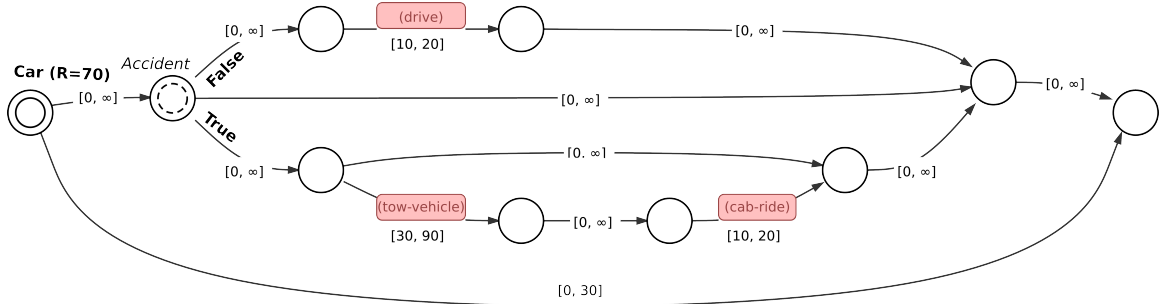


Figure 1-14: Temporal network entailed by choosing to drive to work. A consistent schedule exists with probability 98.7% according to the cRMPL program in Figure 1-11, corresponding to the probability of not being involved in an accident.

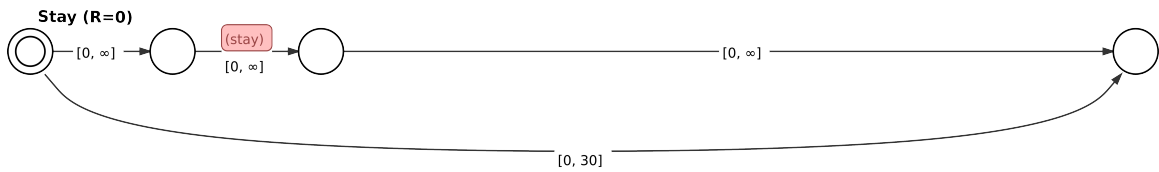


Figure 1-15: Temporal network entailed by choosing to stay at home and telecommuting. This option has a consistent schedule with probability 100%.

network in Figure 1-13 is consistent, we see that, if the person does not fall from the bike, there is plenty of time to ride it to work and arrive on time for the meeting. On the other hand, in case of a fall, the amount of time required to arrive at work and change is at least 35 minutes, causing the person to be late for the meeting. According to the execution model in the cRMPL program in Figure 1-11, falling from the bike is expected to happen with probability 5.1%, which violates our upper bound on tolerable risk of $\Delta = 2\%$. Therefore, even though riding a bike is the preferred option in terms of value, it does not meet the safety requirements expressed by the chance constraint. A similar analysis of Figure 1-14 would lead us to the conclusion that driving a car entails a risk of not having a consistent schedule of 1.3%, which corresponds to the probability of being involved in an accident. However, different from the previous case, a risk of 1.3% is within the tolerance of our chance constraint, rendering driving the optimal deterministic⁵ choice. Had the chance constraint Δ been lower than 1.3%, the only feasible option would have been to stay home and telecommute, since the temporal network in Figure 1-15 is guaranteed to have a consistent schedule (no

⁵In fact, the globally optimal strategy would be to randomize between riding a bike and driving a car, as explained in Chapter 3.

risk).

1.5 Thesis contributions

In its effort to make a contribution towards the endowment of autonomous agents with the ability to behave well and safely while operating under uncertainty, this thesis makes the following contributions:

Contribution 1: Chance-constrained POMDP’s as framework for risk-

bounded planning under uncertainty: our first contribution is a systematic derivation of dynamic *execution risk* for partially observable planning domains in Chapter 3, which we then use to propose Chance-constrained POMDP’s (CC-POMDP’s) as extensions of POMDP’s for risk-bounded planning under uncertainty. The advent of CC-POMDP’s and their associated dynamic measure of execution risk addresses shortcomings of existing static risk allocation and “risk-as-a-cost”-based methods. With respect to the former, we provide two different formulations of chance constraints in terms of CC-POMDP’s execution risk that are guaranteed to be no more conservative than the particular form of chance constraint used in static risk allocation. Regarding the latter, we show that CC-POMDP’s, unlike “risk-as-a-cost”-based methods, are not restricted to applications in which constraint violation must cause plan execution to halt.

Contribution 2: Risk-bounded AO* (RAO*): our second contribution is Risk-

bounded AO* (RAO*), the first algorithm for solving CC-POMDP’s that leverages heuristic forward search (HFS) over belief spaces. Similar to the original AO* [Nilsson, 1982], RAO* guides the search towards promising policies using an admissible value heuristic. In addition, RAO* leverages a second admissible heuristic to propagate dynamic execution risk upper bounds at each search node, allowing it to identify and prune overly risky paths as the search proceeds.

Contribution 3: Chance-constrained RMPL (cRMPL): our third contribu-

tion is the Chance-constrained Reactive Model-based Programming Language

(cRMPL), a novel variant of the RMPL language that introduces support to chance-constraints and probabilistic sensing actions. In addition to the language itself, we provide a novel CC-POMDP-based execution semantics for cRMPL that is the first to exploit the language’s hierarchical structure in order to extract optimal, risk-bounded execution policies from user-provided programs without the need of first “unraveling” all their possible execution traces, as was done in previous approaches.

Contribution 4: Fast, risk-aware probabilistic scheduling: since this thesis is particularly focused on *temporal* planning problems, our last contributions are in the form of PARIS (Chapter 5), the current fastest algorithm for risk-aware scheduling of Probabilistic Simple Temporal Networks (PSTN’s) and PSTN’s with Uncertainty (PSTNU’s); and the first algorithms for risk-aware scheduling of Probabilistic Temporal Plan Networks (PTPN’s) (Chapter 6).

1.6 Thesis roadmap

“A picture is worth a thousand words” is as much of a cliché as it is true. Therefore, prior to its textual explanation, we illustrate the structure of this thesis in Figure 1-16. The real-world need that motivates this thesis is featured at the very top of the block diagram, followed by this thesis’ goal on the second level supporting, but not completely fulfilling, this need, and the different components developed in this thesis to achieve its goal.

After this chapter motivates the need for risk-bounded dynamic execution of temporal plans by autonomous agents and presents an overview of this thesis’ contributions and key ideas, Chapter 2 places them in the context of the great body of work from which this thesis draws great inspiration and builds upon. Next, Chapters 3 to 5 present fundamental building blocks for the results shown in later chapters. Chapter 3 introduces Chance-constrained Partially Observable Markov Decision Processes (CC-POMDP’s) and the Risk-bounded AO* (RAO*) algorithm, two core components of the CLARK system explained in Section 1.4. In Chapter 4, we introduce Chance-

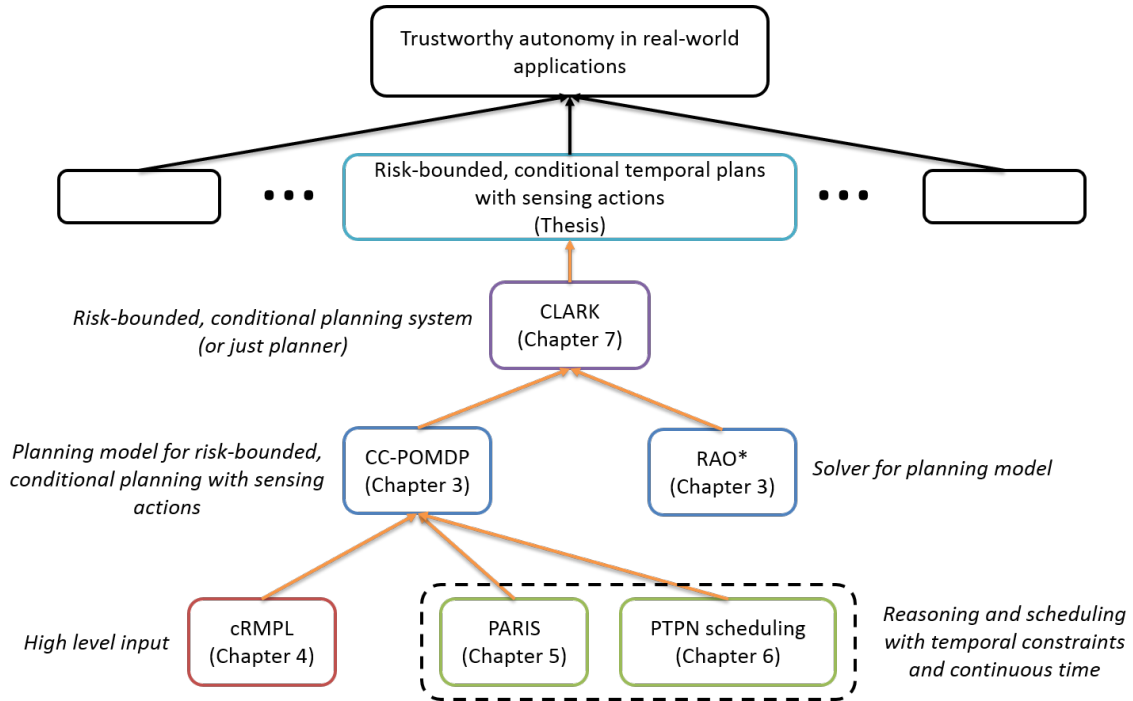


Figure 1-16: Block diagram illustrating the relationship between different components developed in this thesis, its goal (blue block on second level from the top), and the real-world need that motivates it (top block).

constrained RMPL (cRMPL), and how it allows mission operators to specify safe desired behavior for autonomous agents and provide expert “advice” at a high level of abstraction. When discussing the RAO* algorithm, a need for fast risk-aware constraint checkers becomes evident. Thus, in Chapters 5 and 6, we present conditional and unconditional temporal consistency checkers developed in this thesis for risk-aware scheduling under uncertainty.

After previous chapters provide a clear understanding of the fundamental pieces composing the CLARK system, Chapter 7 provides a holistic view of the system, and discusses experiments that show CLARK achieving its goal of generating risk-bounded conditional plans over rich sets of mission constraints in two application domains that can greatly benefit from risk-aware autonomy: collaborative human-robot manufacturing and data retrieval agents. In the *decision-theoretic* setting of Section 7.2, an operator provides a partial behavior specification in the form of a cRMPL program with contingency nodes, where the latter can be either controllable

(decisions by the autonomous agent) or uncontrollable (observations received from the environment). Alternatively, in the *generative* setting of Section 7.3, planning problems are encoded as CC-POMDP instances by enhancing an initial planning model specified in PDDL, and rely solely on the CC-POMDP model to determine the sequence in which activities are executed and observations are taken. Conclusions and avenues for future work are presented in Chapter 8, followed by appendices containing material that complements the core thesis discussions.

Appendix A shows how the RAO* algorithm from Chapter 3 can be extended to support partial belief state and policy branching, while Appendix B briefly presents a SAT-based model of electric power supply networks used as part of the experimental validation of RAO*. Appendix C shows the XML schema used to translate cRMPL programs into PTPN's, the input type to Pike. Finally, Appendix D shows the PDDL model used to construct a CC-POMDP model for CLARK in the Resilient Space Systems (RSS) demonstration of Chapter 7.

Chapter 2

Related work

“If I have seen further, it is by standing on the shoulders of giants.”

Isaac Newton, 1676.

Seeking to contribute to the widespread adoption of autonomous systems in safety-critical applications, we argue in Chapter 1 that autonomous agents must be able to execute risk-bounded, dynamic temporal plans that are conditional on sensing actions. Towards this goal, we propose that such risk-bounded conditional policies be generated as solutions to CC-POMDP’s, while further focusing on the particular case of temporal missions where agents must execute tasks under time pressure and temporal uncertainty. We also introduce the RAO* algorithm for computing CC-POMDP policies, and novel algorithms for risk-aware scheduling under uncertainty that efficiently provide RAO* with estimates of a mission’s scheduling risk. Finally, in order to allow mission operators to specify risk-aware autonomous behavior at a high level of abstraction, we introduce the cRMPL language and its execution semantics in terms of CC-POMDP’s.

The thesis contributions briefly summarized in the previous paragraph would not have been possible without leveraging invaluable insights and results from a broad range of related research fields. In this chapter, we strive to place this thesis within the large body of work that inspired it by drawing parallels between our contributions and related results in the literature.

2.1 (Constrained) MDP’s and POMDP’s

The excellent introduction to automated planning methods by Geffner and Bonet [Geffner and Bonet, 2013] states that “planning is the model-based approach to autonomous behavior where the agent behavior is derived automatically from a model of the actions, sensors, and goals.” The broadness of this definition should serve as an indication of how vast the planning field is, depending on a planning model’s particular features and assumptions. Therefore, for the sake of clarity, this section focuses on the subset of automated planning that had a direct impact on the development of CC-POMDP models in this thesis, as well as on the RAO* algorithm for solving the temporal subset of CC-POMDP’s that this thesis is concerned about.

Partially Observable Markov Decision Processes (POMDP’s), originally introduced in [Smallwood and Sondik, 1973], provide a conceptually simple, yet powerful, framework for generating plans (usually referred to as *policies*) that condition an agent’s actions on its belief about the hidden state of the world. Despite their inherent intractability [Papadimitriou and Tsitsiklis, 1987], POMDP models have grown in popularity in recent years with the advent of faster, cheaper computers, and a growing number of robotics applications that can directly benefit from a POMDP’s ability to handle state uncertainty [Kaelbling et al., 1998].

A solution to a risk-neutral, “standard” POMDP is a policy that maps belief states to actions in order to optimize some measure of expected utility [Smallwood and Sondik, 1973, Papadimitriou and Tsitsiklis, 1987, Kaelbling et al., 1998, Silver and Veness, 2010]. Nevertheless, this might not be sufficient to specify constrained autonomous behavior. For instance, an autonomous vehicle in an urban scenario should not minimize travel times at the expense of unlawful driving and unreasonable risk of collision. A straightforward way to incorporate a notion of constraint violation into POMDP models is to penalize states that violate these constraints in the objective function, and perform policy optimization as usual. However, this strategy has two important caveats. First, it might not be straightforward to precisely quantify “how bad” constraint violations are in units of the objective function: in the previous exam-

ple, what should be the penalty, measured in units of time, for hitting another car on the road? Second, even if one had a concrete way of quantifying these penalties, [Undurti and How, 2010, Undurti, 2011] show that there might not always be a smooth mapping from constraint violation penalties to the probability (risk) of these constraints being violated. In other words, variations of the constraint violation penalty over a continuous interval might entail discontinuous transitions from unreasonably risk-averse to unreasonably risk-taking behavior, with nothing in between. In light of these important limitations for the use of POMDP models in modeling risk-aware autonomous behavior, we proceed to the consideration of existing approaches in the literature for modeling risk-sensitive behavior.

One of the earlier works dealing with decision-making under uncertainty and probabilistic notions of risk is [Yu et al., 1998], which introduces Risk-Sensitive MDP’s (RS-MDP’s). Different from risk-neutral MDP’s, where the goal is to optimize an objective function, a policy for an RS-MDP is one that maximizes the probability of a cumulative cost function remaining below a user-specified threshold over a finite execution horizon. Therefore, instead of mapping states to actions, policies for RS-MDP’s map states and accumulated costs to actions. Improving upon the Value Iteration strategy of [Yu et al., 1998], the authors in [Hou et al., 2014] propose combinations of Topological Value Iteration [Dai et al., 2011] with Depth First Search and Dynamic Programming to efficiently compute RS-MDP policies. The RS-MDP framework has also been extended to POMDP’s [Marecki and Varakantham, 2010, Hou et al., 2016], where the objective becomes maximizing the joint probability of initial states and their cumulative costs exceeding the user-defined threshold.

Our approach to risk-aware conditional planning differs from RS-(PO)MDP’s in two important ways. First, our methods generate *risk-bounded* policies, whereas RS-(PO)MDP policies are *risk-minimal*. Second, RS-(PO)MDP’s assume that cost bound violations can be accurately detected by the autonomous agent and cause execution to halt, whereas we do not make assumptions about the observability of constraint violations in this thesis. The first distinction is important because risk-minimal strategies, as motivated in Section 1.1, can lead to strong conservatism by

inducing a “remain safe at all costs” behavior; a risk-bounded approach, on the other hand, allows an autonomous agent to optimize the value of its mission while keeping itself strictly within user-specified tolerable levels of risk. Concerning the observability of constraint violations, we develop in Chapter 3 a dynamic measure of execution risk that requires no assumptions regarding the observability of constraint violations, as it is necessary for RS-(PO)MDP’s. In that same chapter, we show that requiring constraint violations to halt execution in the modeling of risk-aware behavior can lead to conservatism and overconfidence.

Closer to the approach in this thesis, the Constrained MDP (C-MDP) [Feinberg and Shwarz, 1995, Altman, 1999, Dolgov and Durfee, 2005, Teichteil-Königsbuch, 2012, Sprauel et al., 2014, Trevizan et al., 2016] and POMDP (C-POMDP) [Isom et al., 2008, Undurti and How, 2010, Undurti, 2011, Kim et al., 2011, Poupart et al., 2015] literatures model situations in which a policy must optimize an objective while bounding the expected value of zero or more cost functions below user-specified thresholds. In this context, a bound on mission risk can be specified by creating an indicator cost function that outputs 1 whenever the agent enters a state that violates constraints, and 0 otherwise. Therefore, C-(PO)MDP’s can avoid excessive conservatism by not being restricted to the generation of risk-minimal policies. However, as shown in Chapter 3, the cost-based approach to risk in C-(PO)MDP’s requires constraint violations to halt execution, e.g., [Undurti and How, 2010, Poupart et al., 2015], in order to be guaranteed to return consistent values of risk, potentially causing the same conservatism and overconfidence issues previously mentioned for RS-(PO)MDP’s.

Related to C-(PO)MDP’s and this thesis, Lexicographic (PO)MDP’s [Wray et al., 2015, Pineda et al., 2015, Wray and Zilberstein, 2015] model multi-objective sequential decision problems under uncertainty where an agent lexicographically prefers to improve the value of objective i over objective $i + 1$ at all states. Since ties between multiple objectives are highly unlikely in practice, an L(PO)MDP that exactly optimizes objectives in lexicographic order would almost certainly focus all effort on optimizing the first objective, and completely disregard secondary ones. In order to prevent this from happening, L(PO)MDP’s allow a vector of user-defined *slacks* δ to

be associated with the competing objectives, so that actions available for improving objective $i + 1$ are restricted to those that keep objective i , preferred over $i + 1$, within a small degradation δ_i from its expected value under the optimal policy for the i -th objective.

Even though risk-aware planning is not specifically mentioned in [Wray et al., 2015, Pineda et al., 2015, Wray and Zilberstein, 2015], L(PO)MDP’s can be an interesting alternative to risk-minimizing models such as RS-(PO)MDP. For instance, an L(PO)MDP with mission risk as the primary objective could exploit its risk slack δ_1 to improve upon the conservatism of a policy seeking to minimize risk at all costs. However, in the risk-bounded planning setting of this thesis, L(PO)MDP have two important shortcomings. First, they leverage the same cost-based measure of risk used in C-(PO)MDP, therefore inheriting from the latter the same issues from requiring terminal constraint violations. Second, since slacks are defined relative to (usually) unknown optimal values of the different objectives, enforcing specific risk bounds on an L(PO)MDP policy can be difficult. For instance, in order to ensure that a policy has a risk bound of Δ , we would have to choose the risk slack to be $\delta_1 = \Delta - r_{\min}$, with r_{\min} being the unknown minimum risk value for a specific L(PO)MDP instance.

In order to solve CC-POMDP’s, this thesis introduces RAO*, an algorithm that follows a heuristic forward search strategy in the space of belief states similar to the originally proposed in [Washington, 1996, Washington, 1997, Hansen, 1998], and later revisited within a larger context of contingent and conformant planning methods in [Bonet and Geffner, 2000]. Our RAO* is the first risk-aware member of a family of search algorithms based on AO* [Nilsson, 1982, Pearl, 1984], the latter an extension of A* [Hart et al., 1968] to search on AND-OR graphs. Some notorious members of this family are LAO* [Hansen and Zilberstein, 2001], an extension of AO* that can search for infinite-horizon MDP policies by leveraging Value Iteration (VI) in its value update step; HAO* [Benazera et al., 2005a, Benazera et al., 2005b, Meuleau et al., 2009], a generalization of AO* that can handle hybrid (continuous and discrete) state spaces and continuous resources; and MAA* [Szer et al., 2005], a heuristic search algorithm for computing optimal finite-horizon policies for decentralized POMDP’s

(DEC-POMDPs).

Examples of sampling-based algorithms outside the “AO* family” that can also be used to solve (PO)MDP’s are Real-Time Dynamic Programming (RTDP) [Barto et al., 1995], Labeled RTDP (LRTDP) [Bonet and Geffner, 2003], UCT [Kocsis and Szepesvári, 2006], SARSOP [Kurniawati et al., 2008], and POMCP [Silver and Veness, 2010], among others. Even though none of these incorporate a notion of risk within their search process, risk-aware extensions using the notion of execution risk for CC-POMDP’s similar to RAO*’s should be possible. In Appendix A, we investigate the use Hoeffding’s inequality [Hoeffding, 1963] to develop risk-aware sampling-based solvers.

2.2 Scheduling under uncertainty

Generating conditional plans that allow autonomous agents to execute activities under time pressure, while dealing with stochastic activity durations and their associated scheduling risk, have always been a primary goal of this thesis, and one of its contributions. In this section, we review some of the most important results in the fields of conditional and unconditional scheduling that inspired the scheduling components used within CLARK (Chapters 5 and 6).

The work of Dechter, Meiri, and Pearl [Dechter et al., 1991], which introduces the Simple Temporal Problem (STP), should arguably be the first stop in any review of temporal reasoning methods. An STP is concerned about the existence of assignments (also known as a *schedule*) to temporal events e_i such that a conjunction of linear inequalities (or *simple temporal constraints*) $l \leq e_i - e_j \leq u$ hold, where l and u are given real numbers. An STP is said to be *consistent* if, and only if, a schedule respecting all simple temporal constraints exists. A graph representation of the simple temporal constraints in an STP is usually referred to as a Simple Temporal Network (STN). An STN can model temporal problems where the activity scheduler has complete control over temporal assignments. Thus, if the STN is used by an autonomous agent to schedule activities, there is an implicit assumption that the agent

can exactly control how long each one of the activities is going to take. However, this might not be reasonable in situations where activity durations are influenced by the external environment, such as “boil water”, “search for rock”, or “drive from MIT to Logan airport during rush hour”. In order to address this limitation, the STN with Uncertainty (STNU) was introduced [Vidal, 1999].

An STNU extends an STN with *contingent* (also called *uncontrollable*) constraints and events. In an STNU, a contingent constraint allows the difference between two temporal events to be non-deterministic, but bounded by a known interval $[l, u]$. Depending on how much information about contingent durations is made available to the scheduler during execution, [Vidal, 1999] defines different levels of *controllability* for STNU’s: *weak controllability* assumes all values of contingent durations to be known to the scheduler before it has to make any decisions; *strong controllability* assumes that no such information is ever available to the scheduler; and *dynamic controllability* assumes that the scheduler can only use information about past contingent durations when making future scheduling decisions. Over the years, increasingly more efficient algorithms for STNU scheduling were developed, specially for dynamic controllability [Morris et al., 2001, Morris and Muscettola, 2005, Morris, 2006, Hunsberger, 2009, Hunsberger, 2010, Hunsberger, 2013, Hunsberger, 2014, Morris, 2014].

Modeling contingent durations using STNU’s, however, offers no way to incorporate probabilistic information into contingent duration models. For instance, an STNU cannot model the statement “my morning commute takes 32 minutes on average, with a standard deviation of 10 minutes”. Due to this lack of probabilistic information, STNU scheduling algorithms ensure correctness by resorting to an “all-or-nothing” approach: either a schedule works for all possible realizations of contingent durations, or no schedule is returned, which tends to lead to strong conservatism, as shown in the motivating example in Section 1.1.

The Probabilistic STN (PSTN) [Tsamardinos, 2002] improves upon STNU’s by allowing contingent durations to be represented as random variables with known probability distributions. The notions of controllability for STNU’s can be readily transferred to PSTN’s, and have been further extended with a notion of *scheduling*

risk: different from STNU algorithms, which generate schedules with no scheduling risk or no solution at all, current PSTN algorithms [Tsamardinos, 2002, Wang, 2013, Fang et al., 2014, Fang, 2014, Wang and Williams, 2015a, Santana et al., 2016c] allow schedules to consider restricted intervals of contingent durations - a process informally referred to as “squeezing” -, provided that these restricted duration intervals are observed with high probability. While previous approaches to PSTN scheduling resorted to nonlinear problem formulations, Chapter 5 describes PARIS, the current fastest algorithm for strong PSTN scheduling, which leverages linearized models of temporal uncertainty that are key to CLARK’s empirical tractability when dealing with planning problems with probabilistic temporal uncertainty.

All related work mentioned so far falls into the realm of *unconditional scheduling*, in which the set of temporal constraints composing the temporal reasoning problem is known from the beginning. However, as evidenced by the grounded example in Section 1.4, the conditional nature of the plans generated by CLARK gives rise to *conditional scheduling* problems, in which the set of temporal constraints composing the scheduling problem can depend on *decisions* (or *controllable choices*) made by the autonomous agent and real-time *observations* (or *uncontrollable choices*) collected during plan execution. Therefore, in the following we review extensions of the previous unconditional scheduling formalisms to conditional settings involving controllable and uncontrollable choices.

In addition to STP’s, the authors in [Dechter et al., 1991] define the Temporal Constraint Satisfaction Problem (TCSP), a superclass of STP’s in which the scheduler can choose to enforce one out of potentially several different simple temporal constraints between pairs of events. The Disjunctive Temporal Problem (DTP) [Stergiou and Koubarakis, 2000] improves upon TCSP’s by not requiring that these choices between temporal constraints be restricted to the same pairs of events. The conditional temporal network associated with a DTP is called a Disjunctive Temporal Network (DTN). The first strategy for dynamic execution of DTP’s [Tsamardinos et al., 2001], which generated an exponential number of choice combinations and compiled them independently, was later improved by incremental and labeled compilation strategies

presented in [Shah et al., 2007, Shah and Williams, 2008, Shah et al., 2009, Conrad et al., 2009].

Temporal Plan Networks (TPN's) [Kim et al., 2001, Effinger, 2006] are within the same group of temporal networks with simple temporal constraints and controllable choices as TCSP's and DTN's. However, unlike the two previous, TPN's allow preferences to be placed over the choice assignments, therefore extending the notion of network consistency to *optimal* network consistency. In terms of structure, TPN's are a hierarchical subset of DTN's that are particularly useful in representing execution traces of non-deterministic programs, as shown in Section 1.4 and Chapter 4. Current dynamic executives for TPN's are Kirk [Kim et al., 2001, Shu et al., 2005, Block et al., 2006, Effinger, 2006], Drake [Conrad et al., 2009, Conrad, 2010, Conrad and Williams, 2011], and Pike [Levine and Williams, 2014].

Similar to how STNU's extend STN's, a DTN with Uncertainty (DTNU) [Venable and Yorke-Smith, 2005] extends DTN's by allowing choices between both simple temporal constraints and contingent durations. The concepts of strong [Peintner et al., 2007], weak [Venable et al., 2010], and dynamic [Venable et al., 2010, Cimatti et al., 2014, Cimatti et al., 2016b] consistency for DTNU's are directly borrowed from STNU's, with the added feature that the scheduler, in addition to assigning times to controllable temporal events, can also choose which members of disjunctions of temporal constraints to satisfy.

On a complementary side of the conditional scheduling spectrum, the Conditional Temporal Problem (CTP) [Tsamardinos et al., 2003] was the first to allow the conditioning of the temporal constraints composing a scheduling problem on real-time *observations*. Thus, unlike DTP's, the scheduler for a CTP can only observe the outcome of these uncontrollable choices, instead of being able to directly assign them. Depending on when information about these observations is available, the authors in [Tsamardinos et al., 2003] define notions of weak, strong, and dynamic consistency for CTP's analogous to STNU's: *weak consistency* assumes all values of observations to be known beforehand; *strong consistency* assumes that no such information is ever available; and *dynamic consistency* assumes that the scheduler can only use

information about past observations when making future scheduling decisions.

Taking CTP’s one step further, the Conditional STNU (CSTNU) [Hunsberger et al., 2012] allows both contingent STNU-like durations, as well as the conditioning of constraints on real-time observations. Therefore, weak, strong, and dynamic consistency for CSTNU refer to the availability of information about both contingent durations *and* real-time observations. The problem of checking dynamic consistency of CSTNU’s is a topic of very recent research interest [Combi et al., 2013, Cimatti et al., 2016a], with state-of-the-art methods based on fast algorithms for Timed Game Automata [Cassez et al., 2005].

Unifying features from all previous formalisms, a TPN with Uncertainty (TPNU) [Effinger, 2006, Effinger et al., 2009, Effinger, 2012] extends TPN’s with STNU-like contingent durations and real-time observations, while a Probabilistic TPN (PTPN) [Effinger, 2012, Santana and Williams, 2014] adds another increment of expressiveness to TPNU’s by allowing PSTN-like durations and probabilistic models for observation nodes. The current algorithm in [Effinger, 2012] for dynamic scheduling of PTPN’s is based on mapping to a risk-minimizing MDP with time discretization, which prevents it from scaling beyond very small problems over short time horizons. Therefore, in addition to the PARIS algorithm for risk-aware unconditional scheduling from Chapter 5, this thesis also proposes the first algorithms for risk-aware strong and weak scheduling of PTPN’s [Santana and Williams, 2014]. Chapter 6 describes an improvement upon the original algorithms proposed in [Santana and Williams, 2014] by replacing the STN scheduler used in [Santana and Williams, 2014] by PARIS, which can handle any combination of STNU- and PSTN-like uncontrollable durations.

2.3 Temporal and hybrid planning

As pointed out in Chapter 1, the development of CLARK in this thesis is motivated by *temporal* planning applications, particularly by situations in which activity scheduling must be performed under temporal uncertainty. Moreover, in an effort to extend CLARK to handle state constraints commonly found in robotic applications, such

obstacle-avoidance constraints in path planning, we incorporate insights from the hybrid planning community in our introduction of a theory-dependent constraint violation functions c_v in our definition of CC-POMDP's (see Section 1.3 and Chapter 3). Therefore, CLARK is closely related to existing temporal and hybrid planners in the literature, which we review next.

The advent of the Planning Domain Definition Language (PDDL) [McDermott et al., 1998] was invaluable to the field of *classical planning*, for it provided a standardized input language to existing planners that facilitated the assessment of their relative strengths and weaknesses in subsequent iterations of the International Planning Competition. However, the original PDDL lacked features that are key for the appropriate modeling of realistic planning domains, such as robotics and industrial control applications. Among these features, the ability to handle continuous time, numerical resources, and complex dynamics are key [Fox and Long, 2003, Fox and Long, 2006]. In this context, the PDDL2.1 [Fox and Long, 2003] and PDDL+ [Fox and Long, 2006] languages were proposed as enhancements to PDDL for the modeling, respectively, of temporal and hybrid (mixed discrete-continuous) planning domains, ushering the development of many algorithms that extend beyond the limitations of classical planning and relate to this thesis.

Temporal planning [Fox and Long, 2003, Cushing et al., 2007] is concerned with the generation of plans consisting of *durative actions*, which may or may not be executed concurrently. With the introduction of *timed initial literals* in PDDL2.2 during the 4th International Planning Competition [Hoffmann and Edelkamp, 2005], it also became possible to designate time windows and goal deadlines in temporal planning domains. Starting with ZENO [Penberthy and Weld, 1994], a temporal planner predating PDDL2.1, many different temporal planning algorithms were proposed. Among some of the most noteworthy, we can mention TGP [Smith and Weld, 1999], which incrementally expands a compact planning graph representing mutual exclusions between actions of different durations; Sapa [Do and Kambhampati, 2003], a heuristic forward-chaining planner that can handle durative actions, metric resource constraints, and goal deadlines; CRIKEY3 [Coles et al., 2008], a forward-chaining

temporal planner that supports discrete and duration-independent numeric change; COLIN [Coles et al., 2009, Coles et al., 2012], CRIKEY3’s direct successor, which leverages linear programming to support continuous processes with linear dynamics; POPF [Coles et al., 2010], also a forward-chaining temporal planner that, rather than enforcing a strict total order on all steps added to the plan, pursues a partial-order planning strategy; OPTIC [Benton et al., 2012], which builds upon POPF and supports a wider variety of temporal and non-temporal preferences; and tBurton [Wang and Williams, 2015b, Wang, 2015], a temporal planner for complex networked systems that follows a divide-and-conquer approach for the generation of plans that meet deadlines and maintain durative goals.

Common to all previously-mentioned temporal planners is the assumption that time is *fully controllable*, i.e., the temporal planner can freely choose the start and stop times for the durative actions in the plan, which may not be reasonable for real-world planning domains. Therefore, recent efforts have been undertaken to extend temporal planning to situations in which temporal uncertainty is explicitly taken into account. Along these lines, *conditional* [Mausam and Weld, 2008, Effinger, 2012] and *unconditional* strategies [Beaudry et al., 2010, Cimatti et al., 2015, Micheli et al., 2015] exist.

In the context of conditional strategies, the authors in [Mausam and Weld, 2008] propose Concurrent MDP (CoMDP) models, in which sets of primitive MDP actions can be simultaneously triggered, and use them to solve Concurrent Probabilistic Temporal Planning (CPTP) problems, where the durations of primitive MDP actions can be probabilistic. They map a CPTP into a CoMDP by considering all subsets of actions that can be executed concurrently and, for those, extend the state space with *interwoven* execution states. The latter are obtained by discretizing the duration intervals of each action being currently executed, and recording the number of time increments since they were started. A similar approach is used in [Effinger, 2012] for the dynamic execution of temporal plans with probabilistic durations, but only sequential (rather than concurrent) plans are considered.

Discretized time models are convenient for mapping problems of temporal plan-

ning under uncertainty directly into standard MDP frameworks, which can then be solved with existing algorithms. However, this comes at the expense of an exponential explosion in complexity, given the potentially enormous number of extra states generated by small deltas of temporal discretization over long planning horizons. In fact, some of the main contributions in [Mausam and Weld, 2008] are pruning strategies to mitigate the growth in complexity caused by temporal discretization. CLARK, on the other hand, avoids time discretization by representing the temporal constraints and uncertain durations as (conditional) networks of temporal constraints, and using risk-aware temporal consistency checkers (see Chapters 5 and 6) to verify the temporal feasibility of partial temporal plans.

With respect to unconditional planning with temporal uncertainty, [Beaudry et al., 2010] avoids time discretization by performing heuristic forward search in the space of planning states, while also constructing a Bayesian network that tracks the dependencies between random variables defining the times of events and the durations of actions. As new actions are added to the plan, samples are drawn from the Bayesian network to estimate the probability of the plan being temporally feasible, as well as its expected makespan. The generated plans are risk-bounded, for search terminates when a plan with high probability of success is found.

Direct sampling from a Bayesian network has the advantage of supporting general probabilistic dependencies between random durations, but introduces a significant overhead that grows with the size of the network [Cooper, 1990]. The authors in [Beaudry et al., 2010] address this overhead through a smart caching of samples, which sometimes causes their algorithm to run out of memory. In any case, their approach to temporal planning is similar to CLARK’s when restricted to the generation of unconditional plans, with the added distinction that CLARK’s scheduler for unconditional plans with temporal uncertainty (PARIS, Chapter 5) pursues an analytical approach when handling probabilistic temporal uncertainty.

Still in the field of unconditional planning with temporal uncertainty, there have also been recent developments for the Strong Planning Problem with Temporal Uncertainty (SPPTU) [Cimatti et al., 2015]. A temporal plan for an SPPTU is valid

if, for all possible durations of the uncontrollable actions, the plans resulting from considering such durations as fixed are valid, disregarding temporal uncertainty. According to [Micheli et al., 2015], such *strong plans* (where “strong” means that the plan is robust to temporal uncertainty) are simple to execute and check, thus being suitable for safety-critical systems where guarantees are needed for the uncertainty, while potentially paying a sub-optimality price.

In order to solve SPPTU’s, [Cimatti et al., 2015] replaces the STN scheduler in COLIN [Coles et al., 2009, Coles et al., 2012] with an STNU strong controllability checker, while [Micheli et al., 2015] describes a sound-and-complete compilation technique from temporal plans with STNU-like durations to temporal plans without uncertainty, which is applied to a fragment of the ANML [Smith et al., 2008] language. CLARK differs from these planners by supporting both STNU- and PSTN-like models of duration uncertainty, and including a notion of scheduling risk in the temporal plans that it outputs.

This thesis’ approach to the generation of conditional plans subject to different types of hybrid (continuous-discrete) state constraints, e.g., scheduling under uncertainty, path planning, and power supply, is directly influenced by Planning Modulo Theories (PMT) [Gregory et al., 2012], an extension of concepts from SAT Modulo Theories (SMT) [Nieuwenhuis et al., 2006] to planning; semantic attachment [Dornhege et al., 2012]; and the approach to optimal planning with global numerical state constraints in [Ivankovic et al., 2014]. The idea is to perform a decomposition of a planning problem with hybrid state constraints in two parts: a master planning problem over symbolic operators, whose “high-level” actions usually have a direct impact on the “low-level” hybrid state of the Plant; and a series of reasoning modules that verify the feasibility of constraint theories through specialized solvers.

Variations of this decomposition approach can be found in Task And Motion Planning [Lozano-Pérez and Kaelbling, 2014, Srivastava et al., 2014, Toussaint, 2015, Fernández-González et al., 2015, Lin et al., 2016], usually in the context of activity planning for mobile robots with path planning and manipulation constraints; the use of linear programs in COLIN [Coles et al., 2009, Coles et al., 2012], uNICOrn [Bajada

et al., 2015], and in [Ivankovic et al., 2014] for checking temporal and dynamics constraints; the SMT-based nonlinear planning approach in [Bryce et al., 2015] involving a SAT solver combined with a differential equation solver; the conversion of PDDL+ problems into SAT modulo linear programs in TM-LPSAT [Shin and Davis, 2005]; and the UPMurphi [Della Penna et al., 2012] and DiNo [Piotrowski et al., 2016] hybrid planners for PDDL+, both based on the *discretize and validate* approach using the plan validator VAL [Howey et al., 2004, Della Penna et al., 2012]. Kongming [Li and Williams, 2008, Li, 2010, Li and Williams, 2011], one of the first planners for hybrid domains, follows a decomposition approach that combines Graphplan’s [Blum and Furst, 1997] planning graph with Mixed Integer Linear Programs (MILP’s) for trajectory optimization over continuous states. Our approach in CLARK is different from these hybrid planners by generating conditional plans that handle state uncertainty; and leveraging *risk-aware* constraint checkers for the generation of risk-bounded conditional plans, such as risk-aware path planners with uncertain dynamics [Ono and Williams, 2008, Ono et al., 2012b, Ono et al., 2012a, Arantes et al., 2016b] and risk-aware schedulers (Chapters 5 and 6).

2.4 Programming languages for autonomy

In an effort to provide agent designers with a convenient tool to specify risk-aware autonomous behavior at a high level of abstraction, this thesis draws great inspiration from the Probabilistic Reactive Model-based Programming Language (pRMPL) in [Effinger, 2012] to develop Chance-constrained RMPL (cRMPL) in Chapter 4. A key contribution of pRMPL is the inclusion of exception handling and sensing into the RMPL language [Ingham et al., 2001, Williams et al., 2001, Williams and Ingham, 2002, Williams et al., 2003, Ingham, 2003]: sensing is crucial when operating autonomously in an unstructured environment, while exception handling enables graceful recovery from failures by localizing error identification and recovery, rather than halting the entire executive.

Our cRMPL differs from and improves upon [Effinger, 2012] in three important

ways: first, cRMPL enables risk-bounded programming by adding support to chance constraints, thus moving away from the potential conservatism of risk-minimizing programs in pRMPL. Second, we define the execution semantics of cRMPL in terms of CC-POMDP’s, and show that RAO* can exploit the language’s hierarchical structure in order to extract optimal, risk-bounded execution policies from user-provided programs without the need to first “unravel” all possible execution traces. On the other hand, pRMPL programs are first unraveled into a Probabilistic Temporal Plan Network (PTPN) [Effinger et al., 2009, Effinger, 2012, Santana and Williams, 2014, Levine and Williams, 2014], an exponentially large representation of all possible program traces, which is then used to construct an MDP with discretized temporal durations that is solved through Value Iteration. Since Value Iteration operates on a complete representation of the state space, which becomes significantly larger due to time discretization, execution of pRMPL programs suffers from scalability issues. Finally, from a software engineering standpoint, our choice to implement cRMPL as an extension of Python, as opposed to pRMPL’s implementation as a standalone language, makes it easy to integrate cRMPL within current robotics frameworks, e.g., the Robot Operating System (ROS) [Quigley et al., 2009], a key capability for modern agent programming languages [Ziafati et al., 2012].

As the most recent variant of the RMPL language [Ingham et al., 2001, Williams et al., 2001, Williams and Ingham, 2002, Williams et al., 2003, Ingham, 2003], cRMPL inherits from the latter its relationship to several robot execution [Firby, 1990, Gat, 1997, Simmons and Apfelbaum, 1998], concurrent constraint [Gupta et al., 1996], decision-theoretic [Andre and Russell, 2002, Boutilier et al., 2000, Fritz and McIlraith, 2005, Effinger, 2012], and concurrent reactive [Le Guernic et al., 1986, Harel, 1987, Halbwachs et al., 1991, Halbwachs, 1998, Berry and Gonthier, 1992] programming languages for autonomous systems, which are discussed in the following.

With respect to decision-theoretic languages, DTGolog [Boutilier et al., 2000, Fritz and McIlraith, 2005] extends GOLOG [Levesque et al., 1997] to handle decision-theoretic choice and stochastic outcomes via an interpreter that views optimal completion of a GOLOG program as an MDP. DTGolog allows the programmer to sense

the state of the world, as in an `if-then-else` construct, and is able to estimate the likelihood of program success. In the context of reinforcement learning [Sutton and Barto, 1998], where an autonomous agent observes the outcomes of its actions in the environment in order to learn a policy, ALisp [Andre and Russell, 2002] augments the Lisp language with a non-deterministic choice operator and the ability to execute primitive MDP actions. Similarly to cRMPL, the nondeterministic choice operator in ALisp improves tractability by restricting the decisions made by an agent to those explicitly outlined by the programmer. Both DTGolog and ALisp differ from cRMPL by not supporting concurrency and timing uncertainty, which are key elements for the temporal planning applications that this thesis is concerned about.

As explained in [Effinger, 2012], synchronous languages adhere to a strict notion of concurrency, called the *synchrony hypothesis*, to ensure that safety-critical systems are capable of responding in real time, and that system behavior is defined in a mathematically precise way. By doing so, synchronous languages provide a strong model for concurrency and precision in the construction of reactive embedded systems. The synchronous approach originated in France with the sister languages ESTEREL [Berry and Gonthier, 1992], Lustre [Halbwachs et al., 1991], and Signal [Le Guernic et al., 1986], and then grew to include other languages such as STATECHARTS [Harel, 1987]. However, unlike cRMPL, synchronous languages do not support decision-theoretic temporal planning.

Model-based programming and concurrent constraint programming share common underlying principles, including the notion of computation as deduction over systems of partial information [Gupta et al., 1996]. The goal of control programs written in variants of RMPL, including cRMPL, is to specify the desired behavior of the Plant by means of state trajectory constraints [Williams et al., 2003]. Therefore, cRMPL’s model of interaction is in contrast to the Timed Concurrent Constraint (TCC) programming language [Gupta et al., 1996], which interacts with “low-level” program memory, sensors, and control variables directly, not with the Plant state. By specifying state trajectories at a high level of abstraction, cRMPL releases the programmer from the responsibility to perform the mapping between intended state and

the sensors and actuators, a burden that gets transferred to the program executive.

Finally, as all previous versions of RMPL, cRMPL builds upon previous work on robotic execution languages, such as RAPS [Firby, 1990], ESL [Gat, 1997], and TDL [Simmons and Apfelbaum, 1998], and their goal-directed tasking and monitoring capabilities. A key distinction is that cRMPL constructs fully cover synchronous programming, thus providing integrated goal-directed and temporal reasoning capabilities.

Chapter 3

Generating chance-constrained, conditional plans

“A key element of risk decision-making is determining if the risk is justified.”

FAA Risk Management Handbook. [Administration, 2009]

In this thesis, we argue that an important step towards safe and trustworthy autonomous agents is their endowment with the ability to execute risk-bounded, dynamic temporal plans that are conditional on sensing actions. In this chapter, we lay the foundations for this goal by introducing Chance-Constrained POMDP’s (CC-POMDP’s), a formalism that extends POMDP’s with a dynamic notion of risk and risk-bounded execution. As shown in Section 1.4, CC-POMDP’s play a central role in this thesis by serving as inputs to CLARK, and CC-POMDP solutions are exactly the risk-bounded, conditional plans that this thesis seeks to generate. To achieve the latter, this chapter also introduces RAO*, a heuristic forward search algorithm producing optimal, deterministic, finite-horizon policies for CC-POMDP’s that, in addition to a utility heuristic, leverages an admissible execution risk heuristic to quickly detect and prune overly-risky policy branches.

The experimental results presented in this chapter focus on the performance improvements that result from the modeling of risk-sensitive planning domains as CC-

POMDP’s, and RAO*’s usefulness in efficiently generating policies for challenging CC-POMDP domains of practical interest. In Chapter 7, we make it clear how CC-POMDP’s and RAO* can be combined with the scheduling algorithms presented in Chapters 5 and 6 to allow CLARK to generate risk-bounded, conditional *temporal* plans featuring scheduling uncertainty.

3.1 Introduction

Partially Observable Markov Decision Processes (POMDP’s) [Smallwood and Sondik, 1973] have become one popular framework for optimal planning under actuator and sensor uncertainty, where POMDP solvers find policies that maximize some measure of expected utility [Kaelbling et al., 1998, Silver and Veness, 2010].

In many application domains, however, maximum utility is not enough. Critical missions in real-world scenarios require agents to develop a keen sensitivity to risk, which needs to be traded-off against utility. For instance, a search and rescue drone should maximize the value of the information gathered, subject to safety constraints such as avoiding dangerous areas and keeping sufficient battery levels. In these domains, autonomous agents should seek to optimize expected reward while remaining safe by deliberately keeping the probability of violating one or more constraints within acceptable levels. A bound on the probability of violating constraints is called a *chance constraint* [Birge and Louveaux, 1997]. Unsurprisingly, attempting to model chance constraints as negative rewards leads to models that are over-sensitive to the particular penalty value chosen, and to policies that are overly risk-averse or overly risk-taking [Undurti and How, 2010]. Therefore, to accommodate the aforementioned scenarios, new models and algorithms for *constrained* MDP’s have started to emerge, which handle chance constraints explicitly.

Research has mostly focused on fully observable constrained MDP’s, for which non-trivial theoretical properties are known [Altman, 1999, Feinberg and Shwarz, 1995]. Existing algorithms cover an interesting spectrum of chance constraints over secondary objectives or even execution paths, e.g., [Dolgov and Durfee, 2005, Hou

et al., 2014, Teichteil-Königsbuch, 2012]. For constrained POMDP’s (C-POMDP’s), the state of the art is less mature. It includes a few suboptimal or approximate methods based on extensions of dynamic programming [Isom et al., 2008], point-based value iteration [Kim et al., 2011], approximate linear programming [Poupart et al., 2015], or on-line search [Undurti and How, 2010]. Moreover, as we later show, the modeling of chance constraints through unit costs in the C-POMDP literature has a number of shortcomings.

The first contribution brought by this chapter is a systematic derivation of *execution risk* in POMDP domains, which can be used to enforce different types of chance constraints and improves upon the restrictions of previous cost-based approaches. A second contribution is Risk-bounded AO* (RAO*), the first algorithm for solving chance-constrained POMDP’s (CC-POMDP’s), which harnesses the power of heuristic forward search in belief space [Washington, 1996, Bonet and Geffner, 2000, Szer et al., 2005, Bonet and Geffner, 2009]. Similar to AO* [Nilsson, 1982], RAO* guides the search towards promising policies with respect to reward using an admissible heuristic. In addition, RAO* leverages a second admissible heuristic to propagate execution risk upper bounds at each search node, allowing it to identify and prune overly risky paths as the search proceeds. We demonstrate the usefulness of RAO* in two risk-sensitive domains of practical interest: automated power supply restoration and autonomous science agents. As pointed out in Chapter 1, for computational tractability reasons and the time-bounded nature of the applications that this thesis is concerned about, we follow [Dolgov and Durfee, 2005] in deliberately developing an approach focused on deterministic policies with a finite number of steps, even though the exact bound on the number of planning steps might not be known beforehand.

The chapter is organized as follows. Section 3.2 formally present CC-POMDP’s, and details how execution risk for a conditional plan can be dynamically computed. Next, Section 3.3 discusses shortcomings related to the cost-based treatment of chance constraints in the C-POMDP literature. Section 3.4 presents the RAO* algorithm, followed by our experiments in Section 3.5, and chapter conclusions in Section 3.6.

3.2 Problem formulation

The goal of this section is to formally introduce all the necessary concepts to fully define and understand CC-POMDP's, expanding upon the brief introduction given in Section 1.3. We start by presenting fundamental equations involved in the handling of belief states in Section 3.2.1, followed by a thorough development in Section 3.2.2 of the dynamic mission risk equations that sit at the core of CC-POMDP's (Section 3.2.3) and the RAO* algorithm. We end this section with a discussion about different types of chance constraints in Section 3.2.4.

3.2.1 Managing belief states

When the true state of the underlying system under control, also referred to as the *Plant*, is uncertain and cannot be directly observed, one can only maintain a probability distribution over possible Plant states at any given point in time. We call this representation of state uncertainty a *belief state*, or just *belief*, and we denote by \mathcal{B} the set of beliefs states over \mathcal{S} . For the conditional planning application in this thesis, we will focus on belief states over a *discrete* set of states \mathcal{S} . As depicted in Figure 3-1, a discrete belief state can be understood as set of N state-probability pairs (s, p) , each pair also referred to as a *particle*, such that p represents the probability of s being the true hidden state of the Plant. For notational convenience when dealing with belief states that evolve over time, we denote a belief state at planning step k by $b_k : \mathcal{S} \rightarrow [0, 1]$, and $b(s_k)$ as the probability of state s at step k according to belief b . Moreover, in order for b to be a proper belief, it must be the case that all particle probabilities sum to 1, i.e.,

$$\sum_{s_k} b(s_k) = 1, \forall k \in \mathbb{N}. \quad (3.1)$$

We now distinguish between beliefs based on how much information they incorporate. For that, let \mathcal{O} denote a discrete set of possible sensor observations, and let $o_{m:n}$ denote a sequence of sensor observations between, and including, time steps m and

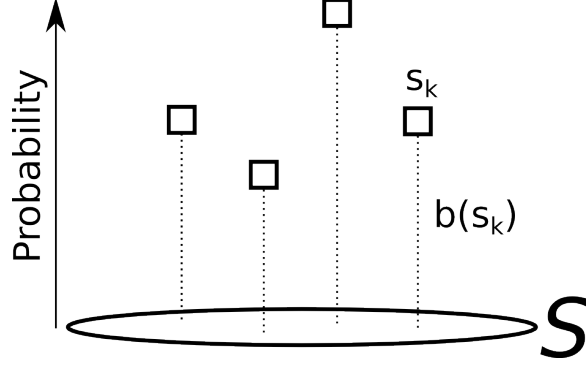


Figure 3-1: Depiction of a discrete belief state over a state space \mathcal{S} .

n . Similarly, let \mathcal{A} and $a_{m:n}$ denote, respectively, a discrete set of plan actions and a sequence of such actions. When a belief state at time k incorporates all sensor and action information available up to, and including, time k , we call this belief *posterior* and will use the notation \hat{b}_k . If, however, the belief state at time k includes action information up to time k , but does not incorporate the latest observation o_k , we call this belief *prior*¹ and denote it by \bar{b}_k . More precisely, prior and posterior beliefs represent the following probability distributions:

$$\begin{aligned}\bar{b}(s_k) &= \Pr(s_k | o_{1:k-1}, a_{0:k-1}), \\ \hat{b}(s_k) &= \Pr(s_k | o_{1:k}, a_{0:k-1}).\end{aligned}\tag{3.2}$$

Our goal now is to develop equations that allow us to propagate beliefs forward in time as the agent executes actions from its conditional plan and collects sensor observations. In order to do so, we follow the standard practice in the POMDP literature and assume the availability of a *stochastic state transition model* $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ such that

$$T(s_k, a_k, s_{k+1}) = \Pr(s_{k+1} | s_k, a_k);\tag{3.3}$$

¹It is also common to refer to prior and posterior beliefs as *a priori* and *a posteriori* beliefs, respectively. We say prior and posterior to highlight the fact that these are belief states *before* and *after* incorporating the last available sensor observation.

and a *stochastic observation model* $O : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ such that

$$O(s_k, o_k) = \Pr(o_k | s_k). \quad (3.4)$$

The Markovian² models (3.3) and (3.4) are directly borrowed from Hidden Markov Models (HMM's) [Baum and Petrie, 1966], with the added feature that the state transition model in (3.3) is parametrized by the action selection.

Suppose that the posterior belief $\hat{b}(s_k)$ is available, and that the agent has chosen to execute action a_k . The process through which information about a_k is incorporated into $\hat{b}(s_k)$ is called *prediction* and allows us to compute $\bar{b}(s_{k+1})$ using (3.3) as follows:

$$\begin{aligned} \bar{b}(s_{k+1}) &= \Pr(s_{k+1} | o_{1:k}, a_{0:k}) = \sum_{s_k} \Pr(s_{k+1}, s_k | o_{1:k}, a_{0:k}), \\ &= \sum_{s_k} \Pr(s_{k+1} | s_k, o_{1:k}, a_{0:k}) \Pr(s_k | o_{1:k}, a_{0:k-1}, a_k), \quad (3.5) \end{aligned}$$

$$= \sum_{s_k} \Pr(s_{k+1} | s_k, a_k) \Pr(s_k | o_{1:k}, a_{0:k-1}), \quad (3.6)$$

$$= \sum_{s_k} T(s_k, a_k, s_{k+1}) \hat{b}(s_k). \quad (3.7)$$

In the transition from (3.5) to (3.6), we exploit the Markov property of (3.3) to write $\Pr(s_{k+1} | s_k, o_{1:k}, a_{0:k}) = \Pr(s_{k+1} | s_k, a_k)$ and the fact that executing a_k does not impact our previous belief $\hat{b}(s_k)$ to write $\Pr(s_k | o_{1:k}, a_{0:k-1}, a_k) = \Pr(s_k | o_{1:k}, a_{0:k-1})$.

Next, suppose that the agent also collects observation o_{k+1} after executing a_k . The *measurement update*, or just *update*, step uses (3.4) and Bayes' rule to incorporate o_{k+1} into $\bar{b}(s_{k+1})$ and generate $\hat{b}(s_{k+1})$ as follows:

$$\hat{b}(s_{k+1}) = \Pr(s_{k+1} | o_{1:k+1}, a_{0:k}) = \frac{1}{\eta} \Pr(o_{k+1} | s_{k+1}, o_{1:k}, a_{0:k}) \Pr(s_{k+1} | o_{1:k}, a_{0:k}), \quad (3.8)$$

$$= \frac{1}{\eta} \Pr(o_{k+1} | s_{k+1}) \bar{b}(s_{k+1}), \quad (3.9)$$

$$= \frac{1}{\eta} O(s_{k+1}, o_{k+1}) \bar{b}(s_{k+1}), \quad (3.10)$$

²The models are said to be Markovian because the probability distributions for next states and observations are fully specified through the knowledge of the current state.

where (3.8) is Bayes' rule; (3.9) is obtained from (3.8) due to the Markov property of (3.4); and the normalization constant η is given by

$$\eta = \Pr(o_{k+1}|o_{1:k}, a_{0:k}) = \sum_{s_{k+1}} O(s_{k+1}, o_{k+1})\bar{b}(s_{k+1}). \quad (3.11)$$

Therefore, given an initial belief \hat{b}_0 representing our initial uncertainty about the state of the Plant, (3.7) and (3.10) can be repeatedly applied to propagate belief states forward in a process sometimes called *filtering*.

With a clear understanding of how action and sensor information is continuously used to refine an agent's belief as it executes a plan, the next section introduces the concept of *execution risk*, a dynamic measure of mission safety that sits at the core of CC-POMDP's and RAO*.

3.2.2 Computing mission risk dynamically

Following [Ono and Williams, 2008, Ono et al., 2012b, Ono et al., 2012a], we define *risk event* as a sequence $s_{0:k} = s_0, s_1, \dots, s_k$ of Plant states violating one or more constraints in a set $C \in 2^{\mathcal{C}}$, where \mathcal{C} is the set of all *conditional plan constraints*. Moreover, let $c_v : \mathcal{S} \times 2^{\mathcal{C}} \rightarrow \{0, 1\}$ be a constraint violation indicator such that $c_v(s, C) = 1$ if, and only if, s violates constraints in $C \in 2^{\mathcal{C}}$. By restricting c_v to be an indicator function, we implicitly assume that states contain all necessary information for the deterministic evaluation of constraint satisfaction.

Before we proceed, it is helpful to make a couple of remarks. First, we make explicit use of the word *conditional* when referring to constraints to highlight the fact that we allow constraints in a risk-bounded conditional plan to depend on the plan execution itself, something particularly useful for risk-bounded missions with sensing actions specified in cRMPL (Chapter 4). For a grounded example, please refer to Figures 1-13, 1-14 and 1-15 from Section 1.4, in which we show temporal networks where *constraint activation* - active constraints are those that must be satisfied by the conditional plan - depends on real-time sensor observations.

Second, it is important to note that we make no assumption about constraint

violations producing observable outcomes, such as causing plan execution to halt, as it is required in the cost-based handling of chance constraints in C-POMDP’s [Undurti and How, 2010, Poupart et al., 2015]. As we further explain in Section 3.3, assuming that constraint violations cause plan execution to terminate imposes modeling limitations that can lead to plan conservatism and overconfident behavior by the autonomous agent.

Last, even though we adopt the same risk semantics of [Ono and Williams, 2008, Ono et al., 2012b, Ono et al., 2012a], we should point out that their mathematical programming-based approach to risk-bounded *unconditional* planning resorts to a *static* allocation of risk to different segments of the mission, while our HFS-based approach for generating risk-bounded *conditional* plans leverages a *dynamic* risk allocation strategy developed in the following and later used by RAO*.

Back to the goal of developing a dynamic measure of risk for conditional plans, let b_k be a belief state as defined in Section 3.2.1, and let $Sa_k(C)$ (for “safe at time k ”) be a Bernoulli random variable denoting whether the system *has not violated* any constraints in C at time k . Following the previous semantics for *risk event*, we define the *execution risk* of a policy π measured from the belief b_k as

$$er(b_k, C|\pi) = 1 - \Pr\left(\bigwedge_{i=k}^h Sa_i(C) \middle| b_k, \pi\right), \quad (3.12)$$

where h is the potentially unknown planning horizon. Since the particular set of constraints C is not important, we omit the dependence of Sa_i on C in the following for notational convenience.

The probability term in (3.12) can be written as

$$\Pr\left(\bigwedge_{i=k}^h Sa_i \middle| b_k, \pi\right) = \Pr\left(\bigwedge_{i=k+1}^h Sa_i \middle| Sa_k, b_k, \pi\right) \Pr(Sa_k|b_k, \pi), \quad (3.13)$$

where $\Pr(Sa_k|b_k, \pi)$ is the probability of the system not being in a constraint-violating

state at the k -th time step. Since b_k is given, $\Pr(Sa_k|b_k, \pi)$ can be computed as

$$\Pr(Sa_k|b_k, \pi) = 1 - r_b(b_k, C), \quad (3.14)$$

where

$$r_b(b_k, C) = \sum_{s_k \in S} b(s_k) c_v(s_k, C) \quad (3.15)$$

is called the *risk* at b_k , as opposed to the *execution risk* at b_k in (3.12). Note that $c_v(s_k, C) = 1$ if, and only if, s_k violates constraints in C .

Once again for notational convenience, we will use the shorthand notation $r_b(b_k)$ and $er(b_k|\pi)$ to refer to $r_b(b_k, C)$ and $er(b_k, C|\pi)$. With these simplifications, the first probability term on the RHS of (3.13) can be written as

$$\Pr\left(\bigwedge_{i=k+1}^h Sa_i \middle| Sa_k, b_k, \pi\right) = \sum_{b_{k+1}} \Pr\left(\bigwedge_{i=k+1}^h Sa_i \middle| b_{k+1}, Sa_k, b_k, \pi\right) \Pr(b_{k+1}|Sa_k, b_k, \pi). \quad (3.16)$$

The summation in (3.16) is over posterior belief states at time $k+1$, which are dependent on the current action $a_k = \pi(b_k)$ and some corresponding observation o_{k+1} . Therefore, we have $\Pr(b_{k+1}|Sa_k, b_k, \pi) = \Pr(o_{k+1}|Sa_k, \pi(b_k), b_k)$. However, unlike (3.10) and (3.11), the conditioning on Sa_k in (3.16) means that its quantities, including the next belief states b_{k+1} , are computed under the assumption that the system is in a safe state at time k .

In order to compute $\Pr(o_{k+1}|Sa_k, a_k, b_k)$, it is useful to define *safe prior* belief

$$\bar{b}^{sa}(s_{k+1}|a_k) = \Pr(s_{k+1}|Sa_k, a_k, b_k) = \frac{\sum_{s_k: c_v(s_k, C)=0} T(s_k, a_k, s_{k+1}) b(s_k)}{1 - r_b(b_k)}, \quad (3.17)$$

which constructs a prior belief state at time $k+1$ according to T , the stochastic state transition function, by only considering the transitions from states s_k such that $c_v(s_k, C) = 0$ (do not violate constraints) and normalizing the belief by the probability

mass of safe states at time k . Note, however, that this *does not* imply $r_b(\bar{b}_{k+1}^{sa}) = 0$, since safe states s_k at time k can potentially transition to unsafe states s_{k+1} at time $k + 1$ according to T . With (3.17), we can define

$$\begin{aligned}
\Pr^{sa}(o_{k+1}|a_k, b_k) &= \Pr(o_{k+1}|Sa_k, a_k, b_k) \\
&= \sum_{s_{k+1}} \Pr(o_{k+1}|s_{k+1}, Sa_k, a_k, b_k) \Pr(s_{k+1}|Sa_k, a_k, b_k), \\
&= \sum_{s_{k+1}} O(s_{k+1}, o_{k+1}) \bar{b}^{sa}(s_{k+1}|a_k),
\end{aligned} \tag{3.18}$$

which is the distribution of observations at time $k + 1$ assuming that the system was in a non-violating state at time k .

Since the belief state b_{k+1} in $\Pr(b_{k+1}|Sa_k, b_k, \pi) = \Pr(o_{k+1}|Sa_k, \pi(b_k), b_k)$ is the same one appearing in $\Pr\left(\bigwedge_{i=k+1}^h Sa_i \middle| b_{k+1}, Sa_k, b_k, \pi\right)$ on the RHS in (3.16), we conclude that it corresponds to the *safe posterior* belief

$$\begin{aligned}
\hat{b}^{sa}(s_{k+1}) &= \Pr(s_{k+1}|o_{k+1}, Sa_k, a_k, b_k), \\
&\propto \Pr(o_{k+1}|s_{k+1}, Sa_k, a_k, b_k) \Pr(s_{k+1}|Sa_k, a_k, b_k), \\
&\propto O(s_{k+1}, o_{k+1}) \bar{b}^{sa}(s_{k+1}|a_k),
\end{aligned} \tag{3.19}$$

where the normalization constant is (3.18). Therefore, from (3.12), we can write

$$\Pr\left(\bigwedge_{i=k+1}^h Sa_i \middle| b_{k+1}, Sa_k, b_k, \pi\right) = (1 - er(b_{k+1}^{sa}|\pi)), \tag{3.20}$$

where $b_{k+1}^{sa} = \hat{b}_{k+1}^{sa}$. Combining the previous equations, we get

$$\begin{aligned}
er(b_k|\pi) &= 1 - \Pr\left(\bigwedge_{i=k}^h Sa_i \middle| b_k, \pi\right), \\
&= 1 - \Pr\left(\bigwedge_{i=k+1}^h Sa_i \middle| Sa_k, b_k, \pi\right) \Pr(Sa_k|b_k, \pi), \\
&= 1 - (1 - r_b(b_k)) \sum_{b_{k+1}^{sa}} \Pr\left(\bigwedge_{i=k+1}^h Sa_i \middle| b_{k+1}^{sa}, Sa_k, b_k, \pi\right) \Pr^{sa}(o_{k+1}|a_k, b_k),
\end{aligned}$$

$$\begin{aligned}
&= 1 - (1 - r_b(b_k)) \sum_{b_{k+1}^{sa}} (1 - er(b_{k+1}^{sa}|\pi)) \Pr^{sa}(o_{k+1}|\pi(b_k), b_k), \\
&= 1 - (1 - r_b(b_k)) \left(1 - \sum_{b_{k+1}^{sa}} \Pr^{sa}(o_{k+1}|\pi(b_k), b_k) er(b_{k+1}^{sa}|\pi) \right),
\end{aligned}$$

leading to

$$er(b_k|\pi) = r_b(b_k) - (1 - r_b(b_k)) \sum_{b_{k+1}^{sa}} \Pr^{sa}(o_{k+1}|\pi(b_k), b_k) er(b_{k+1}^{sa}|\pi), \quad (3.21)$$

which is key to CC-POMDP's and RAO*. If b_k is terminal, (3.21) simplifies to $er(b_k|\pi) = r_b(b_k)$. Prior to the formal definition of CC-POMDP's in the next section, it is worthwhile to make a couple of comments regarding (3.21).

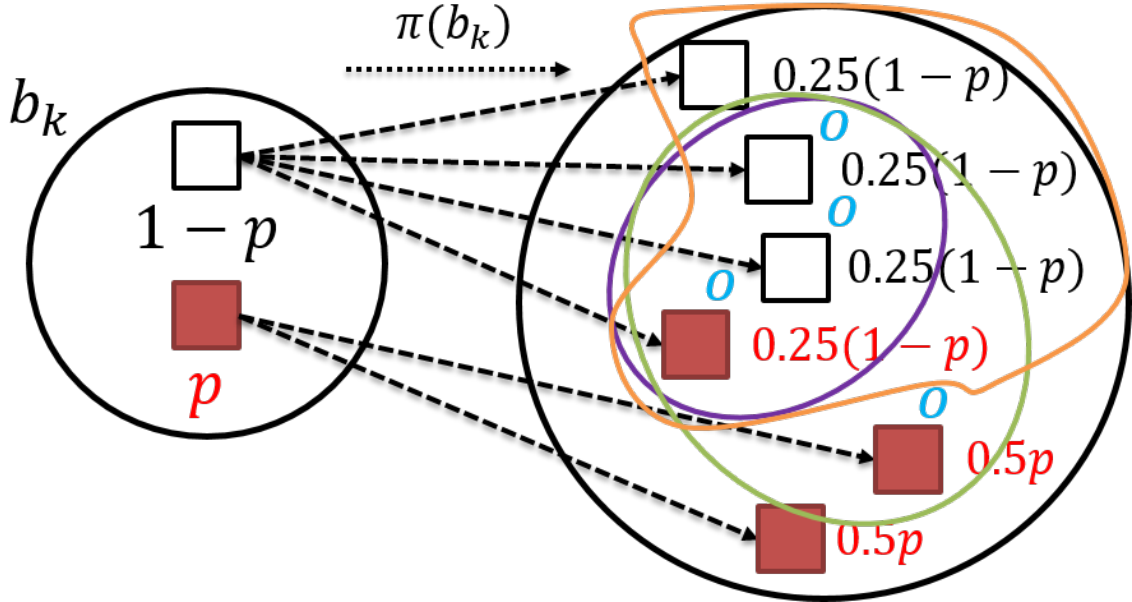


Figure 3-2: Simple graphical example of how safe belief states and observation probabilities are computed. Black circles represent belief states, and squares with probabilities shown next to them are belief state particles. Arrows emanating from particles represent stochastic transitions triggered by action $\pi(b_k)$ at the belief state b_k . Particles shown in red are those whose states s violate constraints, i.e., $c_v(s, C) = 1$, while white particles are *safe*. For this example, assume that the observation o_{k+1} shown in blue is generated with probability 1 by the particles next to it, and with probability 0 everywhere else.

First, note that the dependency on the potentially unknown planning horizon h

disappeared in (3.21).

Second, by requiring knowledge of $er(b_{k+1}^{sa}|\pi)$ for the computation of $er(b_k|\pi)$, (3.21) could be interpreted as a “Bellman backup” [Bellman, 1956] for risk. However, unlike the Bellman backup equation for plan utility in Section 3.4.2, which requires (3.11) in its averaging term, the execution risk in (3.21) uses the safe observation distribution in (3.18), and propagates forward the *safe posterior* belief in (3.19). In order to facilitate the understanding of how safe beliefs and observation distributions are computed, Figure 3-2 shows a simple graphical example. In this figure, particles composing the *safe prior* belief from (3.17) are surrounded by the orange line (all generated by the safe particle in b_k). According to (3.11), the probability of the belief state on the right generating the blue observation o_{k+1} is $\Pr(o_{k+1}|\pi(b_k), b_k) = 0.75 - 0.25p$ (particles circled by the green ellipse), while the probability for the same observation according to the *safe distribution* (3.18) is $\Pr^{sa}(o_{k+1}|\pi(b_k), b_k) = 0.75$ (particles circled by the purple ellipse).

Third, note that (3.21) requires RAO* to keep track of both \hat{b}_{k+1} , the posterior belief from (3.10) used for expected value computations, and \hat{b}_{k+1}^{sa} , the safe posterior belief from (3.19) used for execution risk propagation.

3.2.3 Chance-constrained POMDP’s

Combining the probabilistic state transition and observation model from Section 3.2.1 with the concept of *execution risk* from the previous section, we are ready to extend Section 1.3 and formally define CC-POMDP’s in Definition 3.1.

Definition 3.1 (Chance-Constrained POMDP). *A CC-POMDP is a tuple $H = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R, b_0, \mathcal{C}, c_v, \Delta \rangle$, where*

- \mathcal{S} , \mathcal{A} , and \mathcal{O} are, respectively, discrete sets of planning states, actions, and observations;
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a stochastic state transition function such that

$$T(s_k, a_k, s_{k+1}) = \Pr(s_{k+1}|s_k, a_k);$$

- $O : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ is a stochastic observation function such that

$$O(s_k, o_k) = \Pr(o_k | s_k);$$

- $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function;
- b_0 is the initial belief state;
- \mathcal{C} is a set of conditional constraints defined over \mathcal{S} ;
- $c_v = [c_v^1, \dots, c_v^q]$ is a vector of constraint violation indicators $c_v^i : \mathcal{S} \times 2^{\mathcal{C}} \rightarrow \{0, 1\}, i = 1, 2, \dots, q$, such that $c_v(s, C^i) = 1$ if, and only if, s violates constraints in a subset C^i of \mathcal{C} ;
- $\Delta = [\Delta^1, \dots, \Delta^q]$ is a vector of q execution risk bounds used to define q chance constraints

$$er(b_k, C^i | \pi) \leq \Delta^i, i = 1, 2, \dots, q, k \geq 0. \quad (3.22)$$

A solution to a CC-POMDP is an optimal *policy* (or *plan*) $\pi^* : \mathcal{B} \rightarrow \mathcal{A}$ mapping *belief states* in \mathcal{B} to actions in \mathcal{A} such that

$$\pi^* = \arg \max_{\pi} \mathbb{E} \left[\sum_{t=0}^h R(s_t, a_t) \middle| \pi \right] \quad (3.23)$$

and (3.22) holds.

Chance constraints in [Blackmore, 2007, Ono and Williams, 2008, Undurti and How, 2010, Ono et al., 2012b, Ono et al., 2012a, Poupart et al., 2015] are used to bound the probability of constraint violations over complete executions of π^* . In the execution risk notation of (3.22), that corresponds to chance constraints

$$er(b_0, C^i | \pi) \leq \Delta^i, C^i \in 2^{\mathcal{C}}, i = 1, 2, \dots, q, \quad (3.24)$$

which limit the execution risk from the initial belief state. However, enforcing (3.24) might entail unanticipated (although correct) “daredevil” behavior by the agent in the presence of unlikely observations, which might go against the safety expectations of the mission designer. Therefore, in the next section, we present alternative forms of chance constraints in execution risk notation that are guaranteed to entail safer autonomous behavior than (3.24), while also offering a perspective on how the static risk allocation of [Ono and Williams, 2008, Ono et al., 2012b, Ono et al., 2012a] relates to (3.21).

3.2.4 Enforcing safe behavior at all times

The chance constraints in (3.24) ensure that the probability of constraint violation is bounded over complete executions of the policy. However, that does not mean that the agent will exhibit the same sensitivity to risk at all execution steps. In fact, as shown in Section 3.4.1, unlikely policy branches can be allowed risks close or equal to 1 if that will help improve the objective, giving rise to a “daredevil” attitude. Since this might not be the desired risk-aware behavior, a straightforward way of achieving higher levels of safety is to depart from the chance constraints in (3.24) and, instead, impose a set of chance constraints of the form

$$er(b_k, C^i | \pi) \leq \Delta^i, \forall i, k, \text{ s.t. } b_k \text{ is nonterminal.} \quad (3.25)$$

Intuitively, (3.25) tells the autonomous agent to “remain safe at all times”, whereas the message conveyed by (3.24) is “stay safe overall”. It should be clear that (3.25) implies (3.24), so (3.25) necessarily generates safer policies than (3.24), but also more conservative in terms of utility. Another possibility is to follow [Ono et al., 2012b] and impose

$$\sum_{k=0}^h r_b(b_k, C^i) \leq \Delta^i, \forall i, \quad (3.26)$$

which is a sufficient condition for (3.24) based on Boole’s inequality. One can show

that (3.26) \Rightarrow (3.25), so enforcing (3.26) will lead to policies that are at least as conservative as (3.25).

3.3 Relation to constrained POMDP's

Alternative approaches for chance-constrained POMDP planning have been presented in [Undurti and How, 2010, Poupart et al., 2015], where the authors investigate constrained POMDP's (C-POMDP's). They argue that chance constraints can be modeled within the C-POMDP framework by assigning unit costs to states violating constraints, 0 to others, and performing calculations as usual.

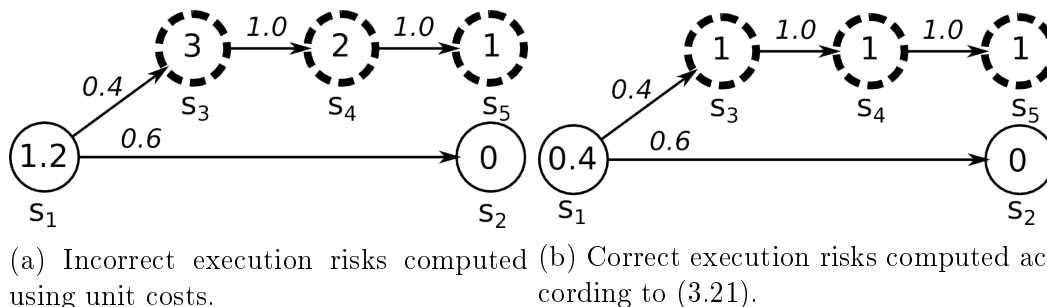


Figure 3-3: Modeling chance constraints via unit costs may yield incorrect results when constraint-violating states (dashed outline) are not terminal. Numbers within states are constraint violation probabilities. Numbers over arrows are probabilities for a non-deterministic action.

There are two main shortcomings associated with the use of unit costs to deal with chance constraints. First, it only yields correct measures of execution risk in the particular case where constraint violations cause policy execution to terminate. If that is not the case, incorrect probability values can be attained, as shown in the simple example in Figure 3-3. Second, assuming that constraint violations cause execution to cease has a strong impact on belief state computations, as shown in Figure 3-4. The key insight here is that, by assuming that constraint violations cause execution to halt, it provides the system with an invaluable observation: at each non-terminal belief state, the risk $r_b(b_k, C)$ in (3.14) must be 0. The reason for that is simple:

$$\text{constraint violation} \Rightarrow \text{termination} \Leftrightarrow \text{execution} \Rightarrow \text{no violation}. \quad (3.27)$$

Figure 3-4 illustrates the impact of (3.27) on belief states: an autonomous agent, starting from an initial belief state without uncertainty (it contains a single particle), executes a probabilistic operator leading to six possible future states according to (3.3). From these six, one of them, whose probability according to (3.7) is p_c , is marked red to symbolize that it violates plan constraints. Hence, according to (3.14), $r_b(\bar{b}_1) = p_c$. Even if the agent is devoid of any sensors, the assumption that constraint violations cause execution to halt creates a “virtual” binary sensor shown in Figure 3-4 outputting “Alive!” or “Dead!”. According to (3.11) and (3.27), “Alive!” and “Dead!” are generated with probabilities $1 - p_c$ and p_c , respectively, and give rise to different posterior belief states \hat{b}_1 . Since (3.27) requires constraint-violating states to generate “Dead!” with probability 1, the belief update step in (3.10) rules out the possibility of any particle being in a constraint violating state upon receiving “Alive!”, thus generating the belief state \hat{b}_1 at the top, for which $r_b(\hat{b}_1) = 0$. Through a similar reasoning, the belief update step with observation “Dead!” generates the posterior \hat{b}_1 at the bottom, which is terminal and has $r_b(\hat{b}_1) = 1$.

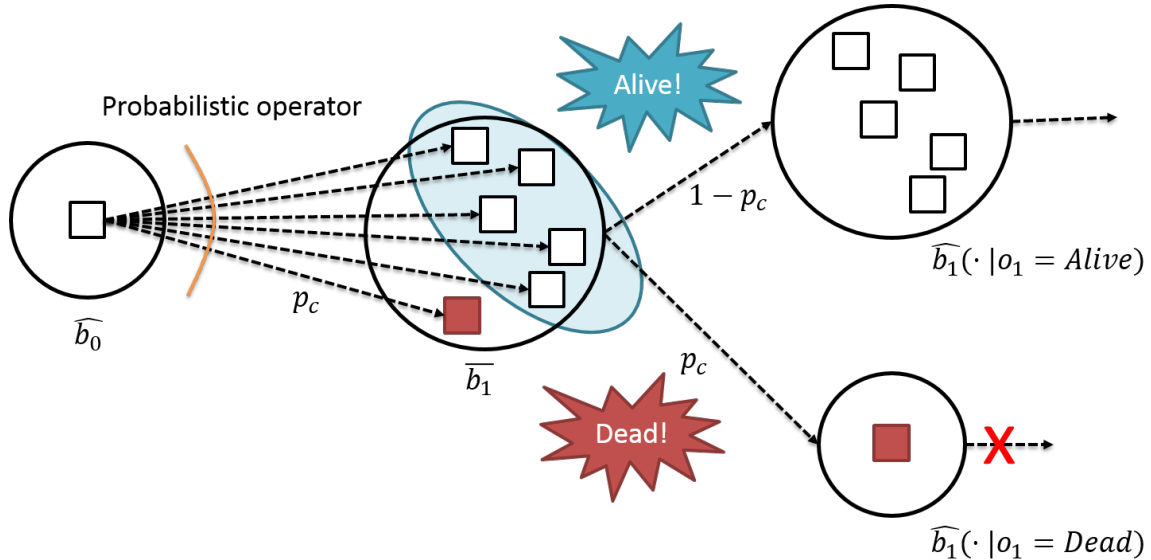


Figure 3-4: Impact on belief states of assuming terminal constraint violations, with squares representing belief state particles. White squares represent particles for which c_v indicates no constraint violation, while red squares denote particles on constraint-violating paths.

Assuming that constraint violations are terminal is reasonable when they are

destructive, e.g., when the agent is irrecoverably damaged after falling into a crater or flying through an obstacle, or forever lost after running out of power. Nevertheless, it is rather limiting in terms of the expressiveness of risk-aware models, and may have negative impacts on plan performance in situations where constraint violations are “benign” or truly unobservable. As an example of “benign” violations, consider the power supply restoration application referred to in Chapter 1 and further analyzed in Section 3.5. In this domain, chance constraints can be used to limit the probability of reconnecting networks faults to generators and causing further blackouts and stress across the network. As undesirable as it may be, reconnecting faults to generators, however, does not necessarily destroy the network. In fact, it might be the only way to “probe” the system in order to significantly reduce the uncertainty about the location of a fault, which in turn allows for a larger amount of power to be restored to the consumers. Therefore, as confirmed by the results in Section 3.5, this is a situation in which assuming terminal constraint violations leads to *conservatism*.

Related to the previous point about non-terminal constraint violations in CC-POMDP models, it is important to stress that, apart from having an impact on the execution risk of a conditional plan, *no special treatment* is required for the handling of plan execution after one or more constraints are violated. In fact, any changes in behavior that one wishes to trigger upon plan constraint violations must be directly encoded as features in the CC-POMDP model, such as modifications to the objective function or the actions available to the autonomous agent. For instance, in the power supply restoration domain from the previous paragraph, belief states before and after constraint violations (generating further blackouts) are treated in the exact same way, while the planetary rover domains in Sections 3.5 and 7.3.1 assume that obstacle collisions are terminal, and enforce this property by restricting the set of available actions to the autonomous agent upon collision to be the empty set.

Regarding the dangers of assuming terminal constraint violations in situations where they are unobservable, consider the example of a Mars rover that must limit the probability of motor overheating as it executes traversals, for that causes components to degrade faster. In addition, assume that the temperature sensor for one of the

wheels has already incurred significant damage and can no longer properly detect overheating. Through the process illustrated in Figure 3-4, the assumption in (3.27) causes the autonomous agent to essentially ignore the possibility of motor overheating in that wheel, leading to *overconfident* behavior. Handling of chance constraints in CC-POMDP's using (3.21) does not suffer from either of these shortcomings.

3.4 Solving CC-POMDP's through RAO*

In this section, we introduce the Risk-bounded AO* algorithm (RAO*) for constructing risk-bounded policies for CC-POMDP's. RAO* is based on heuristic forward search in the space of belief states. The motivation for this is simple and shown in Figure 3-5: given an initial belief state and limited resources (including time), the belief states that are *reachable* from an initial belief b_0 are usually a small fraction of the complete set of beliefs \mathcal{B} .

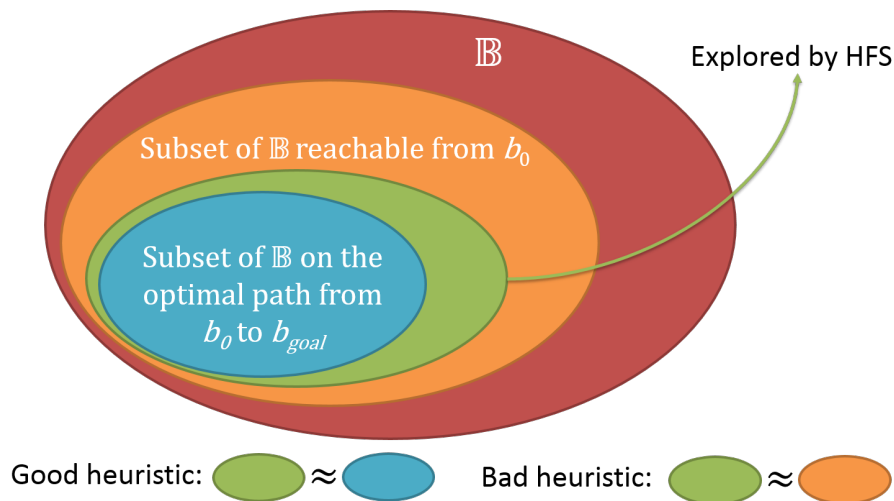


Figure 3-5: Relationship between spaces explored by heuristic forward search.

Similar to AO* in fully observable domains, RAO* explores its search space from an initial belief b_0 by incrementally constructing a hypergraph G , called the *explicit* hypergraph. As shown in Figure 3-6, nodes in G contain posterior belief states - both (3.10) and (3.19) -, and a hyperedge is a compact representation of the process of taking an action and receiving several possible observations in an AND-OR tree

(Figure 3-7). Figure 3-8 shows the hypergraph representation of the AND-OR tree segment in Figure 3-7.

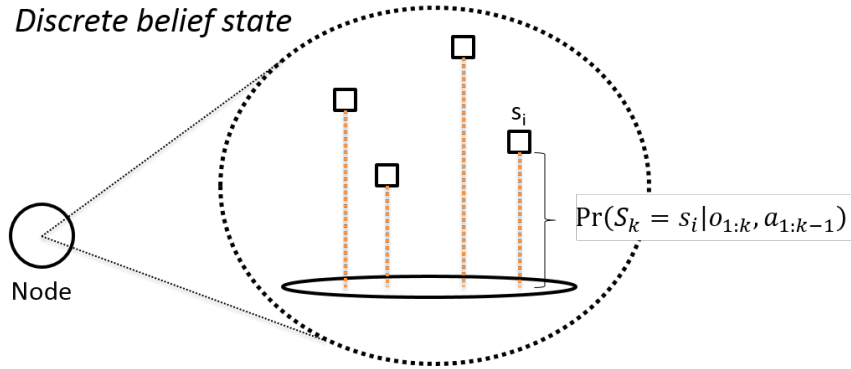


Figure 3-6: Hypergraph node containing a belief state.

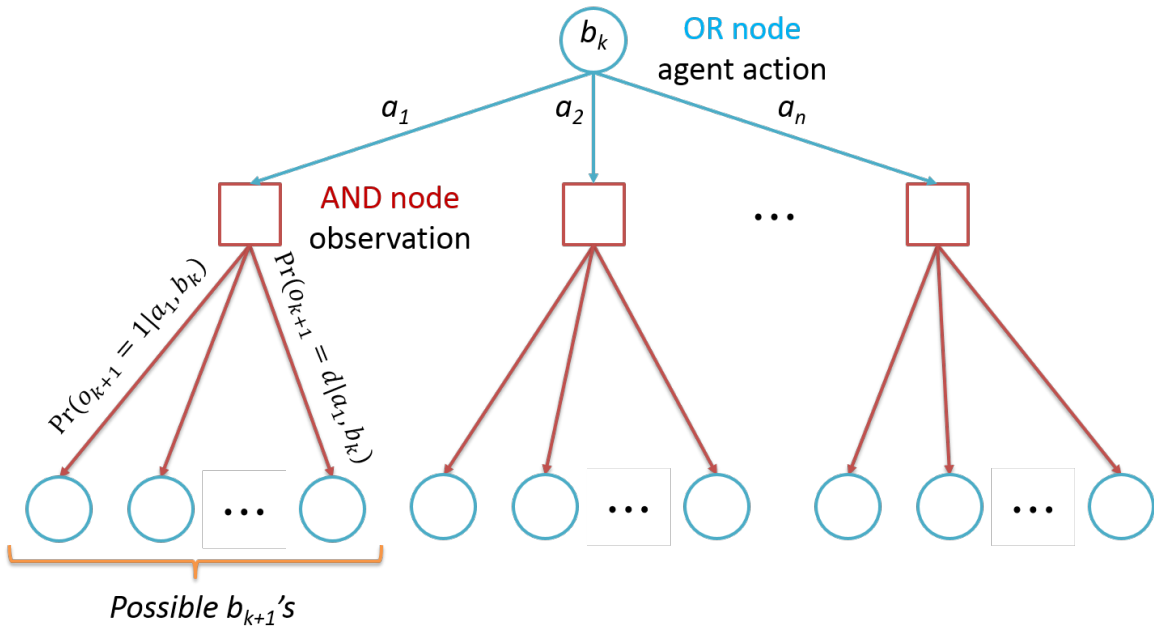


Figure 3-7: Segment of an AND-OR search tree.

3.4.1 Propagating risk bounds forward

As previously mentioned in Section 3.2.2, the direction of execution risk computation in (3.21) is “backwards”, i.e., execution risk propagation happens from child to parent nodes on the search hypergraph constructed by RAO*. However, since we propose to

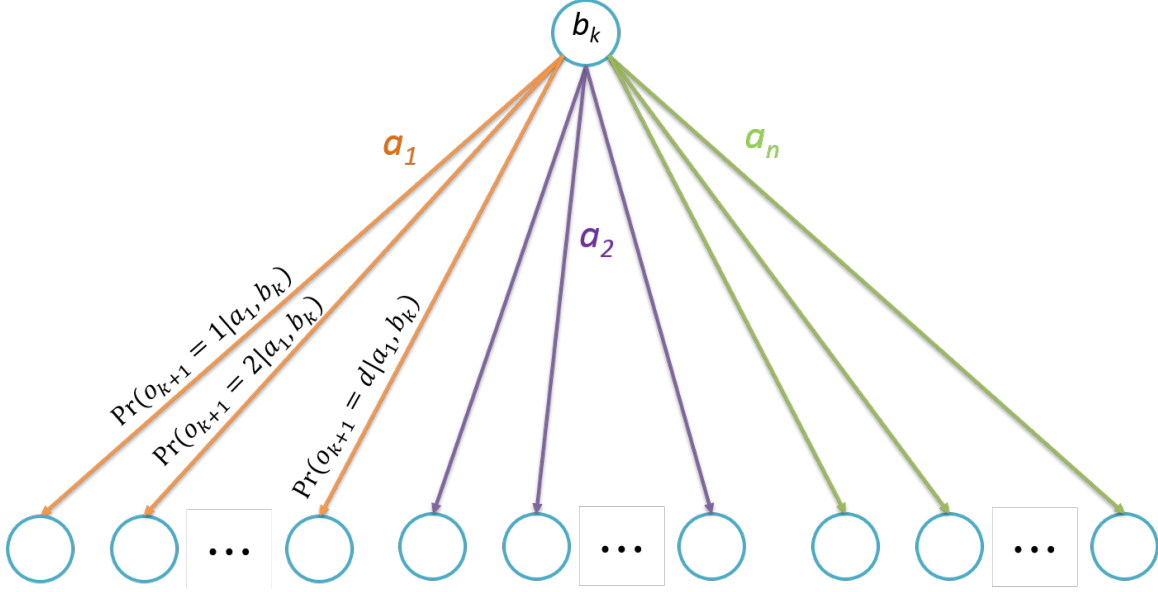


Figure 3-8: Hypergraph representation of an AND-OR tree.

compute CC-POMDP policies by means of heuristic forward search, one should seek ways to propagate estimates of (3.21) forward so as to be able to quickly detect that the current best policy is too risky. For that, let $0 \leq \Delta_k \leq 1$ be a bound for which $er(b_k|\pi) \leq \Delta_k$ must be enforced. Also, let o'_{k+1} be the observation associated with child b'_{k+1} of b_k and with probability $\Pr^{sa}(o'_{k+1}|\pi(b_k), b_k) \neq 0$. From (3.21) and the condition $er(b_k|\pi) \leq \Delta_k$, we get

$$er(b'_{k+1}|\pi) \leq \frac{\left(\frac{\Delta_k - r_b(b_k)}{1 - r_b(b_k)} - \sum_{o_{k+1} \neq o'_{k+1}} \Pr^{sa}(o_{k+1}|\pi(b_k), b_k) er(b'_{k+1}|\pi) \right)}{\Pr^{sa}(o'_{k+1}|\pi(b_k), b_k)}. \quad (3.28)$$

The existence of (3.28) requires $r_b(b_k) < 1$ and $\Pr^{sa}(o'_{k+1}|\pi(b_k), b_k) \neq 0$ whenever $\Pr(o'_{k+1}|\pi(b_k), b_k) \neq 0$. Lemma 3.1 shows that these conditions are equivalent.

Lemma 3.1. *One observes $\Pr^{sa}(o_{k+1}|\pi(b_k), b_k) = 0$ and $\Pr(o_{k+1}|\pi(b_k), b_k) \neq 0$ if, and only if, $r_b(b_k) = 1$.*

Proof:

\Leftarrow : if $r_b(b_k) = 1$, we conclude from (3.14) that $c_v(s_k, C) = 1, \forall s_k$. Hence, all elements in (3.17) and, consequently, (3.18) will have probability 0.

\Rightarrow : from Bayes' rule, we have

$$\Pr(Sa_k|o_{k+1}, a_k, b_k) = \frac{\Pr^{sa}(o_{k+1}|a_k, b_k)(1 - r_b(b_k))}{\Pr(o_{k+1}|a_k, b_k)} = 0$$

Hence, we conclude that $\Pr(\neg Sa_k|o_{k+1}, a_k, b_k) = 1$, i.e., the system is guaranteed to be in a constraint-violating state at time k , yielding $r_b(b_k) = 1$. \square

The execution risk of nodes whose parents have $r_b(b_k) = 1$ is irrelevant, as shown by (3.21). Therefore, it only makes sense to propagate risk bounds in cases where $r_b(b_k) < 1$.

One difficulty associated with (3.28) is that it depends on the execution risk of all siblings of b'_{k+1} , which cannot be computed exactly until terminal nodes are reached. Therefore, one must approximate (3.28) in order to render it computable during forward search.

We can easily define a necessary condition for feasibility of a chance constraint at a search node by means of an admissible execution risk heuristic $h_{er}(b_{k+1}^{sa}|\pi) \leq er(b_{k+1}^{sa}|\pi)$. An admissible estimate $h_{er}(b_{k+1}^{sa}|\pi)$ of $er(b_{k+1}^{sa}|\pi)$ can be constructed from a state heuristic $h_{er}(s_{k+1}^{sa}|\pi)$ by taking the average

$$h_{er}(b_{k+1}^{sa}|\pi) = \sum_{s_{k+1}} b^{sa}(s_{k+1}) h_{er}(s_{k+1}|\pi). \quad (3.29)$$

Combining $h_{er}(b_{k+1}^{sa}|\pi)$ and (3.28) provides us with a necessary condition

$$er(b_{k+1}^{sa}|\pi) \leq \Delta'_{k+1} = \frac{\left(\frac{\Delta_k - r_b(b_k)}{1 - r_b(b_k)} - \sum_{o_{k+1} \neq o'_{k+1}} \Pr^{sa}(o_{k+1}|\pi(b_k), b_k) h_{er}(b_{k+1}^{sa}|\pi) \right)}{\Pr^{sa}(o'_{k+1}|\pi(b_k), b_k)}. \quad (3.30)$$

Since $h_{er}(b_{k+1}^{sa}|\pi)$ computes a lower bound on the execution risk, we conclude that (3.30) gives an upper bound for the true execution risk bound in (3.28). The simplest possible heuristic is $h_{er}(b_{k+1}^{sa}|\pi) = 0$, which assumes that it is absolutely safe to continue executing policy π beyond b_k . Moreover, from the non-negativity of the

terms in (3.21), we see that another possible choice of a lower bound is $h_{er}(b_{k+1}^{sa}|\pi) = r_b(b_{k+1}^{sa})$, which is still myopic, but guaranteed to be an improvement over the previous zero risk heuristic, for it incorporates additional information about the risk of failure at that belief state. All these bounds can be compute forward, starting with $\Delta_0 = \Delta$. Figure 3-9 uses different colors to show how the terms of (3.30) relate to the portion of the hypergraph constructed by RAO* associated with executing action $a_k = \pi(b_k)$ at belief b_k .

$$\Delta'_{k+1} = \frac{1}{Pr^{sa}(o'_{k+1}|\pi(b_k), b_k)} \left(\frac{\Delta_k - r_b(b_k)}{1 - r_b(b_k)} - \sum_{o_{k+1} \neq o'_{k+1}} Pr^{sa}(o_{k+1}|\pi(b_k), b_k) h_{er}(b_{k+1}^{sa}|\pi) \right)$$

Figure 3-9: Visual relationship between (3.30) and the portion of RAO*'s search hypergraph associated with executing action $a_k = \pi(b_k)$ at belief b_k .

This section's discussion of execution risk propagation has focused on the particular case of a single chance constraint. However, as seen in Definition 3.1 (Section 3.2.3), a CC-POMDP may contain multiple chance constraints $er(b_k, C^i|\pi) \leq \Delta^i, i = 1, 2, \dots, q, k \geq 0$, bounding the probability of violating different subsets of constraints $C^i \in 2^{\mathcal{C}}$ during policy execution. Propagating multiple chance constraints forward is a simple extension of the procedure used for a single chance constraint: for each subset of constraints C^i and its corresponding execution risk bound Δ^i , compute the risk term r_b^i in (3.15) using the corresponding constraint violation function c_v^i . Moreover, use c_v^i and equations (3.17), (3.18), and (3.19) to compute $b_{k+1}^{sa,i}$ and $Pr^{sa,i}$ in (3.30). Finally, for each subset of constraints C^i , define a corresponding execution risk heuristic h_{er}^i . An RAO* policy that violates *any* of the execution risk bounds given by (3.30) for each constraint subset C^i is invalid and should be immediately pruned.

3.4.2 Algorithm

Each node in the hypergraph G constructed by RAO* (Algorithm 3.1) is associated with a Q value

$$Q(b_k, a_k) = \sum_{s_k} R(s_k, a_k) b(s_k) + \sum_{o_{k+1}} \Pr(o_{k+1} | a_k, b_k) Q^*(b_{k+1}) \quad (3.31)$$

representing the expected, cumulative reward of taking action a_k at some belief state b_k . The first term corresponds to the expected current reward, while the second term is the expected reward obtained by following the optimal deterministic policy π^* , i.e., $Q^*(b_{k+1}) = Q(b_{k+1}, \pi^*(b_{k+1}))$. Similar to (3.29), an admissible estimate $h_Q(b_{k+1})$ of $Q^*(b_{k+1})$ can be constructed from a state heuristic $h_Q(s_{k+1})$ by taking the average

$$h_Q(b_{k+1}) = \sum_{s_{k+1}} b(s_{k+1}) h_Q(s_{k+1}). \quad (3.32)$$

Given $h_Q(b_{k+1})$, we select actions for the current estimate $\hat{\pi}$ of π^* according to

$$\hat{\pi}(b_k) = \arg \max_{a_k} \hat{Q}(b_k, a_k), \quad (3.33)$$

where

$$\hat{Q}(b_k, a_k) = \sum_{s_k} R(s_k, a_k) b(s_k) + \sum_{o_{k+1}} \Pr(o_{k+1} | a_k, b_k) h_Q(b_{k+1}) \quad (3.34)$$

is the same as (3.31) with $Q^*(b_{k+1})$ replaced by $h_Q(b_{k+1})$. The portion of G corresponding to the current estimate $\hat{\pi}$ of π^* is called the *greedy* graph, for it uses an admissible heuristic estimate $h_Q(b_k, a_k)$ of $Q^*(b_{k+1})$ to explore the most promising areas of G first.

The most important differences between AO* and RAO* lie in Algorithms 3.2 and 3.3. First, since RAO* deals with partially observable domains, node expansion in Algorithm 3.2 involves full Bayesian prediction and update steps, as opposed to a simple branching using the state transition function T . In addition, RAO* leverages

Algorithm 3.1: RAO***Input:** CC-POMDP H , initial belief b_0 .**Output:** Optimal policy π mapping beliefs to actions.

```
1 Function RAO*( $H, b_0$ )
2   Explicit graph  $G$  and policy  $\pi$  initially consist of  $b_0$ .
3   while  $\pi$  has some nonterminal leaf node do
4      $n, G \leftarrow$  expand-policy( $G, \pi$ )
5      $\pi \leftarrow$  update-policy( $n, G, \pi$ )
6   return  $\pi$ .
```

Algorithm 3.2: expand-policy**Input:** Explicit graph G , policy π .**Output:** Expanded explicit G' , expanded leaf node n .

```
1 Function expand-policy( $G, \pi$ )
2    $G' \leftarrow G, n \leftarrow$  choose-promising-leaf( $G, \pi$ )
3   for each action  $a$  available at  $n$  do
4      $ch \leftarrow$  use (3.7), (3.10), (3.11) to expand children of  $(n, a)$ .
5      $\forall c \in ch$ , use (3.14), (3.18), (3.21), and (3.31) with admissible heuristics
6     to estimate  $Q^*$  and  $er$ .
7      $\forall c \in ch$ , use (3.30) to compute execution risk bounds
8     if no  $c \in ch$  violates its risk bound then
9        $G' \leftarrow$  add hyperedge  $[(n, a) \rightarrow ch]$ 
10  if no action added to  $n$  then
11  mark  $n$  as terminal.
12  return  $G', n$ .
```

the heuristic estimates of execution risk explained in Section 3.4.1 in order to perform early pruning of actions that introduce child belief nodes that are guaranteed to violate the chance constraint. The same process is also observed during policy update in Algorithm 3.3, in which heuristic estimates of the execution risk are used to prevent RAO* to keep choosing actions that are promising in terms of heuristic value, but can be proven to violate the chance constraint at an early stage.

The search process explained in Algorithms 3.1, 3.2, and 3.3 can be summarized in three steps:

1. **Solution expansion:** takes a subset (which could be all) of the nonterminal nodes in the policy graph g and expands them. A node is *expanded* by con-

Algorithm 3.3: update-policy

Input: Expanded n , explicit graph G , policy π .

Output: Updated policy π' .

```
1 Function update-policy( $n, G, \pi$ )
2    $Z \leftarrow$  set containing  $n$  and its ancestors reachable by  $\pi$ .
3   while  $Z \neq \emptyset$  do
4      $n \leftarrow$  remove( $Z$ ) node  $n$  with no descendant in  $Z$ .
5     while there are actions to be chosen at  $n$  do
6        $a \leftarrow$  next best action at  $n$  according to (3.33) satisfying execution
7       risk bound.
8       Propagate execution risk bound of  $n$  to the children of the
9       hyperedge  $(n, a)$ 
10      if no children violates its exec. risk bound then
11      |  $\pi(n) \leftarrow a$ ; break
12    if no action was selected at  $n$  then
13    | mark  $n$  as terminal
```

structuring hyperedges associated with the actions available at that node, and using heuristics to estimate both value and execution risk of the child nodes connected to these hyperedges;

2. **Value/risk update:** propagate the heuristic estimates of value and execution risk from newly-expanded nodes “up” the explicit graph G ;
3. **Policy update:** use the updated estimates of value and execution risk in G to update the policy graph g .

Termination of this three-step process is attained when the policy graph g has no more nonterminal nodes to be expanded, in which case g is the optimal, chance-constrained, deterministic policy that solves the CC-POMDP.

When constructing the explicit graph G incrementally, an important subproblem in RAO* is how to handle duplicate nodes. If one forces all newly-expanded nodes in G to be treated as unique, G devolves into a *hypertree*, i.e., a tree with hyperedges, instead of remaining as a directed, acyclic hypergraph. Turning G into a hypertree does not affect RAO*’s soundness and completeness, and eliminates the overhead involved in checking if a newly-created search node is already part of G . However,

treating instances of the same node as separate entities may cause RAO* to perform *repeated computations*, as it is forced to expand the same hypertree spanning from different instances of the same node.

Although interesting from a theoretical standpoint, choosing to treat G as a hypergraph or a hypertree can be of little to no relevance in practice, since many CC-POMDP planning domains, especially those involving state uncertainty and partial observability, will naturally cause G to become a hypertree even if one spends the effort to check if newly-expanded nodes have already been added to G . This is because, in order for two nodes to be deemed the same, it must be the case that

- they share the beliefs \hat{b}_{k+1} and \hat{b}_{k+1}^{sa} computed, respectively, with (3.10) and (3.19);
- they have the same execution risk bound, which is in turn a function of a node's siblings according to a hyperedge.

An exception to this rule are fully observable CC-POMDP planning domains, where the task of comparing belief states is simplified by the fact that all beliefs contain a single particle with probability 1, in which case beliefs can be compared by just comparing if their single states are the same. However, outside this particular case, the hashing operations that are usually involved in testing if a node is already part of G almost never detect duplications, and their overhead can be easily eliminated by forcing G to be a hypertree.

Seeking to provide a clearer understanding of RAO*'s inner workings, the next section presents a few iterations of the algorithm on a simple grounded example.

3.4.3 Grounded example

For the sake of focusing our attention on the process of incremental construction of the explicit graph G and the policy graph g by RAO*, this example directly provides numbers for hyperedge probabilities and heuristic estimates of value and risk, whose computation is the topic of previous sections of this chapter. Moreover, this example

uses negative reward values to represent a CC-POMDP instance in which the goal is to minimize cost. Finally, Figure 3-10 shows the graphical representation of different elements involved in RAO*'s search for a policy in this particular grounded example.



Figure 3-10: From left to right: node in g , the greedy graph; node in G , the explicit graph, but not in g ; node with $r_b = 1$ (guaranteed to violate constraints); color used to represent heuristic estimates. In opposition to nodes with red outlines, we assume in this particular example that nodes with black outlines have $r_b = 0$.

Let Figure 3-11 represent the initial state of RAO*'s search for a CC-POMDP policy. It consists of the node representing the initial belief state b_0 in Definition 3.1, along with the set of open (nonterminal) policy nodes, and the initial state of the policy graph g . Since the search has not yet started, the only open node is b_0 itself, and the policy g makes no optimal operator assignment to b_0 . Figure 3-11 also shows that a chance constraint $er(b_0, C|\pi) \leq \Delta = 5\%$ is imposed on this search problem, where C is some given set of state constraints.

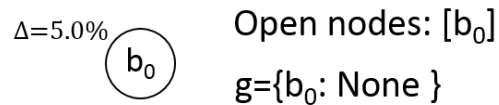


Figure 3-11: Initial state of the search for a CC-POMDP policy featuring a chance constraint $er(b_0, C|\pi) \leq \Delta = 5\%$.

Following the algorithm from the previous section, the first step in RAO* is to expand a nonterminal node from Figure 3-11. Since the only open node is the one containing b_0 , it is selected for expansion, and the result of this expansion is shown in Figure 3-12.

In this figure, two hyperedges emanate from the initial belief state b_0 : action a_1 is associated with the red hyperedge on the left, while action a_2 is associated with the orange hyperedge on the right. The numbers inside the newly-expanded child nodes are their heuristic value estimates h_Q . For the heuristic execution risk estimates given by h_{er} , we assume them to be the simple admissible heuristic $h_{er} = r_b$. Since all

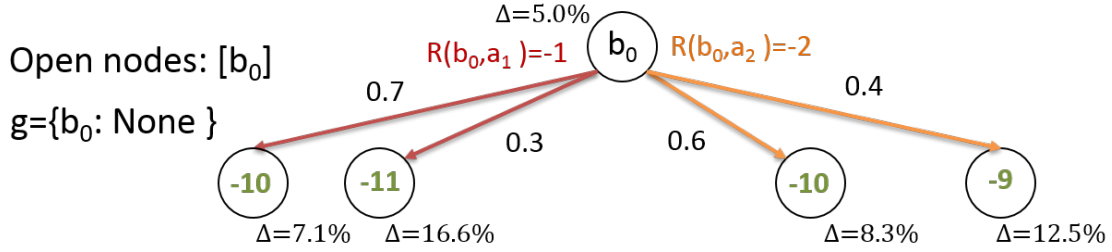


Figure 3-12: State of the search after expanding the initial belief state and propagating execution risk bounds forward.

newly-expanded child nodes have black outlines, we conclude from Figure 3-10 that all nodes have $r_b = 0$. The last step in the expansion step of RAO* is to propagate execution risk bounds forward according to (3.30). Taking the node with heuristic value estimate of -10 in Figure 3-12 as an example, its execution risk bound of 8.3% is obtained as follows:

$$\begin{aligned} \Delta'_1 &= \frac{\left(\frac{\Delta_0 - r_b(b_0)}{1 - r_b(b_0)} - \sum_{o_1 \neq o'_1} \text{Pr}^{sa}(o_1 | \pi(b_0), b_0) h_{er}(b_1^{sa} | \pi) \right)}{\text{Pr}^{sa}(o'_1 | \pi(b_0), b_0)}, \\ &= \frac{\left(\frac{0.05 - 0}{1 - 0} - 0.4 \times 0 \right)}{0.6} = 8.3\%. \end{aligned}$$

With the *solution expansion* step concluded, RAO* proceeds to the *value/risk update* step. According to Figure 3-12, we have

$$Q(b_0, a_1) = -1 + (-10 \times 0.7 - 11 \times 0.3) = -11.3,$$

$$Q(b_0, a_2) = -2 + (-10 \times 0.6 - 9 \times 0.4) = -11.6,$$

and the fact that $er(b_0, a_1) = er(b_0, a_2) = 0 < 5\%$ shows that neither a_1 , nor a_2 , can be pruned at this point on the grounds of being too risky. Since the goal is to minimize cost (or, equivalently, maximize the cumulative sum of negative rewards), action a_1 is marked as the most promising at node b_0 .

The last step in the iteration is to *update the policy* graph g in light of the newly-

expanded nodes and their value and risk estimates. Since the most promising action to be taken at b_0 according to Figure 3-12 is a_1 and it does not cause the execution risk bound at b_0 to be violated, the left portion of G is selected as the new best estimate of g , as shown in Figure 3-13.

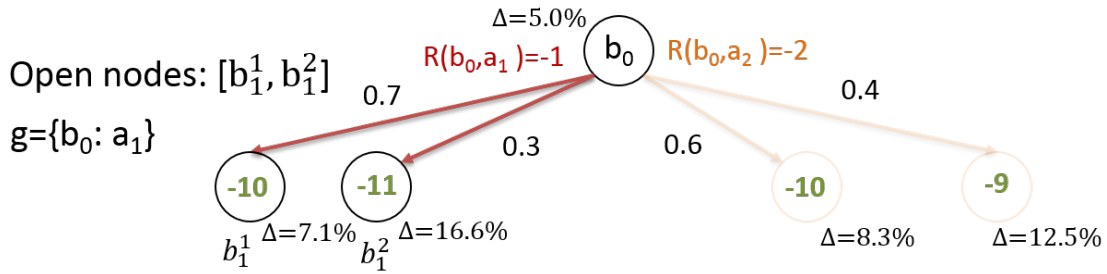


Figure 3-13: Outcome of updating the policy graph based on the numbers shown in Figure 3-12. The highlighted portion of the explicit graph G corresponds to the best available estimate of the optimal policy graph g .

With the previous iteration concluded, we are back to the solution expansion phase. According to Figure 3-13, either b_1^1 , b_1^2 , or even both could be selected for expansion. In this example, we arbitrarily choose to just expand b_1^2 , and the result is shown in Figure 3-14

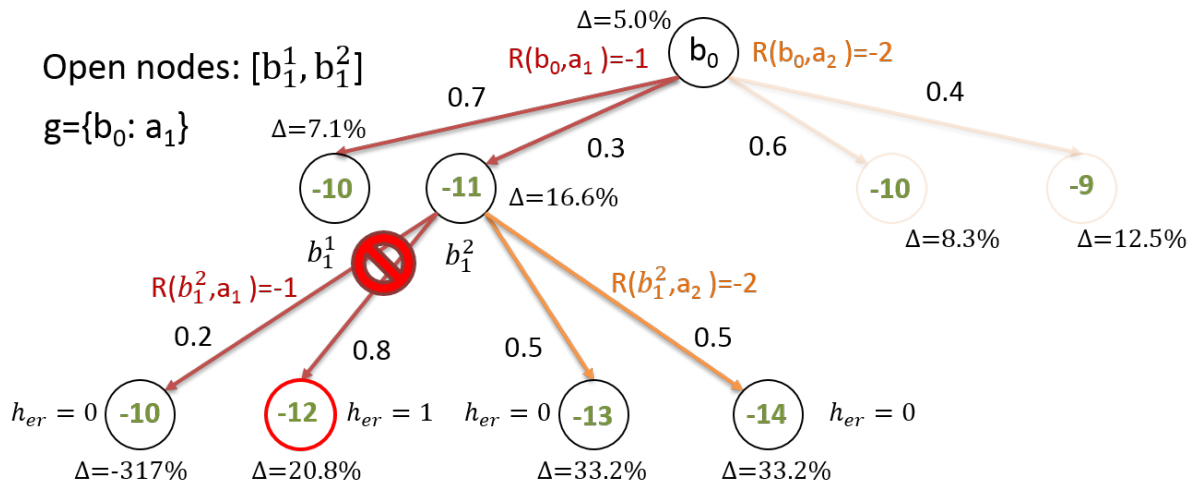


Figure 3-14: Result of expanding node b_1^2 in Figure 3-13. Notice the child node with red outline and $r_b = 1$, which causes the left hyperedge of b_1^2 to be pruned on the grounds of being too risky.

Similar to Figure 3-12, heuristic estimates of value and risk are computed during

the node expansion step in Figure 3-14. Different from Figure 3-12, however, notice the node with red outline symbolizing a belief state such that $r_b = 1$. Despite this difference, forward propagation of execution risk bounds according to (3.30) proceeds as usual. Focusing our attention on the left hyperedge associated with action a_1 , the left child receives the execution risk bound

$$\Delta'_2 = \frac{\left(\frac{0.166 - 0}{1 - 0} - 0.8 \times 1\right)}{0.2} = -317\%,$$

while the right child is assigned the execution risk bound

$$\Delta'_2 = \frac{\left(\frac{0.166 - 0}{1 - 0} - 0.2 \times 0\right)}{0.8} = 20.8\%.$$

Satisfying the negative execution risk bound in Figure 3-14 is clearly impossible, for execution risks are probabilities and, therefore, must remain within $[0, 1]$. Since satisfying the bounds computed by (3.30) is a *necessary* condition for chance constraint feasibility, the fact that one of the children violates their bounds is sufficient grounds for pruning a_1 at node b_1^2 , yielding the situation shown in Figure 3-15. Note that the child with $\Delta'_2 = 20.8\%$ is also in violation of the execution risk bound, since its admissible execution risk heuristic is $h_{er} = r_b = 100\% > 20.8\%$.

With a single hyperedge left at node b_1^2 , the *value/risk update* step becomes

$$Q(b_1^2, a_2) = -2 + (-13 \times 0.5 - 14 \times 0.5) = -15.5,$$

$$Q(b_0, a_1) = -1 + (-10 \times 0.7 - 15.5 \times 0.3) = -12.65,$$

$$Q(b_0, a_2) = -2 + (-10 \times 0.6 - 9 \times 0.4) = -11.6,$$

and $er(b_1^2, a_2) = 0 < 16.6\%$, $er(b_0, a_1) = er(b_0, a_2) = 0 < 5\%$. Since it is now a_2 the action that looks most promising at b_0 , the portion of G selected as the new best estimate of g is the one shown in Figure 3-16. This process is repeated until there are no more open nodes in g to be expanded, in which case RAO* returns g as the

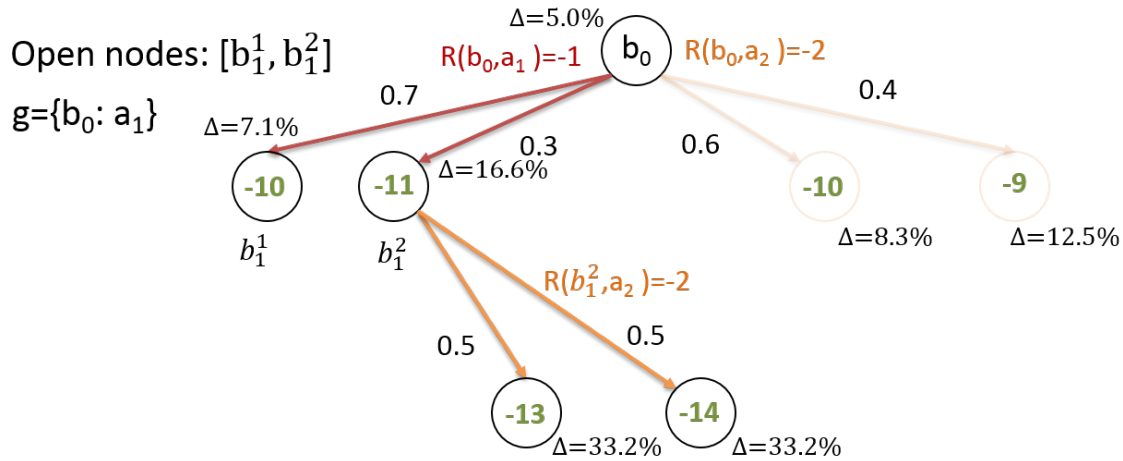


Figure 3-15: Result of pruning the search graph in Figure 3-14 due to violations of execution risk bounds.

optimal mapping from belief states to actions.

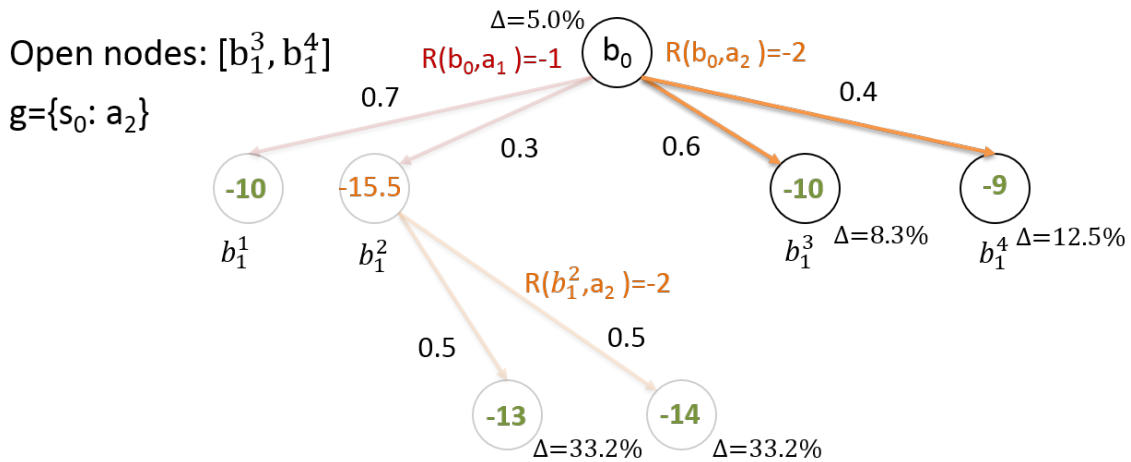


Figure 3-16: Estimate of the best policy g after two full iterations of RAO* on this simple example.

Before we conclude, it is worthwhile to mention that, had the constraint violation in Figure 3-14 happened on a hyperedge branch with probability less than 16.6%, action a_1 would not have been pruned at node b_1^2 .

3.4.4 Properties

The proofs of soundness, completeness, and optimality for RAO* are given in Lemma 3.2 and Theorem 3.1.

Lemma 3.2. *Risk-based pruning of actions in Algorithms 3.2 (line 6) and 3.3 (line 7) is sound.*

Proof: The RHS of (3.28) is the true execution risk bound for $er(b_{k+1}^{sa'}|\pi)$. The execution risk bound on the RHS of (3.30) is an upper bound for the bound in (3.28), since we replace $er(b_{k+1}^{sa}|\pi)$ for the siblings of $b_{k+1}^{sa'}$ by admissible estimates (lower bounds) $h_{er}(b_{k+1}^{sa}|\pi)$. In the aforementioned pruning steps, we compare $h_{er}(b_{k+1}^{sa'}|\pi)$, a lower bound on the true value $er(b_{k+1}^{sa'}|\pi)$, to the upper bound (3.30). Verifying $h_{er}(b_{k+1}^{sa'}|\pi) > (3.30)$ is sufficient to establish $er(b_{k+1}^{sa'}|\pi) > (3.28)$, i.e., action a currently under consideration is guaranteed to violate the chance constraint. \square

Theorem 3.1. *RAO* is complete and produces the optimal deterministic, finite-horizon policies meeting the chance constraints.*

Proof: a CC-POMDP, as described in Definition 3.1, has a finite number of policy branches, and Lemma 3.2 shows that RAO* only prunes policy branches that are guaranteed not to be part of any chance-constrained solution. Therefore, if no chance-constrained policy exists, RAO* will eventually return an empty policy.

Concerning the optimality of RAO* with respect to the utility function, it follows from the admissibility of $h_Q(b_k, a_k)$ in (3.33) and the optimality guarantee of AO*. \square

The computational complexity of RAO* (Algorithm 3.1) is strongly dependent on the quality of h_Q and h_{er} , the value and execution risk heuristics, the amount of pruning of the policy graph caused by the risk bounds Δ , and the specific parameters of the CC-POMDP given as input. At the same time, most of the effort in Algorithm 3.1 is spent expanding the explicit search graph G in line 4, while the policy update step in line 5 represents a small overhead on top of each expansion. Therefore,

seeking to provide an approximate characterization of the *worst case computational complexity* of RAO*, here we focus on the largest possible size of the explicit graph G as a function of the model, and the complexity of performing belief state updates during the process of expanding G .

Keeping in mind the hypergraph segment shown in Figure 3-8, let $|\mathcal{S}|$, $|\mathcal{A}|$, and $|\mathcal{O}|$ denote, respectively, the sizes of the discrete state, action, and observation spaces of the CC-POMDP given as input to RAO*. Moreover, let h be the finite execution horizon of the policy generated by RAO*, measured as the maximum depth (distance from root) of the nodes in G . For each graph node in G , at most $|\mathcal{A}|$ hyperedges representing available actions can emanate from it, each one with at most $|\mathcal{O}|$ edges representing particular observations. Therefore, for each expanded node, there are at most $|\mathcal{A}||\mathcal{O}|$ new children, and a maximum of

$$(|\mathcal{A}||\mathcal{O}|)^h \tag{3.35}$$

nodes in G for a planning horizon h .

Moreover, since RAO* operates over belief states, each newly-expanded child requires belief states to be propagated as described in Section 3.2.1. The largest size of a belief state in G is $|\mathcal{S}|$, and the belief prediction step in (3.7) is $O(|\mathcal{S}|^2)$. This computational cost, however, is shared among all children of the same hyperedge, i.e., each child can be seen as incurring a belief prediction cost of $O\left(\frac{|\mathcal{S}|^2}{|\mathcal{O}|}\right)$. Next, the belief state update in (3.10) has a cost of $O(|\mathcal{S}|)$ for each child, yielding a total belief state cost of

$$O\left(\frac{|\mathcal{S}|^2}{|\mathcal{O}|} + |\mathcal{S}|\right) \tag{3.36}$$

per child node. By multiplying (3.35) and (3.36), we get

$$O(|\mathcal{S}|^2|\mathcal{A}|^h|\mathcal{O}|^{h-1} + |\mathcal{S}||\mathcal{A}|^h|\mathcal{O}|^h) \tag{3.37}$$

as the worst case computational complexity for RAO*, as approximated by the com-

putational cost of expanding G .

3.5 Experiments

This section provides empirical evidence of the usefulness and general applicability of CC-POMDP’s as a modeling tool for risk-sensitive applications, and shows how RAO* performs when computing risk-bounded policies in two challenging domains of practical interest: automated planning for science agents (SA) [Benazera et al., 2005b]; and power supply restoration (PSR) [Thiébaux and Cordier, 2001]. All models and RAO* were implemented in Python and ran on an Intel Core i7-2630QM CPU with 8GB of RAM.

Our SA domain is based on the planetary rover scenario described in [Benazera et al., 2005b]. Starting from some initial position in a map with obstacles, the science agent may visit four different sites on the map, each of which could contain new discoveries with probability based on a prior belief. If the agent visits a location that contains new discoveries, we assume that it will find it with high probability. The agent’s position is uncertain, and position uncertainty is unbounded, hence there is always a non-zero risk of collision when the agent is traveling between locations. The agent is required to finish its mission at a relay station, where it can communicate with an orbiting satellite and transmit its findings. Since the relay satellite periodically moves below the horizon, there is a limited time window for the agent to gather as much information as possible and arrive at the relay station in order to communicate with the satellite. Moreover, we assume the duration of each traversal to be uncontrollable, but bounded. In this domain, we use a single chance constraint to ensure that the event “arrives at the relay location on time” happens with probability at least $1 - \Delta$. The SA domain has size $|\mathcal{S}| = 6144$; $|\mathcal{A}| = 34$, $|\mathcal{O}| = 10$.

In the *PSR* domain [Thiébaux and Cordier, 2001], the objective is to reconfigure a faulty power network by switching lines on or off so as to resupply as many customers as possible. One of the safety constraints is to keep faults isolated at all times, to avoid endangering people and enlarging the set of areas left without power. However, fault

locations are hidden, and more information cannot be obtained without taking the risk of resupplying a fault. Therefore, the chance constraint is used to limit the probability of connecting power generators to faulty buses. Our experiments focused on the semi-rural network from [Thiébaux and Cordier, 2001], which was significantly beyond the reach of [Bonet and Thiébaux, 2003] even for single faults. In our experiments, there were always circuit breakers at each generator, plus different numbers of additional circuit breakers depending on the experiment. Observations correspond to circuit breakers being open or closed, and actions to opening and closing switches. The PSR domain is strongly combinatorial, with $|\mathcal{S}| = 2^{61}$; $|\mathcal{A}| = 68$, $|\mathcal{O}| = 32$.

We evaluated the performance of RAO* in both domains under various conditions, and the results are summarized in Tables 3.1 (higher utility is better) and 3.2 (lower cost is better). The runtime for RAO* is always displayed in the *Time* column; *Nodes* is the number of hypergraph nodes expanded during search, each one of them containing a belief state with one or more particles; and *States* is the number of evaluated belief state particles. It is worthwhile to mention that constraint violations in PSR *do not cause execution to terminate*, and the same is true for scheduling violations in SA. The only type of terminal constraint violation are collisions in SA, and RAO* makes proper use of this extra bit of information to update its beliefs. Therefore, PSR and SA are examples of risk-sensitive domains which can be appropriately modeled as CC-POMDP’s, but not as C-POMDP’s with unit costs. The heuristics used were straightforward: for the execution risk, we used the admissible heuristic $h_{er}(b_k|\pi) = r_b(b_k)$ in both domains. For Q values, the heuristic for each state in PSR consisted in the final penalty incurred if only its faulty nodes were not resupplied, while in SA it was the sum of the utilities of all non-visited discoveries.

As expected, both tables show that increasing the maximum amount of risk Δ allowed during execution can only improve the policy’s objective. The improvement is not monotonic, though. The impact of the chance constraint on the objective is discontinuous on Δ when only deterministic policies are considered, since one cannot randomly select between two actions in order to achieve a continuous interpolation between risk levels. Being able to compute increasingly better approximations of a

policy’s execution risk, combined with forward propagation of risk bounds, also allow RAO* to converge faster by quickly pruning candidate policies that are guaranteed to violate the chance constraint. This can be clearly observed in Table 3.2 when we move from $\Delta = 0.5$ to $\Delta = 1.0$ (no chance constraint).

Another important aspect is the impact of sensor information on the performance of RAO*. Adding more sources of sensing information increases the branching on the search hypergraph used by RAO*, so one could expect performance to degrade. However, that is not necessarily the case, as shown by the left and right numbers in the cells of Table 3.2. By adding more sensors to the power network, RAO* can more quickly reduce the size of its belief states, therefore leading to a reduced number of states evaluated during search. Another benefit of reduced belief states is that RAO* can more effectively reroute energy in the network within the given risk bound, leading to lower execution costs.

Finally, we wanted to investigate how well a C-POMDP approach would perform in these domains relative to a CC-POMDP. Following the literature, we made the additional assumption that execution halts at all constraint violations, and assigned unit terminal costs to those search nodes. Results on two example instances of PSR and SA domains were the following: I) in SA, C-POMDP and CC-POMDP both attained an utility of 29.454; II) in PSR, C-POMDP reached a final cost of 53.330, while CC-POMDP attained 36.509. The chance constraints were always identical for C-POMDP and CC-POMDP. First, one should notice that both models had the same performance in the SA domain, which is in agreement with the claim that they coincide in the particular case where all constraint violations are terminal. The same, however, clearly does not hold in the PSR domain, where the C-POMDP model had significantly worse performance than its corresponding CC-POMDP with the exact same parameters. In cost-based C-POMDP approaches, assuming that constraint violations are terminal greatly restricts the space of potential solution policies in domains with non-destructive constraint violations, leading to conservatism. A CC-POMDP formulation, on the other hand, can potentially attain significantly better performance while offering the same safety guarantee.

Window[s]	Δ	Time[s]	Nodes	States	Utility
20	0.05	1.30	1	32	0.000
30	0.01	1.32	1	32	0.000
30	0.05	49.35	83	578	29.168
40	0.002	9.92	15	164	21.958
40	0.01	44.86	75	551	29.433
40	0.05	38.79	65	443	29.433
100	0.002	95.23	127	1220	24.970
100	0.01	184.80	161	1247	29.454
100	0.05	174.90	151	1151	29.454

Table 3.1: SA results for various time windows and risk levels. The *Window* column refers to the time window for the SA agent to gather information, not a runtime limit for RAO*.

Δ	Time[s]	Nodes	States	Cost
0	0.025/0.013	1.57/1.29	5.86/2.71	45.0/30.0
.5	0.059/0.014	3.43/1.29	10.71/2.71	44.18/30.0
1	2.256/0.165	69.3/11.14	260.4/23.43	30.54/22.89
0	0.078/0.043	2.0/1.67	18.0/8.3	84.0/63.0
.5	0.157/0.014	3.0/1.29	27.0/2.71	84.0/30.0
1	32.78/0.28	248.7/5.67	1340/32.33	77.12/57.03
0	1.122/0.093	7.0/2.0	189.0/12.0	126.0/94.50
.5	0.613/0.26	4.5/4.5	121.5/34.5	126.0/94.50
1	123.9/51.36	481.5/480	8590.5/2648	117.6/80.89

Table 3.2: PSR results for various numbers of faults (#) and risk levels. Top: avg. of 7 single faults. Middle: avg. of 3 double faults. Bottom: avg. of 2 triple faults. Left (right) numbers correspond to 12 (16) network sensors.

3.6 Conclusions

This chapter formally introduces CC-POMDP’s and their associated dynamic measure of execution risk. It also presents RAO*, an HFS-based algorithm for solving CC-POMDP’s that not only incorporates execution risk in its computations, but also propagates it forward in order to perform early pruning of overly risky policy branches. The experiments in this chapter provide experimental evidence of two important facts: first, that CC-POMDP’s are effective in addressing shortcomings in previous cost-based handlings of chance constraints; and second, that RAO* is able to solve challenging risk-sensitive planning problems of practical interest and size by combining insights of AO* with forward propagation of risk upper bounds.

With CC-POMDP's and RAO* as two fundamental building blocks of CLARK, the next chapter introduces cRMPL, a programming language that can serve as a higher level, more abstract option for the specification of risk-aware planning problems by mission operators. After motivating the usefulness of the language and presenting its syntax, we show how CC-POMDP's and RAO* play a key role in the problem of extracting optimal, chance-constrained execution policies from a cRMPL program specification.

Chapter 4

Programming risk-aware missions with cRMPL

“A programming language is low level when its programs require attention to the irrelevant.”

Alan Perlis.

The previous chapter introduces CC-POMDP’s as a means to describe risk-sensitive domains of planning under uncertainty. Simply put, a CC-POMDP establishes which actions are available for an autonomous agent to choose from; the “value” associated with their execution; and the observations that the agent can collect from its sensors as it executes its policy dynamically. Concomitantly, safe behavior corresponds to the agent’s state remaining within feasibility regions defined by state constraints, which may as well include temporal constraints restricting the scheduling of plan actions.

Related to this thesis’ goal of generating risk-bounded plans for agents that make decisions based on sensor observations, it is important that mission operators be given a tool for specifying such risk-aware mission at a high level of abstraction, as if by means of a programming language. For that purpose, this chapter introduces the Chance-constrained Reactive Model-based Programming Language (cRMPL), and its relationship with CC-POMDP models from Chapter 3. Our cRMPL extends the RMPL variant from [Effinger, 2012], itself a variant of previous versions of RMPL

[Ingham et al., 2001, Williams et al., 2001, Williams and Ingham, 2002, Williams et al., 2003, Ingham, 2003], by adding support to probabilistic sensing actions and the ability to impose chance constraints over any cRMPL (sub)expression. By doing so, cRMPL allows the high-level specification of control programs for autonomous agents that must operate under uncertainty and bounded risk.

4.1 Introduction

As all previous variants of RMPL [Ingham et al., 2001, Williams et al., 2001, Williams and Ingham, 2002, Williams et al., 2003, Ingham, 2003, Effinger, 2012], a key feature of cRMPL is that it is particularly useful when mission operators desire to exert tight control over the decisions available to an autonomous agent and how it reacts to its sensors, while offloading the burden of lower level plan dispatching to a program executive. Similar to DTGolog [Boutilier et al., 2000], a cRMPL program could be seen as “advice” from an expert mission operator that *constraints* the space of decisions and sensor observations that should be taken into account by a decision-theoretic executive for that program. In doing so, we implicitly assume that the mission operator is capable of providing an accurate approximation of the solution to a complex conditional planning problem in the form of a cRMPL program, but is unwilling to commit to specific decisions in the plan due to the difficulty of reasoning about the underlying constraints.

As in the original RMPL, mission specifications in cRMPL offer flexibility in the choice of action sequences used to reach goals, which are exploited during execution to achieve robustness. In cRMPL, like traditional reactive programs, time-evolved behavior can be specified using standard concurrent programming constructs, including parallel and sequential execution; conditional execution; iteration; contingencies (if-like statements); and timed execution. Programs in cRMPL can also specify temporal bounds in time-critical missions.

Unlike traditional languages and previous versions of RMPL, cRMPL can bound the risk of execution failure through the incorporation of chance constraints. In timed

cRMPL programs, chance constraints can, among other uses, specify upper bounds on the probability of violating temporal constraints within an *episode*, which are explained in Section 4.4.1. Similar to previous versions of RMPL, cRMPL improves robustness by including *choice* constructs that can either represent *decisions* (also referred to as *controllable choices*) between functionally-equivalent procedures, which the program executive can assign at runtime in order to adapt to the execution at hand; or potentially noisy *sensor observations* (also referred to as *uncontrollable choices*) that the agent collects during execution, such as the presence of an object in the scene or the last action failing to accomplish its intended goals.

An important concept presented in this chapter is the framing of cRMPL execution as a CC-POMDP, and the use of RAO* for incremental, heuristically-guided generation of optimal execution policies for cRMPL programs. This strategy is in contrast with previous approaches to decision-theoretic RMPL execution [Kim et al., 2001, Conrad and Williams, 2011, Effinger, 2012, Levine and Williams, 2014], which require an exponentially large unraveling of all possible execution traces for the program before starting the search for an optimal execution policy. Also related to the framing of cRMPL execution as a CC-POMDP, it allows us to depart from the time discretization approach in [Effinger, 2012] and, similar to state-of-the-art temporal planners [Coles et al., 2009, Coles et al., 2012, Cimatti et al., 2015, Wang, 2015], determine the existence of a feasible schedule by reasoning over temporal constraint networks featuring continuous time.

This chapter starts by motivating the need for risk-bounded model-based programming in Section 4.2, where we use cRMPL to specify a simple risk-bounded Mars survey that must be completed under time pressure by an autonomous rover. Next, Section 4.3 presents a short review of the most important design desiderata of cRMPL, followed by a description of *episodes* in Section 4.4.1, the fundamental building blocks in our implementation. Next, Section 4.4.2 presents the different types of constraints that can be imposed on cRMPL episodes, and Section 4.4.3 shows how these episodes can be composed together in order to form larger, hierarchical programs. The execution semantics of cRMPL in terms of a mapping to CC-POMDP's is given

in Section 4.5, followed by concluding remarks in Section 4.6.

4.2 Motivation: programming high level missions

Figure 4-1 depicts a planetary exploration scenario on the MobileSim [Adept MobileRobots, 2005] simulator, with names of interesting sites overlaid on the map. Spirit, an intrepid rover, must navigate around terrain obstacles in order to visit several of these sites, while also making sure to arrive at the “relay” location in time to communicate with an orbiting satellite.

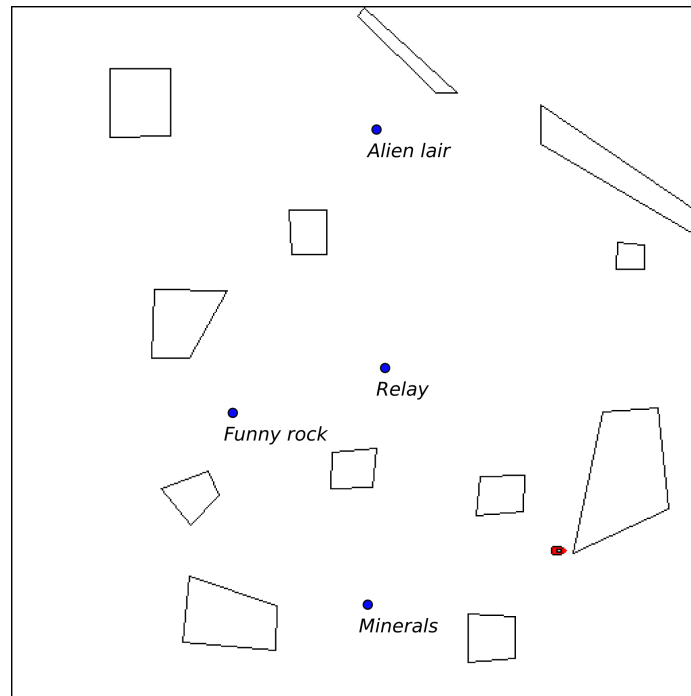


Figure 4-1: Mars rover scenario where a robotic scout must explore different regions of the map before driving back to a relay location and communicating with an orbiting satellite.

After conferencing with the team of scientists with interest in exploring this region, the mission operator wants to communicate the following to the rover:

“From your current location, initiate the mission by traveling to the ‘minerals’ location, followed by a traversal to ‘funny rock’. At that point, decide whether to visit a potential alien lair, where choosing to visit ‘alien lair’

is preferred over not visiting it. If you decide to visit ‘alien lair’, go to ‘relay’ next. Otherwise, go directly to ‘relay’. Each traversal should have a risk of collision no greater than 1%. The mission should be completed in no more than 1000 seconds.”

The fact that this mission description is written as a short English paragraph suggests that it communicates goals, constraints, and preferences at a level of abstraction natural for human interaction, except for maybe the portion about bounded risk of collision. However, even the latter could be intuitively understood by someone familiar with vehicle dynamics: “do not travel too fast and in close proximity to obstacles”. Nevertheless, while convenient for human specification and understanding, there are a number of fundamental mission aspects that have not been specified by the paragraph: how should the rover navigate its environment in order to limit the risk of collision? How should it dispatch activities, particularly when traversals exhibit random variations in their durations? How can it decide whether to visit “alien lair” or not? Are 1000 seconds sufficient time? All of these are crucial details for the successful execution of the mission, but they require careful, and often laborious, reasoning over models, goals, and mission constraints. Instead of forcing mission operators to go through the arduous and error-prone process of eliciting all of these details, cRMPL, inheriting the model-based programming paradigm of RMPL, allows this burden to be transferred to the program executive (the CLARK executive in Figure 1-10), while also adding support to risk-bounded execution.

The cRMPL control program corresponding to the English description of the rover mission is shown in Figure 4-2. Since this thesis’ implementation of cRMPL is done as an extension of Python¹, the cRMPL program is an instance of the RMPyL class. The meaning of the `sequence` and `decide` constructs should be evident from their names: `sequence` imposes the constraint that its arguments must be executed *one after the other*, while `decide` allows the program executive to *nondeterministically choose* different program execution threads, while potentially assigning different re-

¹More specifically, CPython, the standard implementation of the Python interpreter done in the C programming language.

```

loc={'start':(8.751,-8.625),
     'minerals':(0.0,-10.0),
     'funny_rock':(-5.0,-2.0),
     'relay':(0.0,0.0),
     'alien_lair':(0.0,10.0)}

rov1 = Rover(name='spirit')
prog = RMPyL(name='run()')#name=run() is a requirement for Enterprise at the moment

prog *= prog.sequence(
    rov1.go_to(start=loc['start'],goal=loc['minerals'],risk=0.01),
    rov1.go_to(start=loc['minerals'],goal=loc['funny_rock'],risk=0.01),
    prog.decide({'name':rov1.name+'-visit-aliens',
                'domain':['Visit','Do-not-visit'],
                'utility':[10,0]},
               prog.sequence(
                   rov1.go_to(start=loc['funny_rock'],goal=loc['alien_lair'],risk=0.01),
                   rov1.go_to(start=loc['alien_lair'],goal=loc['relay'],risk=0.01)),
               rov1.go_to(start=loc['funny_rock'],goal=loc['relay'],risk=0.01)))

prog.add_overall_temporal_constraint(ctype='controllable',lb=0.0,ub=1000.0)

```

Figure 4-2: Simple rover control program expressed in cRMPL.

wards to each one of them. The command at the bottom of Figure 4-2 is syntactic sugar to impose a temporal constraint restricting the temporal execution of the program to 1000 seconds. Those and many other cRMPL constructs are explained in detail throughout this chapter.

The `go_to` operator in Figure 4-2 represents risk-bounded traversals with probabilistic durations, an example of cRMPL’s ability to specify chance-constrained program execution. Its encoding within cRMPL leverages pSulu [Ono and Williams, 2008, Blackmore et al., 2011, Ono et al., 2012b, Ono et al., 2012a, Ono et al., 2013], a chance-constrained path planner, and is explained in detail in Section 7.1.4. However, for the sake of providing an intuitive understanding of the encoding, consider the fictitious traversal shown in Figure 4-3. When pSulu plans a trajectory between two distant locations on the map, it determines a set of intermediate waypoints along the way, shown as little black circles in Figure 4-3. Since these intermediate waypoints are close to each other and connected by collision-free segments, the rover is capable of dispatching waypoint-following tasks that seek to move it from one waypoint to another on a straight line, while avoiding small rocks and other minor terrain features. Therefore, waypoint-following are directly executable activities (also referred

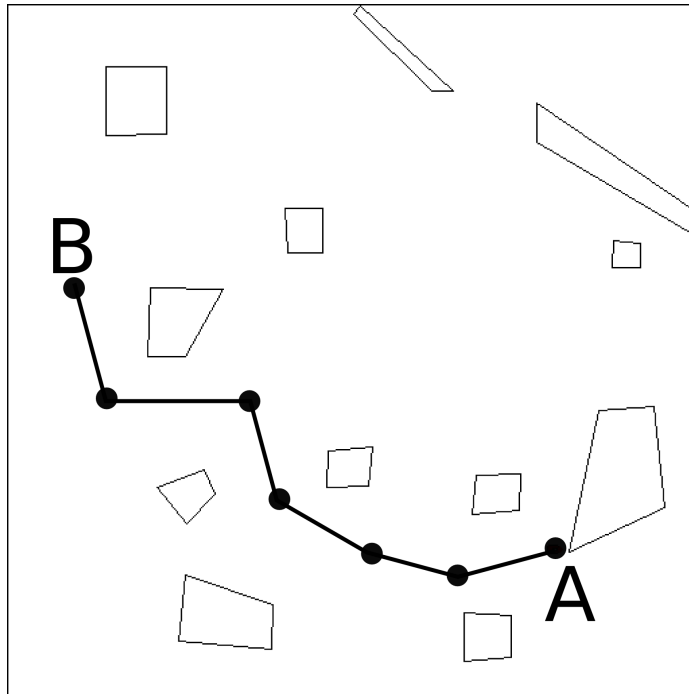


Figure 4-3: Example route between two arbitrary locations A and B on a map. The intermediate dots connecting path segments are intermediate waypoints that the robot should visit in its traversal in order to maintain a safe distance from obstacles.

to as *primitive*) for the rover, whose temporal durations are a function of the distance between waypoints and the rover’s physical limitations. Since a traversal from A to B consists on the sequential execution of these waypoint-following primitives, the whole traversal is represented as a **sequence** of waypoint-following activities.

Figure 4-4 shows the result of the CLARK executive dispatching the cRMPL program in Figure 4-2, which is connected to MobileSim through the *Enterprise* architecture [Burke et al., 2014, Timmons et al., 2015], as explained in Section 7.1.3. Since there is enough time and a higher reward is associated with the execution thread that visits “alien lair”, Spirit chooses to visit that site.

4.3 Design desiderata for cRMPL

According to [Williams and Ingham, 2002], a model-based program is comprised of two components. The first is a control program, which uses standard programming constructs to codify specifications of desired system behavior. In addition, to execute

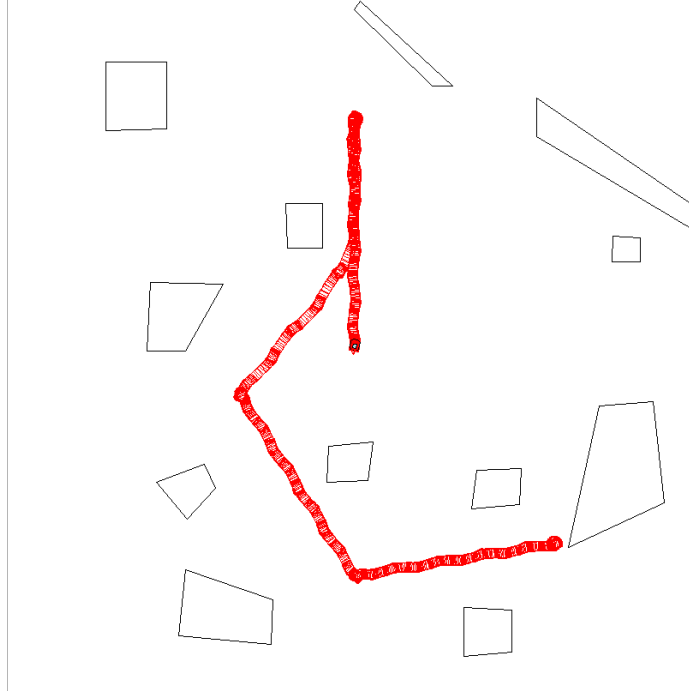


Figure 4-4: Result of the CLARK executive dispatching the cRMPL program in Figure 4-2.

the control program, the execution kernel (e.g., the CLARK executive in Figure 1-10) needs a model of the underlying system it must control. Hence, the second component is a Plant model that includes nominal behavior and common failure modes. In this thesis, the first component corresponds to cRMPL control programs, whose syntax and execution semantics are explained in this chapter. The second component, the Plant model, is implicitly defined by the constraint-checking operators in a CC-POMDP (Chapter 3).

The first two important features of cRMPL are its ability to generate *risk-bounded* control policies, improving upon the conservatism of previous risk-minimizing approaches [Effinger, 2012]; and the incorporation of *probabilistic sensing actions* into the program execution semantics, therefore allowing for a principled handling of belief states.

The third important feature of the cRMPL version developed in this thesis is related to its ease of use within current robotics frameworks, e.g., the Robot Operating System (ROS) [Quigley et al., 2009], a key capability for modern agent programming

languages [Ziafati et al., 2012]. Recent efforts to integrate the original standalone version of RMPL into robotic applications led the core features of the language to be made available through extensions of existing general-purpose programming languages, such as Common Lisp and Python. In particular, this thesis developed *RMPyL*, as we refer to our Python-based implementation of the cRMPL constructs. Our choice for Python was not fortuitous: first, as pointed out in [Millman and Aivazis, 2011], Python and its scientific tools [Jones et al., 01] have arguably become a *de facto* standard for computation-driven scientific research, with popular libraries for matrix manipulation [van der Walt et al., 2011], mathematical and constraint programming [Hart et al., 2012, Hebrard et al., 2010], machine learning [Pedregosa et al., 2011], among many others; second, it is one of the standard programming languages used in ROS; third, the popularity of Python and its simple² integration with high-performance C/C++ code makes it particularly well-suited for interacting with external constraint solvers used to assess risk in CC-POMDP models.

4.4 Syntax

This section explains in detail the different elements of the cRMPL syntax shown in the Extended Backus-Naur Form (EBNF) grammar of Figure 4-5.

4.4.1 Episodes

The core building block of cRMPL is the *episode* (see Definition 4.2), a pictorial example of which is shown in Figure 4-6. Intuitively, an episode marks a period of time during which an *activity* must be performed while observing zero or more constraints over relevant system variables, generally referred to as *state*. Special attention is given to two particular types of constraints: *temporal constraints* place limitations on the relationships between *temporal events*, while *chance constraints* place limits on the probability of violating constraints within an episode. Episode

²For virtually seamless integration, see [Behnel et al., 2011].

```

    <Episode> ::= <Primitive episode> | <Composite episode>
    <Primitive episode> ::= Episode(action=<Action>,duration=<Duration dict.>)
    <Composite Episode> ::= <Sequence episode> | <Parallel episode> | <Choice episode>
    <Sequence episode> ::= sequence(<Episode>{,<Episode>})
    <Parallel episode> ::= parallel(<Episode>{,<Episode>})
    <Choice episode> ::= choose(<Choice>,<Episode>{,<Episode>})
    <Choice> ::= Choice(domain=<Discrete dom.>,ctype=<Choice Type>)
    <Choice Type> ::= "controllable" | "set-bounded" | "probabilistic"
    <Action> ::= String describing the action to be executed
    <Duration dict.> ::= Dictionary of duration properties.
    <Constraint> ::= <Temporal const.> | <Chance const.> | <State const.>
    <Temporal const.> ::= TemporalConstraint(start=<Event>, end=<Event>,
    ctype=<Temp. type>)
    <Temp. type> ::= "controllable" | "uncontrollable_bounded" | "uncon-
    trollable_probabilistic"
    <Event> ::= Event()
    <Chance const.> ::= ChanceConstraint(scope={<Constraint>},
    risk=<Probability>)
    <Probability> ::= Probability value between 0 and 1.
    <State const.> ::= <Assignment const.> | <Linear const.>
    <Assignment const.> ::= AssignmentStateConstraint(scope={<State
    var.>}, values=<Domain>)
    <Linear const.> ::= LinearStateConstraint(scope={<State var.>},
    expr=<Linear expression>)
    <State var.> ::= StateVariable(domain=<Domain>)
    <Domain> ::= <Discrete dom.> | <Continuous dom.>
    <Discrete dom.> ::= Discrete list of symbols.
    <Continuous dom.> ::= Continuous range of values.
    <Linear expression> ::= Coefficients and relationship of a linear expression.

```

Figure 4-5: Extended Backus-Naur Form (EBNF) grammar for cRMPL.

constraints that are not temporal or chance constraints are generally referred to as *state constraints*.

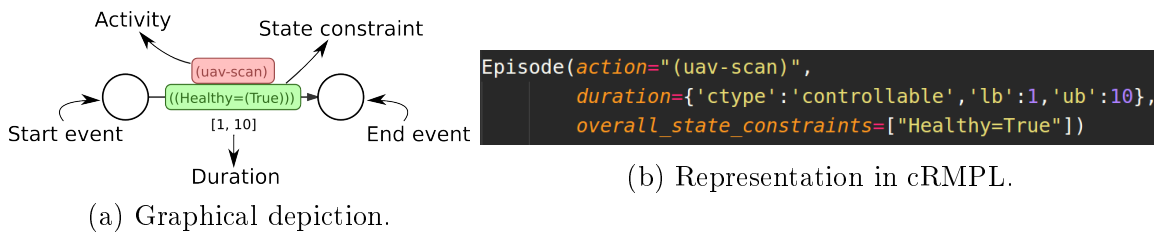


Figure 4-6: Episode specifying that an unmanned aerial vehicle (UAV) should scan an area for a period between 1 and 10 time units, while making sure that it maintains itself in a healthy state through the state constraint `Healthy=True`. If `uav-scan` can be directly executed by the UAV, this would be a primitive episode. Otherwise, if `uav-scan` requires a combination of more fundamental episodes, then this episode would be composite.

Akin to mainstream programming languages and following the original principles in RMPL, an important concept related to episodes in cRMPL is their *hierarchical composition*, the topic of Section 4.4.3. Hierarchy is a fundamental tool to be able to describe complex behavior in terms of well-defined combinations of increasingly simpler functionality, up to the level that can be directly understood by the underlying hardware executing the activities. For instance, consider the process through which a `print`-like statement in the programming language of your choice is repeatedly refined into combinations of more basic, lower level commands that eventually cause the pixels on the screen to display a message such as “Hello, world!”. If the activity specified for an episode can be directly executed by the system under control, we call the episode *primitive*. Alternatively, if the activity to be performed within an episode consists of a combination of other episodes, we call the episode *composite*.

Definition 4.2 (Episode). *An episode is a tuple $E = \langle en, t_s, t_e, \mathcal{C}, A \rangle$, where*

- *en : Boolean variable that is true if, and only if, the episode is enabled for execution;*
- *t_s, t_e : respectively, temporal events marking the start and end of E , such that $t_s, t_e \in \mathbb{R}^+, t_s \leq t_e$;*

- \mathcal{C} : set of constraints that should hold during the period $[t_s, t_e]$ provided that the episode is enabled. It is partitioned into temporal constraints \mathcal{C}_t , chance constraints \mathcal{C}_c , and state constraints \mathcal{C}_s ;
- A : activity to be performed during the period $[t_s, t_e]$ while observing \mathcal{C} provided that the episode is enabled. For primitive episodes, A is a function that can be directly executed by the underlying system. For composite episodes, A is a function that governs the enabling of E 's components.

4.4.2 Episode constraints

True to the spirit of previous RMPL variants and the agent programming languages that inspired them, cRMPL seeks to control autonomous agents by a temporal succession of constraints that specify how the (hidden) state of the Plant should evolve across time, rather than describing the steps to achieve such evolution as in general-purpose programming languages. Moreover, by allowing the specification of chance constraints that bound the probability of constraint violations within episodes, cRMPL departs from the conservatism of risk-minimal execution strategies by accepting the possibility that state and environmental uncertainty, including model imperfections, may cause execution to deviate from its “nominal” path. In this section, we present the different types of constraints that cRMPL can currently impose on episodes, such as the one shown in Figure 4-6.

Temporal constraints

Three basic types of simple temporal constraints are supported in cRMPL: Simple Temporal Constraints (STC's) [Dechter et al., 1991]; STCs with Uncertainty (STCU's) [Vidal and Ghallab, 1996, Vidal, 1999]; and Probabilistic STCs (PSTC's) [Tsamardinos, 2002, Fang et al., 2014, Wang and Williams, 2015a, Yu et al., 2015]. A temporal network allowing any combination of such simple temporal constraints is called a Probabilistic Simple Temporal Network with Uncertainty (PSTNU) [Santana et al., 2016c]. Disjunctive and conditional temporal networks can be represented as

combinations of these different types of simple temporal constraints with the choice operators from Section 4.4.3. Example of such networks are Conditional Temporal Plans (CTPs) [Tsamardinos et al., 2003]; Temporal Plan Networks with Uncertainty (TPNU’s) [Effinger et al., 2009, Effinger, 2012]; Disjunctive Temporal Problems with Uncertainty (DTPU) [Venable et al., 2010]; Conditional Simple Temporal Network with Uncertainty (CSTNU) [Hunsberger et al., 2012]; and the Probabilistic Temporal Plan Networks (PTPN’s) [Levine and Williams, 2014, Santana and Williams, 2014].

An STC over two temporal events e_1 and e_2 is a tuple $\langle e_1, e_2, l, u \rangle$ representing the constraint $l \leq e_2 - e_1 \leq u$, $l \leq u$, $l, u \in \mathbb{R} \cup \{-\infty, \infty\}$. An STC simply places metric limits on the temporal distance between two arbitrary temporal events, regardless of how their temporal assignments, also called a *schedule*, is chosen.

An STCU resembles an STC and is given by a tuple $\langle e_1, e_2, l, u \rangle$, $l \leq u$, $l, u \in \mathbb{R}_{>0}$. However, instead of simply imposing limits on the temporal distance between two arbitrary events, and STCU expresses the relationship $e_2 = e_1 + d$, $l \leq d \leq u$, where d is a set-bounded, non-deterministic, non-negative duration whose value is chosen by an external agent, usually referred to as the “environment” or “Nature”. Therefore, different from an STC, an STCU imposes the additional constraint that e_2 must be a *contingent* (or *uncontrollable*) event: its schedule will be chosen by the environment during execution so that $e_2 - e_1 \in [l, u]$, but its specific value cannot be determined beforehand.

Similar to an STCU, a PSTC is a tuple $\langle e_1, e_2, d \rangle$ defining the relationship $e_2 = e_1 + d$, where d is also assumed to be a non-deterministic duration whose value is chosen by the environment. Nevertheless, different from an STCU, a PSTC incorporates information about the relative frequency of different values of d in the form of a known *probability distribution* with *positive support*.

A detailed presentation of all these types of temporal constraints, along with an efficient algorithm for risk-aware scheduling, are given in Chapter 5.

State constraints

Let X be a vector of state variables, which can be either discrete state variables over finite domains; or numerical state variables over continuous ranges of values. For those types of state variables, two types of constraints are available:

- general linear constraints of the form $AX_n \begin{smallmatrix} \leq \\ \geq \end{smallmatrix} b$, where A, b are constant matrices of appropriate dimensions and X_n is the subset of X composed of numerical state variables;
- and assignment constraints $X = c$, where c is a vector of constants.

Inspired by PDDL2.1 [Fox and Long, 2003], it is possible to specify more precisely when such state constraints must hold within an episode:

- **at start**: state constraints that must hold by the time the start event of an episode is executed;
- **at end**: same as *at start*, but for the end event of an episode;
- **during**: state constraints that must hold during the whole period between the start and end events, but not necessarily at the extrema;
- **overall**: state constraints that must hold at all previously-mentioned periods.

Chance constraints

The ability to impose chance constraints on episodes is one of the most important features of cRMPL. In essence, a chance constraint provides a bound Δ on the probability of a set of constraints $\mathcal{C}_c \subseteq \mathcal{C}$ from an episode E being violated during execution. Therefore, a chance constraint is a tuple $\langle \mathcal{C}_c, \Delta \rangle$.

4.4.3 Composing episodes in cRMPL

Following previous work on RMPL [Kim et al., 2001, Effinger, 2012], cRMPL subroutines can be hierarchically combined using three fundamental operators: **sequence**,

parallel, and choose. The choose operator can be further subdivided into *controllable choices*, which can be assigned by the agent; and *uncontrollable choices*, which are assigned by Nature and can only be observed by the agent at runtime. These fundamental operators, in turn, can be combined to generate more complex behavior, such as iteration and exception-handling.

The outcome of applying the **sequence**, **parallel**, or **choose** operators to a list L_E of two or more episodes will be a composite episode E_{comp} with components $\text{Components}(E_{comp}) = L_E$. As in Figure 4-6, the activity associated with an episode is graphically represented as a red box.

Sequence composition

In a **sequence** composition of episodes E_{seq} , component episodes are executed one after the other, as enforced by the $[0, \infty]$ STC's shown in Figure 4-7 and belonging to $\mathcal{C}_t(E_{seq})$. In cRMPL, a list of episodes can be composed in sequence through the operator **sequence**.

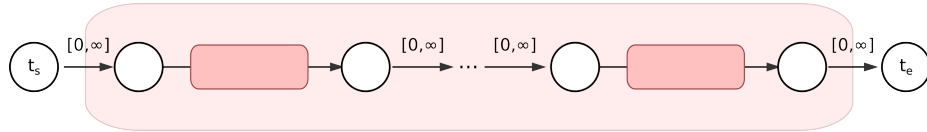


Figure 4-7: Composite episode generated by the **sequence** operator, which enforces sequential temporal execution by means of $[0, \infty]$ STC's.

The activity function A of a **sequence** episode is such that

$$en(E_{seq}) \Leftrightarrow en(E_i), \forall E_i \in \text{Components}(E_{seq}). \quad (4.1)$$

In other words, enabling a sequential composite episode E_{seq} is equivalent to enabling all of its components episodes. Notice that the STC's in Figure 4-7, which must be satisfied whenever $en(E_{seq})$ holds, make sure that the components of E_{seq} are dispatched in the correct order.

Parallel composition

In the `parallel` composition of episodes E_{par} from Figure 4-8, component episodes can be scheduled to be executed at the same time. In cRMPL, a list of episodes can be composed in parallel through the operator `parallel`.

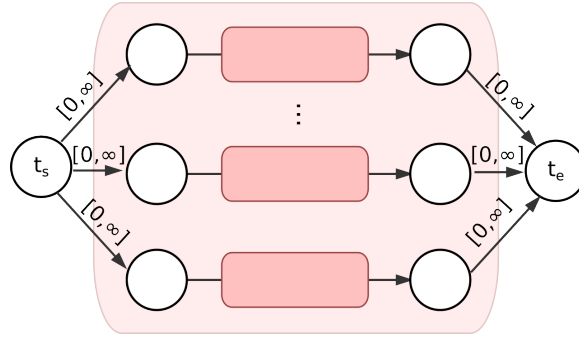


Figure 4-8: Composite episode generated by the `parallel` operator. Different from `sequence`, component episodes in a parallel composition can be scheduled to happen at the same time.

The activity function A of a `parallel` episode is given by (4.1) with E_{seq} replaced by E_{par} .

Choice composition

A `choose` composition is similar to `parallel` in terms of structure, but *only one of the branches is ever executed*. Therefore, constraints from only one of the options must be enforced by the program. Moreover, the start event t_s of a `choose` episode will always be a `<Choice>` element, as defined by the grammar in Figure 4-5.

Choices are pictorially represented as double circles, as shown in Figure 4-9. Choices are either controllable (assigned by the control program) or uncontrollable (assigned by Nature). Uncontrollable choices can be either *set-bounded*, when there is no probability distribution associated with the different outcomes; or *probabilistic*, when such a probability distribution is available. Since controllable choices are often used to represent decisions by the program executive, while uncontrollable choices usually represent sensor readings and other types of environmental observations, cRMPL

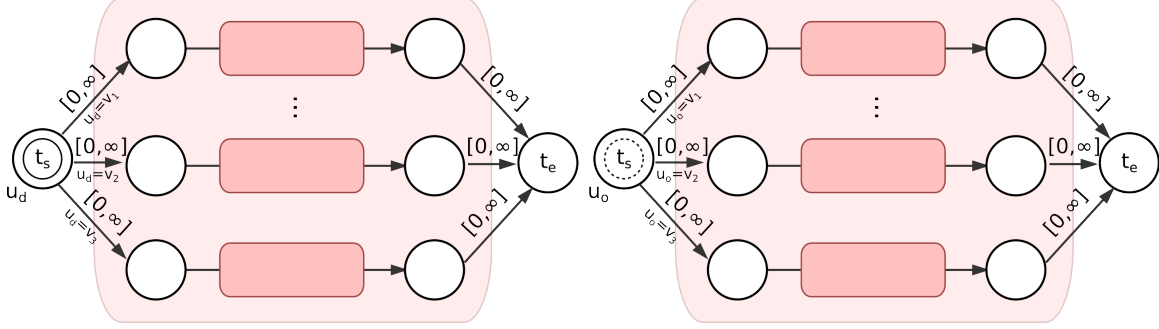


Figure 4-9: Composite episodes generated by two instances of the **choose** operator. The composite episode on the left corresponds to a controllable choice (decision) u_d , while the one on the right corresponds to an uncontrollable choice (observation) u_o .

also incorporates syntactic-sugar constructs **decide** and **observe** to represent, respectively, controllable and uncontrollable choices.

Different from **sequence** and **parallel** compositions, where enabling the outer composite episode entails the enabling of all of its components, only a single component of a **choose** composition E_{cho} will ever be enabled. This “branch selection” is represented as an assignment to a discrete $\langle \text{Choice} \rangle$ element associated with E_{cho} ’s start event t_s . Let u denote this choice element and let $\{v_1, v_2, \dots, v_n\}$ be its discrete domain. Also, let E_i be the component episode associated with the assignment $u = v_i$. The activity function A of a **choose** episode is given by

$$en(E_{cho}) \wedge u = v_i \Leftrightarrow en(E_i), \forall E_i \in \mathbf{Components}(E_{cho}). \quad (4.2)$$

Iteration composition

Different flavors of iteration are implemented in cRMPL on top of the **sequence** and **choose** operators. These are

$$\begin{aligned} \mathbf{while}(Test, E_l) = & \mathbf{observe}(Test, \\ & \mathbf{sequence}(E_l, \mathbf{while}(Test, E_l)), \\ & \mathbf{Episode}(\mathbf{action}=\mathbf{None})), \end{aligned} \quad (4.3)$$

$$\begin{aligned}
\text{for}(n, E_l) = & \text{observe}(n > 0, \\
& \text{sequence}(E_l, \text{for}(n - 1, E_l)), \\
& \text{Episode}(\text{action}=\text{None})), \tag{4.4}
\end{aligned}$$

$$\begin{aligned}
\text{loop}(D, E_l) = & \text{decide}(D, \\
& \text{sequence}(E_l, \text{loop}(D, E_l)), \\
& \text{Episode}(\text{action}=\text{None})). \tag{4.5}
\end{aligned}$$

In (4.3)-(4.5), E_l is an episode executed on each loop of the iteration and `Episode(action = None)` is a primitive episode that executes no operation (no-op). A `while` iteration takes a Boolean uncontrollable choice $Test$ and will keep executing the iteration until $Test$ evaluates to False, as it is usual in most programming languages. Similarly, a `for` iteration is based on a Boolean uncontrollable choice that executes E_l exactly n times. Finally, and different from the previous two, a `loop` iteration allows the control program to pick whether to execute another iteration based on a controllable choice D , which could, for example, assign a reward for every time the loop is executed.

4.5 Execution semantics

This section defines the execution semantics of cRMPL programs in terms of CC-POMDP models, and shows how RAO* can be used to generate optimal, chance-constrained execution policies for cRMPL programs that must terminate in finite time. The key idea behind our approach is to construct a CC-POMDP model that simulates valid executions of the cRMPL program, and use RAO* to incrementally unravel the program’s hierarchical structure, make optimal assignments to controllable choice nodes, and ensure that episode constraints activated during program unraveling are jointly feasible with high probability.

In order to establish a parallel with previous work on RMPL [Williams and In-

gham, 2002, Williams et al., 2003, Effinger, 2012] and define the semantics of valid executions of cRMPL programs, Section 4.5.1 starts by showing the relationship between cRMPL constructs, Hierarchical Constraint Automata (HCA), and Probabilistic Temporal Plan Networks (PTPN's). Next, Section 4.5.2 develops the CC-POMDP model that simulates valid executions of cRMPL programs, and shows how RAO* can generate optimal executions for these cRMPL programs by solving its CC-POMDP simulation using heuristic forward search.

4.5.1 Valid executions of a cRMPL program

Before presenting the details involved in the formal definition of valid cRMPL program executions, it is worthwhile to develop an intuitive understanding of the concept by making a parallel with existing general-purpose programming languages.

From a conceptual standpoint, valid executions of cRMPL programs are not very different from valid executions of programs written in any common programming language. First, any valid execution of a cRMPL program must start at its outermost “main” episode, which is at the top of the hierarchical composition of episodes that form the cRMPL program. This is directly analogous to the “main” function used as the entry point for programming languages such as C/C++ and Java, from which all other subprocedures are invoked. Second, the hierarchy of composite episodes in cRMPL must be recursively unraveled until they can be completely expressed in terms of *primitive* episodes, since these represent activities that can be directly executed by the underlying system that the cRMPL program is trying to control. In traditional programming languages such as C++, this would correspond to the compiler crawling the structure of function calls in the program to make sure that they can be eventually expressed as low-level instructions that the underlying processor can execute. Third, episodes in a cRMPL program must be dispatched according to the temporal and branching structure defined by its episode composition constructs, much in the same way that a program in C++ must strictly adhere to the execution flow imposed by language constructs such as `if-else`, as well as the ordering in which instructions appear in the program. Finally, a valid execution of a cRMPL program terminates at

a state where there are no more composite episodes to be unraveled, and all episode constraints activated during the unraveling of the program are jointly feasible with high probability. For a traditional programming language, that would correspond to reaching the end of the “main” function without any uncaught exceptions.

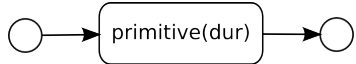
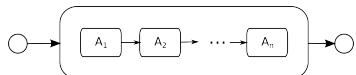
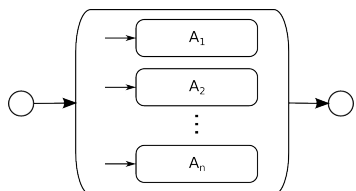
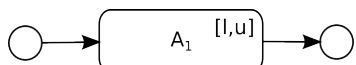
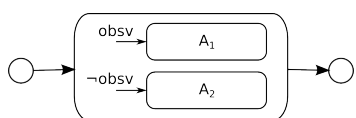
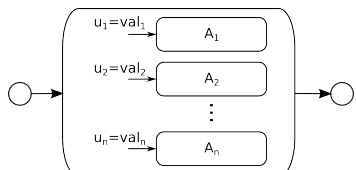
pRMPL [Effinger, 2012]	cRMPL	HCA
<code>primitive(dur)</code>	<code>Episode(action=primitive, duration=dur)</code>	
<code>sequence{A₁, A₂, ...}</code>	<code>sequence(A₁, A₂, ...)</code>	
<code>parallel{A₁, A₂, ...}</code>	<code>parallel(A₁, A₂, ...)</code>	
<code>[l, u] (name){A₁}</code>	<code>A₁.duration=[l, u]</code>	
<code>if (obsv){A₁}else{A₂}</code>	<code>observe(obsv, A₁, A₂)</code>	
<code>choose{A₁, A₂, ...}</code>	<code>decide(u₁, A₁, A₂, ...)</code>	

Table 4.1: Relationship between the pRMPL variant of [Effinger, 2012], cRMPL, and HCA [Williams et al., 2003].

Hierarchical Constraint Automata (HCA) have been used to define the execution semantics of previous variants of RMPL [Williams et al., 2003]. Therefore, seeking to place this thesis’ cRMPL within the context of prior work on the RMPL language, we begin this section by introducing HCA in Definition 4.3, and a parallel between HCA, pRMPL [Effinger, 2012] - on which cRMPL is based -, and cRMPL constructs in Table 4.1. By presenting Definition 4.3 and Table 4.1, our goal is twofold: first, we seek

to indicate that HCA and cRMPL episodes (Definition 4.2) are closely-related concepts, since both are *hierarchical* representations of *conditional* constraint networks that become active the moment an HCA location is marked, or a cRMPL episode is enabled; and second, we wish to draw a parallel between valid executions of cRMPL programs, which are defined in terms of the unraveling of cRMPL episodes, and valid executions of RMPL programs in [Williams et al., 2003], which are defined in terms of valid executions of an HCA. A possible mapping between an HCA H and an episode E is given below:

- Constraints: G and M , respectively the goal and maintenance constraints of H , can be combined to form the constraints \mathcal{C} of episode E ;
- State variables: the state variables \mathcal{V} of HCA H can be obtained from the scope of constraints in \mathcal{C} ;
- Locations: Boolean HCA locations \mathcal{L} correspond to enabling variables en in episodes;
- Initial marking: the initial marking \mathcal{L}_0 corresponds to the enabling of the episode representing the complete cRMPL program;
- Transitions: the HCA transition function T is implemented by the episode activity functions explained in Section 4.4.3. Appropriate assignments to V in each $T(l_i)$ can be represented as cRMPL Boolean choices enabled by l_i .

Definition 4.3 (Hierarchical Constraint Automaton (HCA) [Williams et al., 2003]).
An HCA is a tuple $\langle \mathcal{L}, \mathcal{L}_0, \mathcal{V}, G, M, T \rangle$ where

- \mathcal{L} : *set of locations, partitioned into primitive locations \mathcal{L}_p and composite locations \mathcal{L}_c . Each composite location denotes a hierarchical constraint automaton;*
- $\mathcal{L}_0 \subseteq \mathcal{L}$: *set of start locations (also called the initial marking);*
- \mathcal{V} : *set of plant state variables, with each $v_i \in \mathcal{V}$ ranging over a finite domain $\mathcal{D}[v_i]$. $\mathcal{C}[\mathcal{V}]$ denotes the set of all finite domain constraints over \mathcal{V} ;*

- $G : \mathcal{L}_p \rightarrow \mathcal{C}[\mathcal{V}]$: function that associates with each location $l_p^i \in \mathcal{L}_p$ a finite domain constraint $G(l_p^i)$ that the plant progresses towards whenever l_p^i is marked. $G(l_p^i)$ is called the goal constraint of l_p^i . Goal constraints $G(l_p^i)$ may be thought of as “set points”, representing a set of states that the plant must evolve towards when l_p^i is marked;
- $M : \mathcal{L} \rightarrow \mathcal{C}[\mathcal{V}]$: function that associates with each location $l_i \in \mathcal{L}$ a finite domain constraint $M(l_i)$ that must hold at the current instant for l_i to be marked. $M(l_i)$ is called the maintenance constraint of l_i . Maintenance constraints $M(l_i)$ may be viewed as representing monitored constraints that must be maintained in order for execution to progress towards achieving any goal constraints specified within l_i ;
- $T : \mathcal{L} \times \mathcal{C}[\mathcal{V}] \rightarrow 2^{\mathcal{L}}$: function that associates with each location $l_i \in \mathcal{L}$ a transition function $T(l_i)$. Each $T(l_i) : \mathcal{C}[\mathcal{V}] \rightarrow 2^{\mathcal{L}}$ specifies a set of locations to be marked at time $t + 1$, given appropriate assignments to V at time t .

As explained in [Williams et al., 2003], HCA are used to symbolize the different execution stages of an RMPL program by a *control sequencer*, and how transitions between different execution states occur. A state in the execution of an HCA is called a *marking*, and denotes all *marked* HCA locations. Starting at the initial marking \mathcal{L}_0 , the control sequencer creates a set consisting of each marked location whose maintenance constraint is satisfied by the current Plant state³. Next, it conjoins all goal constraints of this set to produce a configuration goal, towards which the Plant state must be driven by the *mode reconfiguration* module. After the latter executes a single command that makes progress towards achieving the goal constraints, the control sequencer receives an update of the Plant state, which is used by the HCA to advance to a new marking. This is achieved by taking all enabled transitions from marked primitive locations whose goal constraints are achieved or whose maintenance constraints have been violated, and from marked composite locations that no longer

³In [Williams et al., 2003], the current Plant state is replaced by its maximum a posteriori (MAP) estimate in the presence of uncertainty.

contain any marked subautomata. This cycle repeats until an *empty marking* is achieved, at which point program execution is deemed completed.

A similar process defined in terms of the unraveling of cRMPL episodes is used in Definition 4.4 to describe the properties of valid executions of cRMPL programs.

Definition 4.4 (Valid execution of a cRMPL program). *A valid execution of a cRMPL program must have the following properties:*

1. *It must enable the episode at the top of the episode composition hierarchy;*
2. *It must unravel all enabled episodes;*
3. *Unraveling **sequence** and **parallel** composite episodes enables all of its component episodes and constraints;*
4. *Unraveling **controllable choice** episodes only enables a single choice branch corresponding to the choice assignment (decision);*
5. *Unraveling **uncontrollable choice** episodes enables all choice branches (observation);*
6. *It ends when there are no more episodes to unravel, and the conditional constraints enabled by the program execution are jointly feasible with high probability.*

Representing plan constraints by means of hierarchical compositions of cRMPL episodes is a useful tool from a modeling standpoint, for it allows the programmer to design the mission at a high level of abstraction while relying on the program executive to reason about lower level actions that drive the Plant towards the fulfillment of these constraints. However, as first introduced in [Kim et al., 2001] and explained in Definition 4.4, executing a cRMPL program requires the *unraveling* of this hierarchical program structure in order to expose its essential elements: *controllable choices*, which represent decisions that the program executive must make at runtime; *uncontrollable choices*, which model runtime observations that may affect the way the program is

executed (à la the ubiquitous `if-then-else` construct); *primitive* activities, which are the ones that can be directly executed to control the Plant; and *conditional constraints*, which stem from the enabling of their container episodes by means of sequences of assignments to choice variables, as defined in (4.2). As first mentioned in Section 1.4, we refer to this unraveling of cRMPL programs as a Probabilistic Temporal Plan Network (PTPN) [Levine and Williams, 2014, Santana and Williams, 2014], an extension of previous TPN’s and TPNU’s [Kim et al., 2001, Walcott, 2004, Effinger et al., 2009] (Table 4.2). Definition 4.5 presents a labeled representation of PTPN’s based on [Tsamardinos et al., 2003, Santana and Williams, 2014], so as to provide a smoother transition into the next section, where we show that RAO* can compute optimal, chance-constrained execution policies for cRMPL programs by framing cRMPL execution as a CC-POMDP instance.

Definition 4.5 (Probabilistic Temporal Plan Network (PTPN)). *A PTPN is a tuple $P = \langle \mathcal{L}, \mathcal{D}, \mathcal{U}, \mathcal{E}_p, \mathcal{C}, R, O \rangle$, where*

- \mathcal{D} : set of labeled controllable choices (decisions) with finite, discrete domains;
- \mathcal{U} : set of labeled uncontrollable choices (observations) ranging over a finite, discrete universe;
- \mathcal{L} : set of Boolean labels (also referred to as guard conditions) of the form

$$L = a_1 \wedge a_2 \wedge \dots \wedge a_n, \tag{4.6}$$

where a_i is an assignment $c_i = v$, $v \in \text{Domain}(c_i)$, $c_i \in (\mathcal{D} \cup \mathcal{U})$. We use the notation $L \Rightarrow e$ to denote that L enables an element e ;

- \mathcal{E}_p : set of labeled primitive episodes;
- \mathcal{C} : set of labeled episode constraints;
- $R : \mathcal{D} \rightarrow \mathbb{R}$: function mapping decisions to numerical rewards;
- $O : 2^{\mathcal{U}} \rightarrow [0, 1]$: joint probability distribution of uncontrollable choices.

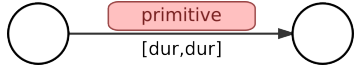
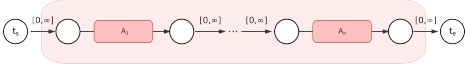
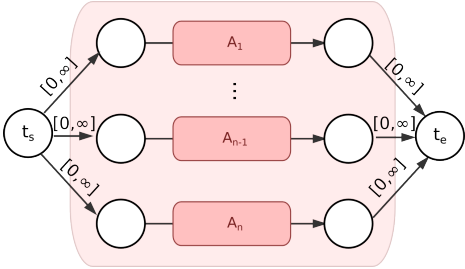
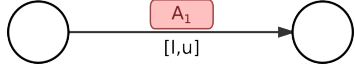
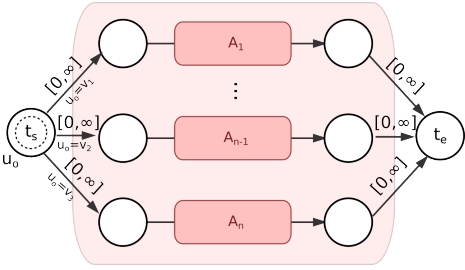
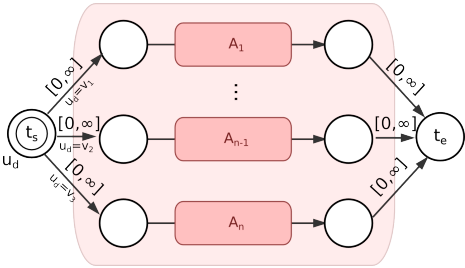
cRMPL	PTPN
Episode(action=primitive, duration=dur)	
sequence(A_1, A_2, \dots)	
parallel(A_1, A_2, \dots)	
$A_1.duration=[l, u]$	
observe(obsv, A_1, A_2)	
decide(u_1, A_1, A_2, \dots)	

Table 4.2: Relationship between cRMPL and PTPN constructs.

Intuitively, a PTPN is a graph representation of possible executions of a cRMPL program, with labels denoting “choice paths” (regardless of whether they are controllable or not) that cause episodes E to be enabled. For instance, in Figure 4-10, let $Transp \in \{\text{Bike}, \text{Car}, \text{Stay}\}$ be the controllable choice that selects a type of transportation and $Crash \in \{\text{True}, \text{False}\}$ be the uncontrollable choice that governs

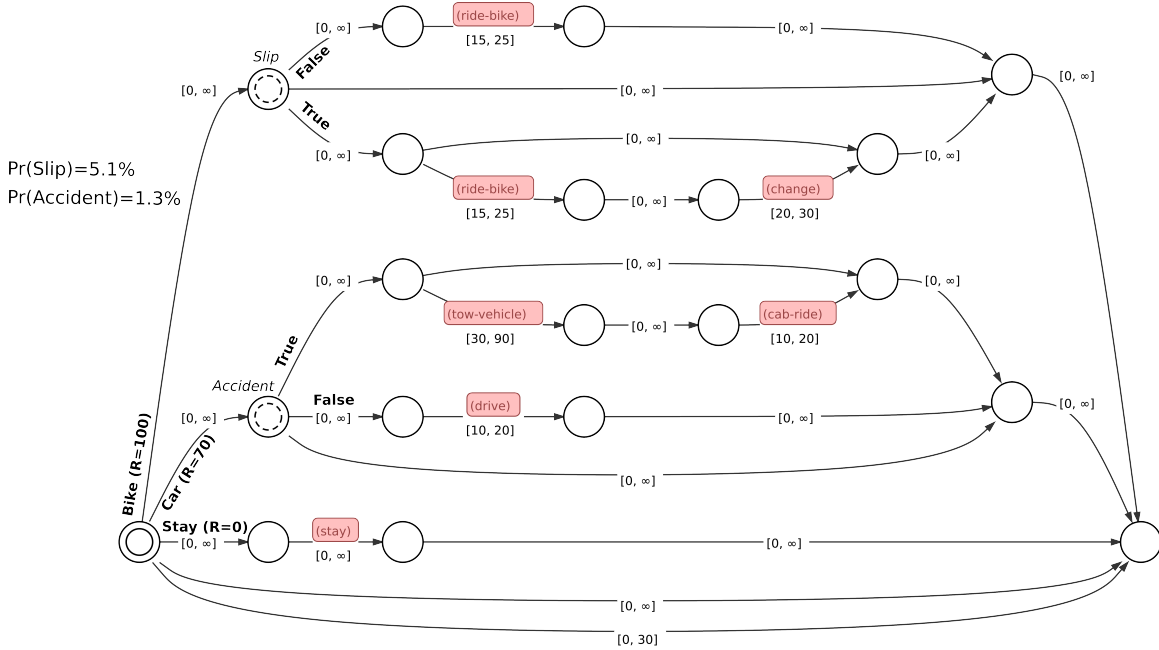


Figure 4-10: PTPN example (repeated from Figure 1-12 on page 43).

whether there will be an accident if one chooses to drive to work. In this case, the label for the episode with activity “(stay)” is $Transp = Stay$, while the episode with activity “(cab-ride)” has label $Transp = Car \wedge Crash = True$. With that said, let E_c be a component of a composite episode E_p and u_p be a $\langle Choice \rangle$ element. From (4.1) and (4.2), we have

$$L(E_c) = \begin{cases} L(E_p), & \text{for sequence or parallel composition,} \\ L(E_p) \wedge (u_p = v), & \text{for choose and } E_c \text{ associated with } u_p = v. \end{cases} \quad (4.7)$$

The termination condition $L(E_{prog}) = True$ for the episode representing the complete cRMPL program allows episode labels to be recursively computed by (4.7).

The apparent simplicity of PTPN’s may deceive the reader into not noticing the world of intractability that they hide, given that their size grows exponentially with the number of choices. In complex applications where the cRMPL program executive must reason over a space of hundreds, thousands, or potentially even an infinite⁴

⁴A PTPN with an infinite number of choices could be generated by an iteration operator from Section 4.4.3.

number of choices, there might not even be enough space and time to finish writing the PTPN before we start analyzing it. This is the reason why previous methods that require a fully unraveled PTPN as input [Conrad and Williams, 2011, Santana and Williams, 2014, Levine and Williams, 2014] tend to not scale well even for modest size problems. Seeking to address this problem, the next section presents a mapping from cRMPL execution to CC-POMDP that allows RAO* to exploit the hierarchical structure of cRMPL programs and focus its unraveling on program traces that look promising, as guided by RAO*'s heuristics.

4.5.2 Execution of cRMPL programs as CC-POMDP

In previous approaches to decision-theoretic⁵ RMPL execution [Kim et al., 2001, Effinger, 2012, Levine and Williams, 2014, Santana and Williams, 2014], explicit representations of all possible program execution states, e.g., a PTPN-like structure [Kim et al., 2001, Levine and Williams, 2014, Santana and Williams, 2014] or the complete state space of an MDP with discretized activity durations [Effinger, 2012], are used to compute optimal, or just feasible, program execution policies. However, as pointed out in Sections 2.2 and 4.5.1, the exponential growth in the size of these explicit representations as a function of the number of choices in the program limits the complexity of the problems that can be effectively solved in practice. In order to see that, consider the simple decision-making scenario described in the following, along with its cRMPL⁶ description in Figure 4-11:

“Peter and his friends went to an amusement park on a beautiful summer weekend, and he is set on riding the roller coasters as many times as possible. However, he has only two hours left, and every round through the queue takes between 45 minutes to an hour. What should Peter do?”

⁵Decision-theoretic programming languages are those that contain controllable choice (decision) constructs that can be assigned by the program executive.

⁶In case the reader is wondering, the `lambda` function is needed to make sure that roller coaster-riding episodes in the loop are represented as distinct instances.

```

prog = RMPyL()
ride_gen = lambda: Episode(action='ride',
                           duration={'ctype': 'uncontrollable_bounded',
                                     'lb': 45, 'ub': 60})
prog *= prog.loop(ride_gen, run_utility=1, stop_utility=0)
prog.add_overall_temporal_constraint(ctype='controllable', lb=0, ub=120)

```

Figure 4-11: Roller coaster-riding scenario described in cRMPL.

This problem is so simple that it can be solved by inspection: two roller coaster rides would take between 90 and 120 minutes to complete, while three rides would take at least 135 minutes. Therefore, given the 120 minute time limit, the optimal policy for Peter is to ride the roller coaster twice and leave. However, it could not be solved by first unraveling the cRMPL in Figure 4-11 into a PTPN, for the PTPN would be infinite. This anecdotal example serves as motivation for our approach to generate optimal, chance-constrained execution policies for cRMPL programs: frame valid cRMPL execution as a CC-POMDP and solve it using RAO*'s heuristic forward search strategy.

Figure 4-12 shows the unraveling of the cRMPL program in Figure 4-11 using our strategy. The advantage of doing so is to allow the hierarchical structure of cRMPL to be incrementally unraveled by RAO*, as it is guided by the utility of the controllable choices assigned so far and a heuristic estimate of the utility-to-go. In order to understand the unraveling steps depicted in Figure 4-12, it is worthwhile to recall from (4.5) that `loop` is implemented in cRMPL as a choice between a `sequence` of an iteration episode followed by another `loop`, or halting the loop. For the program in Figure 4-11, after unraveling the loop twice (assigning “RUN” to loop choices), the probabilistic scheduler from Chapter 5 indicates to RAO* that riding the roller coaster a third time violates the overall temporal constraint of being done in 120 minutes, causing RAO* to stop the iteration (assigning “STOP” to the third loop choice from the top) and halt the unraveling process (last operator at the second to last state from the bottom). Notice that, as expected, RAO* assigns a value of 2 to this execution, as indicated by the value field at the top initial state. Our mapping from valid, optimal

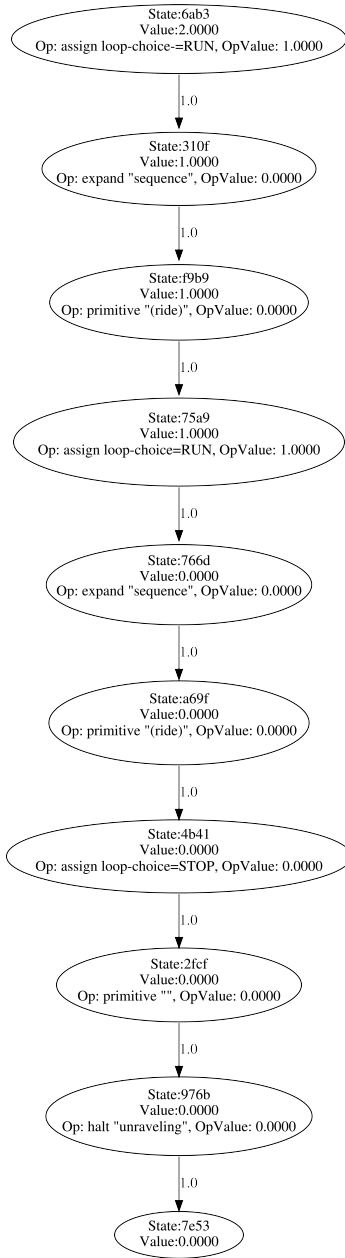


Figure 4-12: Incremental unraveling of the cRMPL program from Figure 4-11. Temporal consistency is checked by PARIS, the probabilistic scheduling algorithm described in Chapter 5.

cRMPL execution to CC-POMDP is formally explained in Definition 4.6.

Definition 4.6 (cRMPL execution as CC-POMDP). *The following CC-POMDP is used to compute chance-constrained, optimal executions of a cRMPL program p_g .*

- \mathcal{C} : universe of all possible labeled constraints in p_g ;

- \mathcal{S} : states $s \in \mathcal{S}$ are tuples $s = \langle en, de, cs \rangle$, where en is a list of enabled episodes that must be unraveled; de is a list of assignments to controllable choices (decisions); and $cs \subseteq \mathcal{C}$ is a reference to a global constraint store, which records labeled constraints that have been enabled during the search process;
- b_0 : deterministic belief state with initial state $s_0 = \langle [e_{p_g}], \emptyset, \text{Constraints}(e_{p_g}) \rangle$, where e_{p_g} is the top, “main” episode in p_g ’s hierarchy, and $\text{Constraints}(e_{p_g})$ is the set of labeled constraints of e_{p_g} . Notice that, since e_{p_g} must be enabled in any valid execution of p_g , the constraints in the set $\text{Constraints}(e_{p_g})$ must necessarily have empty labels, i.e., they are always enabled;
- \mathcal{O}, O : the model is fully observable, so these are respectively \mathcal{S} and the identity function;
- \mathcal{A} : there are four available actions:
 - Assign ($a_=(D) = v$): assigns a value v to a controllable choice episode (decision) D ;
 - Observe ($a_o(O)$): observes an uncontrollable choice episode (observation) O ;
 - Expand ($a_u(E)$): unravels an episode E ;
 - Halt (a_h): halts execution.
- T : there are four types of transitions:
 - $T(s_1, a_=(D) = v, s_2) = 1.0$ is a deterministic transition, where the enabled list en in s_2 is the same as in s_1 , except that the controllable choice episode D is replaced by its internal component episode associated with the assignment $D = v$; the decisions de for s_2 are the same as in s_1 , plus the assignment $D = v$; and the labeled constraints from the controllable choice episode D are added to the global constraint store cs ;
 - $T(s_1, a_o(O), s_2(o)) = p$ is a stochastic transition with probability p corresponding to the observation $O = o$; the enabled list en in $s_2(o)$ is the same

- as in s_1 , except that the uncontrollable choice episode O is replaced by its internal component episode associated with the observation $O = o$; the decisions de for $s_2(o)$ are the same as in s_1 ; and the labeled constraints from the uncontrollable choice episode O are added to the global constraint store cs ;
- $T(s_1, a_u(E), s_2) = 1.0$ is a deterministic transition, where the enabled list en in s_2 is the same as in s_1 , except that E is replaced by its components if it is a `sequence` or `parallel` episode, or nothing if it is a primitive episode; the decisions de for s_2 are the same as in s_1 ; and the labeled constraints from the episode E are added to the global constraint store cs ;
 - $T(s_1, a_h, s_{cv}) = p_{cv}$ is a stochastic transition, where s_1 must be a state with an empty enabled list en ; and s_{cv} is a terminal state with an empty enabled list and the same list of decisions de as s_1 . In this transition, nothing is added to the global constraint store cs . The state s_{cv} represents the execution of the cRMPL program ending in a state that violates constraints, and the probability p_{cv} of this event happening depends on the semantic interpretation of $C(s_1) \subseteq cs$, the subset of labeled constraints in the constraint store that is consistent with the decisions de in s_1 . For instance, if $C(s_1)$ forms a Probabilistic Simple Temporal Network with Uncertainty (PSTNU), the PARIS algorithm in Chapter 5 can check if a strong schedule for $C(s_1)$ exists for different levels of scheduling risk, and the probability p_{cv} would correspond to the scheduling risk bound returned by PARIS. On the other hand, if the temporal constraints in $C(s_1)$ are dependent on assignments to both uncontrollable and controllable choices, the extension of PARIS to PTPN scheduling in Chapter 6 can be used to check if $C(s_1)$ is strongly consistent, and again the probability p_{cv} would correspond to the scheduling risk bound. The remaining transition probability is associated with $T(s_1, a_h, s_{sa}) = 1 - p_{cv}$, where s_{sa} is identical to s_{cv} , except for the fact that it marks that the program ends in a state that does not violate constraints.

- $R(s, a_{\perp}(D) = v) = r$, where r is the utility value associated with $a_{\perp}(D) = v$, and 0 otherwise;
- c_v : in this model, we have $c_v(s_{cv}, \cdot) = 1$ for the terminal state s_{cv} representing constraint violations, and $c_v = 0$ everywhere else.
- Δ : vector of execution risk bounds for the violation of $C(s) \subseteq cs$.

In the CC-POMDP model explained in Definition 4.6, it is important to point out that, even though the constraint violation function c_v only returns 1 at *terminal* execution states, that does not mean that RAO* can only assess (or estimate) the execution risk arising from temporal uncertainty (or any other type of constraint involved in the cRMPL program) at the very end of its policy expansion. As seen in Chapter 3, RAO* leverages an admissible (lower bound) heuristic estimate of the execution risk $h_{er}(s)$ at *non-terminal* states s in the hopes of performing early pruning of policy nodes that are guaranteed to violate chance constraints. For instance, in the case where $C(s) \subseteq cs$, the subset of labeled constraints in the constraint store that is consistent with the decisions de in s , form a PSTNU, an admissible estimate $h_{er}(s)$ is given by the *minimum risk* formulation of PARIS (see Section 5.4).

Next, Lemma 4.3 shows that the CC-POMDP model in Definition 4.6 simulates valid executions of its corresponding cRMPL program.

Lemma 4.3. *The CC-POMDP in Definition 4.6 simulates valid executions of its corresponding cRMPL program.*

Proof: We show this property by verifying that program executions simulated by the CC-POMDP in Definition 4.6 satisfy all properties in Definition 4.4.

As required by properties 1 and 2 of valid executions of cRMPL programs, the execution simulated by the CC-POMDP in Definition 4.6 starts at the “main” program episode, proceeding until all episodes have been unraveled into their primitive components and the latter have been all taken out of the list enabled episodes. Properties 3, 4, and 5 are implemented, respectively, by the state transition function T

with the expansion (a_u), assignment (a_+), and observation (s_o) actions. Finally, property 6 is enforced by the halting action a_h and the constraint violation functions c_v . \square

With the result from Lemma 4.3, Lemma 4.4 shows that RAO* can be used to generate optimal execution policies for the class of cRMPL programs that this thesis focuses on.

Lemma 4.4. *RAO* can compute optimal, chance-constrained, deterministic, valid execution policies for cRMPL programs with finite-duration activities that must eventually terminate.*

Proof: Lemma 4.3 shows that valid executions of a cRMPL program can be simulated as a CC-POMDP. For cRMPL programs representing temporal planning problems that must eventually terminate and with finite-duration primitive actions, there is a finite (although potentially unknown) execution horizon $h = \lceil t_{\max}/d_{\min} \rceil$, where t_{\max} is the maximum continuous temporal plan length, and d_{\min} is the shortest primitive episode duration. Therefore, an optimal, chance-constrained, deterministic, and valid execution policy for the cRMPL program can be obtained by formulating its execution as CC-POMDP shown in Definition 4.6, and computing an execution policy that terminates within a finite number of steps with RAO*. \square

Seeking to provide an intuitive understanding of how the CC-POMDP in Definition 4.6 is used to generate optimal, chance-constrained, valid executions of cRMPL programs, the next section provides further details on the coupling between RAO* and PARIS used to compute the execution policy in Figure 4-12.

Walk-through of the roller coaster example

The recursive nature of the roller coaster example from the beginning of Section 4.5.2 is illustrated in Figure 4-13, which shows a fully unraveled PTPN for the cRMPL program in Figure 4-11 when the number of loop iterations is limited to be no more than 5. By developing the mapping from cRMPL execution to CC-POMDP in Def-

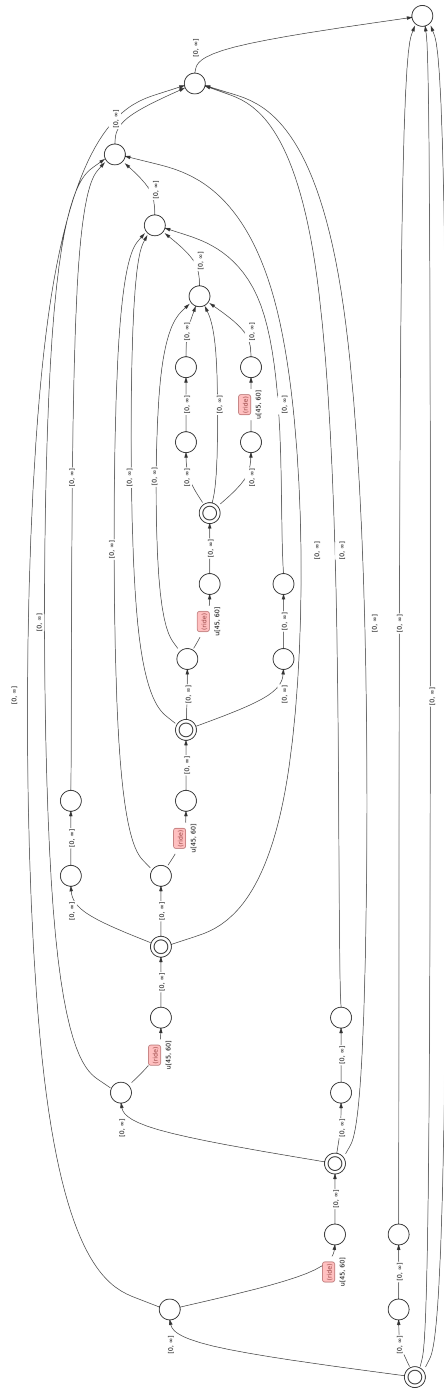


Figure 4-13: Fully unraveled PTPN for the cRMPL program in Figure 4-11 when the number of loop iterations is limited to be no more than 5.

initiation 4.6, and using RAO* to incrementally compute optimal, valid executions of a cRMPL program such as the one shown in Figure 4-12, our goal is to avoid the inherent intractability of previous approaches to RMPL execution, which operate on

explicit representations of all possible executions of the program, such as the PTPN in Figure 4-13.

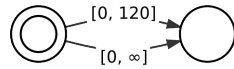


Figure 4-14: Elements of the constraint store at b_0 , the initial (deterministic) belief state for the execution of the cRMPL program in Figure 4-11.

The unraveling of the cRMPL program in Figure 4-11 starts at the initial belief state b_0 from Definition 4.6, where the outermost loop episode is enabled, no assignments to controllable choices have been made, and the constraint store contains the two temporal constraints shown in Figure 4-14 (a thorough presentation of temporal constraints and scheduling under uncertainty is given in Chapter 5, along with the PARIS algorithm). In this figure, the double circle represents the decision of whether to run the loop or not, as defined in (4.5), and the circle to the right is the temporal event representing the end of program execution. The simple temporal constraint $[0, \infty]$ is automatically added by cRMPL and means that the time of the first decision about running the loop must not come after the end of the program. The additional $[0, 120]$ comes from the user requirement in Figure 4-11 that limits program execution to take no more than 120 minutes (or any other unit of time being used in the program). These two constraints are exactly the elements returned by $Constraints(e_{pg})$ in Definition 4.6. Since the cRMPL program in Figure 4-11 specifies no execution risk bound for temporal consistency, a bound $er(b_0|\pi) \leq \Delta = 0$ (no scheduling risk) is automatically assumed. Since the root node in Figure 4-12 is not terminal, the *minimum risk* formulation of PARIS from Section 5.4 is used to compute a lower bound on the scheduling risk for the program. Since the temporal constraints shown in Figure 4-14 are trivially satisfiable, PARIS returns a lower bound of 0, which is consistent with $\Delta = 0$, and the unraveling proceeds.

Figure 4-15 shows the constraints $C(s_{1r}) \subseteq cs$ at the unraveling state s_{1r} after the first (**ride**) primitive is unraveled and taken off the list of enabled episodes. In this figure, we see that several additional temporal constraints and events have been inserted between the events marking the first loop decision (double circle on

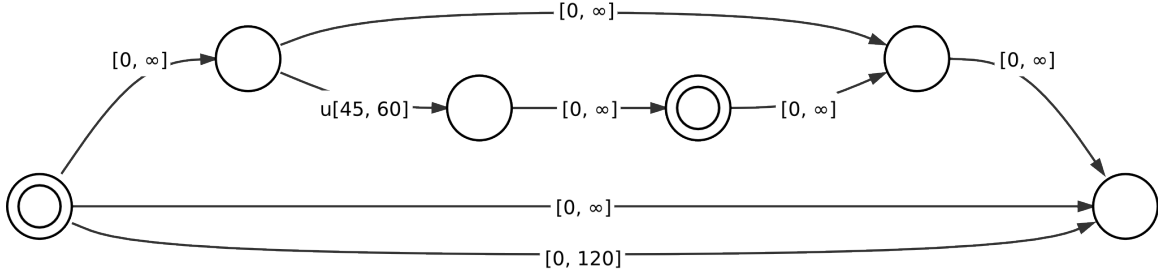


Figure 4-15: Temporal constraints tested for consistency with PARIS after the first (**ride**) primitive is unraveled.

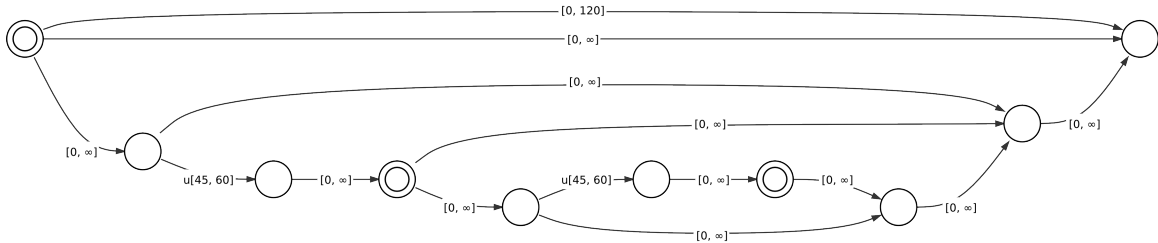


Figure 4-16: Temporal constraints tested for consistency with PARIS after the second (**ride**) primitive is unraveled.

the bottom left) and the end of the program (circle on the bottom right), which are still connected by the $[0, 120]$ simple temporal constraint. At this intermediate, non-terminal state, the minimum risk formulation of PARIS is again used to estimate a lower bound on the probability of program execution violating temporal requirements and, since the constraints in Figure 4-15 are guaranteed to be jointly feasible, it returns a lower bound of 0 (the program is guaranteed to be temporally feasible so far). A similar situation is depicted in Figure 4-16, which shows the constraints $C(s_{2r}) \subseteq cs$ at the unraveling state s_{2r} after the second (**ride**) primitive is unraveled and taken off the list of enabled episodes. At this point, RAO* is required to assign a value (“RUN” or “STOP”) to the loop decision variable shown as the rightmost double circle in Figure 4-16.

Since assigning “RUN” to loop decisions is preferred over “STOP” (Peter is trying to ride the roller coaster as many times as possible), RAO* first considers the most promising option of running the loop a third time, thus entailing the constraints $C(s_{3r}) \subseteq cs$ shown in Figure 4-17. Since this is a non-terminal state, minimum risk PARIS is once again used to estimate a lower bound on the scheduling risk for this

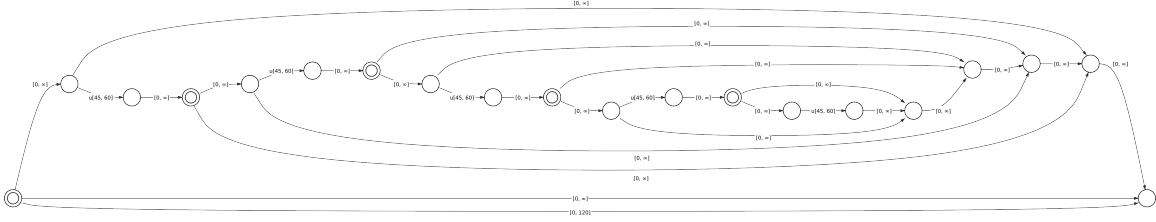


Figure 4-17: Temporal constraints tested for consistency with PARIS after the third (ride) primitive is unraveled. Unlike the previous cases, here PARIS returns that no strongly consistent schedule exists.

program. However, unlike the previous cases, the temporal constraints in Figure 4-17 are guaranteed *not to be jointly feasible*, i.e., PARIS returns a scheduling risk lower bound of 100%. Even though the specificities of the algorithm are only given in Chapter 5, the intuitive understanding of why that is the case is the same one given at the beginning of Section 4.5.2: since no probability distribution is specified for the uncontrollable duration $[45, 60]$ of each roller coaster ride, PARIS resorts to an analysis of the extrema of this interval and concludes that, even if every roller coaster ride takes its minimum duration of 45 minutes, this would still be in violation of the overall time window of 120 minutes to complete the execution of the program. Since an execution risk lower bound of 100% is certainly in violation of the chance constraint $\Delta = 0$, RAO* is able to prune this policy branch that executes the loop three times even before reaching terminal belief states, thus providing evidence of the importance of estimating the execution risk of partial CC-POMDP policies as the search for a solution unfolds.

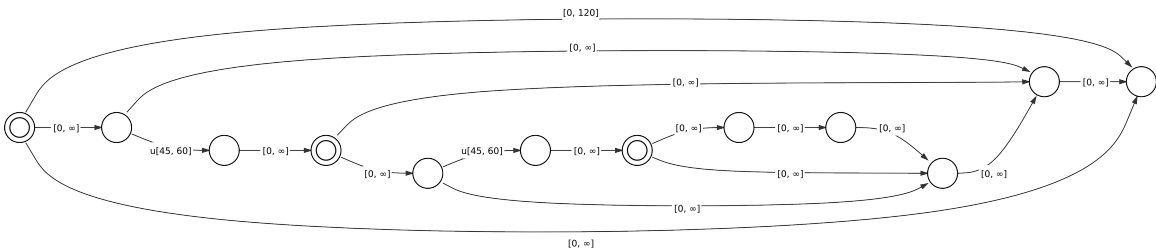


Figure 4-18: Temporal constraints tested for consistency at the halting state in the RAO* policy shown in Figure 4-12. It corresponds to two runs of the loop, followed by the decision to stop the iteration. As required by valid cRMPL executions in Definition 4.4, these constraints are jointly feasible with high probability (100%, in this example).

After pruning the policy branch that runs the roller coaster loop three times, the remaining most promising option is to assign “STOP” to the third loop decision, in which case the decision episode is replaced by a no-op activity with controllable $[0, \infty]$ duration marking the end of the iteration, as shown in Figure 4-18. Since this is a state with no more enabled episodes to unravel, a halting action a_h is generated by the CC-POMDP model in Definition 4.6, and the probability of constraint violation p_{cv} is obtained by running minimum risk PARIS on the temporal constraints shown in Figure 4-18. Since these temporal constraints are guaranteed to be jointly feasible, PARIS returns a probability $p_{cv} = 0$, which means that the cRMPL execution transitions to a terminal state that is guaranteed to fulfill its temporal requirements, therefore satisfying the chance constraint $er(b_0|\pi) \leq 0$. The total utility for this program is 2, corresponding to the two roller coaster rides, and the execution risk measuring the probability of violating temporal constraints is 0. This is exactly the result shown in Figure 4-12.

4.6 Conclusions

This chapter introduces cRMPL, a programming language for autonomous systems that is innovative in its support to chance-constrained planning, and that allows risk-aware missions to be specified at a high level of abstraction. As all previous variants of RMPL [Ingham et al., 2001, Williams et al., 2001, Williams and Ingham, 2002, Williams et al., 2003, Ingham, 2003, Effinger, 2012], cRMPL is particularly useful in situations where mission operators desire to exert tight control over the decisions available to an autonomous agent and how it reacts to its sensors, while offloading the burden of lower level plan dispatching to a program executive.

In addition to supporting chance constraints, another important concept presented in this chapter is the framing of cRMPL execution as a CC-POMDP that simulates the process of stepping through program states. Since RAO* in Chapter 3 is an algorithm that allows solutions to a CC-POMDP to be incrementally constructed, our approach to the generation of optimal execution policies for cRMPL programs in Section 4.5.2 is

in contrast to [Kim et al., 2001, Conrad and Williams, 2011, Effinger, 2012, Levine and Williams, 2014], which require a complete, exponentially large unraveling of a PTPN representing all traces of a cRMPL prior to the selection of an optimal execution policy.

Also related to the framing of cRMPL execution as a CC-POMDP, it allows us to depart from the time discretization approach in [Effinger, 2012] and, similar to state-of-the-art temporal planners [Coles et al., 2009, Coles et al., 2012, Cimatti et al., 2015, Wang, 2015], determine the existence of a feasible schedule by reasoning over temporal constraint networks featuring continuous time. In fact, the last statement is valid for any CC-POMDP model featuring temporal constraints, not only those resulting from executing cRMPL programs. There are, however, three important observations to be made about the scheduling of cRMPL programs: first, it must support a rich combination of different types of controllable and uncontrollable temporal constraint representations, as described in Section 4.4.2; second, determining the existence of a schedule must be done quickly, for RAO* performs several constraint feasibility checks as it constructs a policy; and third, the scheduling of events may be conditional on probabilistic observations collected during plan execution. The first and second points are addressed in Chapter 5, while the third point is the topic of Chapter 6.

Chapter 5

Risk-sensitive unconditional scheduling under uncertainty

“Tempus fugit.”

Virgil.

This thesis’ title reads “Dynamic Execution of Temporal Plans with Sensing Actions and Bounded Risk”. In its development of CC-POMDP’s and RAO*, Chapter 3 focuses on “Dynamic Execution of Plans with Sensing Actions and Bounded Risk” without paying particular attention to any specific type of constraints, but justifying our decision to focus on finite horizon policies as being motivated by agents that must execute plans under time pressure. Next, cRMPL in Chapter 4 gives mission operators the ability to specify risk-bounded planning problems with any mix of controllable (STN-like), set-bounded (STNU-like), and probabilistic (PSTN-like) temporal constraints, whose impact on the mission schedule might depend on real-time sensor observations.

In this chapter and the next, our goal is to insert the *temporal* aspect into policies generated by CLARK, by providing novel and efficient algorithms that can check the feasibility of a (conditional) temporal network featuring duration uncertainty. Inspired by risk-sensitive robust scheduling for planetary rovers, this chapter introduces the Probabilistic Simple Temporal Network with Uncertainty (PSTNU), a temporal

constraint formalism that unifies the set-bounded and probabilistic temporal uncertainty models from the STNU and PSTN literature, and describes PARIS, a novel sound and provably polynomial-time algorithm for risk-sensitive strong scheduling of PSTNU's.

Due to its linear encoding of typical temporal uncertainty models, PARIS is shown to outperform the current fastest algorithm for risk-sensitive strong PSTN scheduling by nearly four orders of magnitude in some instances of a popular probabilistic scheduling dataset [Fang et al., 2014], often reducing the time to verify schedule feasibility from hours to fractions of a second. The speed at which PARIS can verify the existence of a strong schedule for a PSTNU is key for this thesis' practical use, for it allows CLARK to generate risk-bounded temporal plans featuring duration uncertainty within reasonable amounts of time.

5.1 Introduction

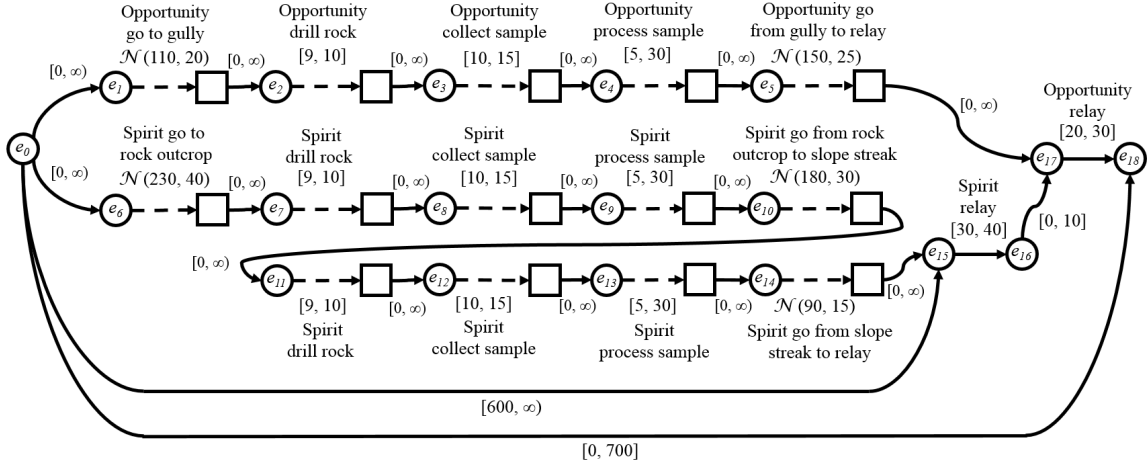
Endowing autonomous agents with a keen sensitivity to temporal uncertainty is key in enabling them to reliably complete time-critical missions in the real-world. This requirement has been the subject of recent research initiatives to incorporate duration uncertainty into temporal planning frameworks [Beaudry et al., 2010, Cimatti et al., 2015, Micheli et al., 2015]. A dominant effort has been to extend the set-bounded uncertainty model from Simple Temporal Networks with Uncertainty (STNU's) [Vidal, 1999] to a probabilistic setting, in which the risk of violating deadlines and other temporal requirements can be quantified [Tsamardinos, 2002, Fang et al., 2014, Wang and Williams, 2015a].

Three essential questions about Probabilistic Simple Temporal Networks (PSTN's) have been previously addressed. Tsamardinos in [Tsamardinos, 2002] showed that a risk-minimizing schedule of a PSTN could be derived analytically based on the probability density function (p.d.f) for the different stochastic durations. In his derivation, he reduced the PSTN to an STNU with variable bounds, and effectively solved for the bounds that would guarantee strong controllability. This reduction was further de-

veloped in the context of chance constraints, where one seeks a schedule whose risk of violating temporal requirements is guaranteed to be bound by a user-defined threshold. Operating in a general nonlinear optimization setting, Fang et al. [Fang et al., 2014] applied the STNU strong controllability reductions from [Vidal, 1999] to propose *Picard*, a chance-constrained strong scheduler for PSTN’s capable of optimizing any schedule-related objective function. Wang and Williams [Wang and Williams, 2015a], on the other hand, present *Rubato*, an efficient algorithm for verifying the existence of strong chance-constrained schedules that leverages a conflict-directed generate-and-test architecture to propose candidate STNU’s that meet the chance constraints and discover temporal conflicts.

Although the aforementioned previous work introduces effective methods for dealing with temporal uncertainty and scheduling risk, two main challenges remain. First, neither STNU’s, nor PSTN’s, accurately capture the varying levels of knowledge that a mission operator might have about the sources of temporal uncertainty that can have an impact on a mission. While set-bounded uncertainty in STNU’s fails to represent frequency models for random quantities captured by probability distributions, the fully probabilistic models in a PSTN assumes a potentially unreasonable level of understanding of the natural phenomena that give rise to uncertain temporal behavior. Therefore, a first contribution of this chapter is to unify STNU and PSTN models by introducing Probabilistic Simple Temporal Networks with Uncertainty (PSTNU’s).

Our second and most important contribution is related to the complexity of scaling current probabilistic scheduling methods, especially if one has in mind the goal of running such algorithms on embedded robotic hardware, where on-board computation and energy is scarce. While the state of the art to date employs general-purpose nonlinear solvers to implement probabilistic scheduling methods, this chapter introduces the **P**olynomial-time **A**lgorithm for **R**isk-aware **S**cheduling (PARIS), a new and provably polynomial-time algorithm for strong scheduling of PSTNU’s. Inspired by linear approximation techniques used to reduce the complexity of AC power flow analysis [Coffrin and Van Hentenryck, 2014], PARIS leverages a *fully linear* encoding of the risk-aware probabilistic scheduling problem that allows it to drastically reduce



(a) Probabilistic Simple Temporal Network with Uncertainty (PSTNU).

$e_0=0$	$e_1=0$	$e_2=132.35$
$e_3=142.35$	$e_4=157.35$	$e_5=187.35$
$e_6=0$	$e_7=246.64$	$e_8=256.64$
$e_9=271.64$	$e_{10}=301.64$	$e_{11}=495.75$
$e_{12}=505.75$	$e_{13}=520.75$	$e_{14}=550.75$
$e_{15}=650$	$e_{16}=680$	$e_{17}=680$
$e_{18}=700$		

(b) Activity schedule computed by the Polynomial-time Algorithm for Risk-aware Scheduling (PARIS).

Figure 5-1: Rover coordination under temporal uncertainty. (a) Scenario representation as PSTNU. (b) Strong activity schedule for the PSTNU in (a) with scheduling risk bound of 6.7% (i.e., all temporal requirements met with probability of at least 93.3%).

runtime and memory requirements. Empirical evaluation on a probabilistic scheduling dataset first introduced in [Fang et al., 2014] shows a several-order-of-magnitude improvement over the current fastest algorithm for PSTN scheduling, while our new planetary rover-inspired PSTNU dataset indicates that PARIS is, indeed, amenable for deployment on embedded hardware.

5.1.1 Motivation: planetary rover coordination

Future planetary exploration missions will require an increased level of coordination between multiple spacecrafts under temporal uncertainty. Figure 5-1 depicts a planetary exploration scenario illustrating important requirements for such missions: two

autonomous rovers, *Spirit* and *Opportunity*, should explore three sites on a region of Mars (gully, slope streak, and rock outcrop). After exploring all sites, both rovers must travel to a relay site and transmit their findings to an orbiting satellite within a limited time window.

There are several complicating temporal factors. First, traversal times between locations are uncertain. However, rover mobility has been extensively studied, both from experience on Mars and in simulated environments. Therefore, these times can be modeled probabilistically based on the distance between sites and terrain features. Second, the science-gathering activities at each site have uncertain durations. Due to lack of real and simulated data, there are no uncertainty models to aid in the prediction of how long these activities will last, but lower and upper bounds are built into the rover firmware to prevent deadlocks. Finally, at the relay site, there is an absolute time window in which the satellite is in field-of-view. Seeking to maximize throughput, both rovers should transmit at approximately the same time, but their transmissions should not overlap.

Due to power and communication limitations, the rovers cannot coordinate during plan execution. Therefore, one needs to precompute a schedule that satisfies all temporal requirements, while being robust to the uncertainty in activity durations. In other words, this is a strong temporal planning problem, which the authors of [Micheli et al., 2015] note is applicable to many safety-critical autonomous missions. The work [Cimatti et al., 2015] approaches strong temporal planning by replacing the STN scheduler in COLIN [Coles et al., 2012] with an STNU strong controllability solver. In our scenario, we would like a risk-aware strong controllability solver that handles both probabilistic and set-bounded uncertain durations, which could be incorporated into a planner in a similar manner.

5.2 Background & PSTNU's

Motivated by the aforementioned rover coordination scenario, our risk-aware scheduling methods operate on Probabilistic Simple Temporal Networks with Uncertainty

(PSTNU's), a novel temporal modeling formalism unifying features from Simple Temporal Networks (STN's) [Dechter et al., 1991], Simple Temporal Networks with Uncertainty (STNU's) [Vidal, 1999], and Probabilistic Simple Temporal Networks (PSTN's) [Tsamardinos, 2002]. For the sake of completeness, we briefly review these concepts. Figure 5-2 shows the different elements in a PSTNU.

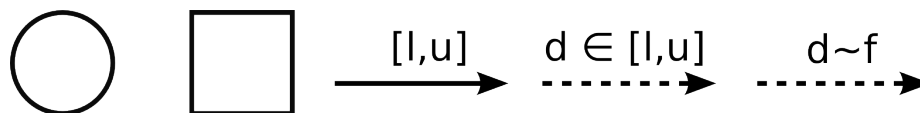


Figure 5-2: Elements of a PSTNU, where $[l, u]$ is a given interval and f is a known probability density function (pdf). From left to right: controllable event; contingent (uncontrollable) event; Simple Temporal Constraint (STC); STC with Uncertainty (STCU); Probabilistic STC (PSTC).

An STN (Definition 5.7) can model scheduling problems where the agent has control over the temporal assignments (a.k.a. the *schedule*) to all events. Possible assignments are restricted by simple temporal constraints (STC's), which limit the distance between the timing of two events. If there exists at least one schedule fulfilling all STC's, we say that the STN is *consistent*. Otherwise, it is *inconsistent*.

Definition 5.7 (STN [Dechter et al., 1991]). *An STN is a tuple $\langle \mathcal{E}_c, \mathcal{C}_r \rangle$, where*

- \mathcal{E}_c : set of **controllable** temporal events, all of which must be assigned by the scheduler;
- \mathcal{C}_r : set of **requirement** STC's of the form $l \leq e_2 - e_1 \leq u$, $l, u \in \mathbb{R} \cup \{-\infty, \infty\}$, where $e_1, e_2 \in \mathcal{E}_c$.

An STN is capable of modeling externally-imposed temporal requirements (e.g., “return to base in less than 30 minutes”) and durations of agent-controlled activities (e.g., “hibernate for 1 hour”), but is unable to model activities with uncertain durations (e.g., “drill a 2 cm hole in a rock” or “travel between two sites”). An STNU (Definition 5.8) addresses this limitation by extending STN's with *contingent* (also called *uncontrollable*) constraints and events. In an STNU, a contingent constraint is represented as an STC with uncertainty (STCU), which allows the difference between

two temporal events to be specified as non-deterministic, but bounded by a known interval $[l, u]$.

Depending on how much information about contingent durations is made available to the scheduler during execution, different levels of *controllability* for STNU's are defined in [Vidal, 1999]: *weak controllability* assumes all values of contingent durations to be known to the scheduler before it has to make any decisions; *strong controllability* assumes that no such information is ever available to the scheduler; and *dynamic controllability* assumes that the scheduler can only use information about past contingent durations when making future scheduling decisions.

Definition 5.8 (STNU [Vidal, 1999]). *An STNU is a tuple $\langle \mathcal{E}_c, \mathcal{E}_u, \mathcal{C}_r, \mathcal{C}_u \rangle$ that extends STN's by adding:*

- \mathcal{E}_u : set of **contingent** temporal events, which are assigned by an uncontrollable external agent (“Nature”);
- \mathcal{C}_u : set of **contingent** simple temporal constraints with uncertainty (STCU's) of the form

$$e_2 = e_1 + d, \quad l \leq d \leq u, \quad l, u \in \mathbb{R}_{>0}, \quad (5.1)$$

where d is an interval-bounded, non-deterministic duration; $e_1 \in \mathcal{E}_c \cup \mathcal{E}_u$; and $e_2 \in \mathcal{E}_u$.

- \mathcal{C}_r : same as in Definition 5.7, but we allow $e_1, e_2 \in \mathcal{E}_c \cup \mathcal{E}_u$.

Modeling contingent durations using STCU's is rather restrictive, for they do not incorporate information about the relative frequency of the different values for d in (5.1). For instance, they cannot model the statement “the rover requires 20 minutes to complete the traversal on average, with a standard deviation of 5 minutes”. To address this need, a PSTN (Definition 5.9) allows contingent durations to be represented as random variables with known probability distributions. The notions of controllability for STNU's can be readily transferred to PSTN's. Moreover, controllability for

PSTN’s have been further extended with a notion of *scheduling risk* [Tsamardinos, 2002, Fang et al., 2014, Wang and Williams, 2015a].

Definition 5.9 (PSTN [Tsamardinos, 2002]). *Similar to STNU’s, a PSTN is a tuple $\langle \mathcal{E}_c, \mathcal{E}_u, \mathcal{C}_r, \mathcal{C}_u \rangle$, where \mathcal{C}_u contains probabilistic simple temporal constraints (PSTC’s). A PSTC is also of the form $e_2 = e_1 + d$, with the additional assumption that d is a continuous random variable following a known probability distribution with positive support.*

While STNU’s ignore extra knowledge that one might have about temporal uncertainty, PSTN’s can err on the side of requiring *too much knowledge* to be available. As exemplified in Section 5.1.1, probabilistic models may not be known for every source of uncertainty affecting a mission. In most practical applications, these models are obtained through statistical analysis of experimental data, and “guessing” a model in the absence of data can lead to unquantifiable levels of mission risk. A PSTNU (Definition 5.10) addresses these issues by simply allowing the uncertainty models from STNU’s and PSTN’s to co-exist, so that mission operators can leverage probabilistic models for temporal uncertainty if, and only if, there is evidence to support them.

Definition 5.10 (PSTNU). *Same as a PSTN, but \mathcal{C}_u may contain any combination of STCU’s and PSTC’s.*

5.3 Problem formulation

Following [Vidal, 1999], given two sets of events $A \subseteq \mathcal{E}_c$ and $B \subseteq \mathcal{E}_u$, we denote a *control sequence* by $\delta : A \rightarrow \mathbb{R}$ and a *situation* by $\omega : B \rightarrow \mathbb{R}$. Intuitively, δ represents a partial assignment to controllable events in a PSTNU, while ω is a partial assignment to contingent events made by Nature. If δ and ω assign values to every controllable and contingent event in a PSTNU, we call them *complete* control sequence and situation. Consider now the set of control sequences $\mathcal{S} = \{\delta \mid \delta : A \subseteq \mathcal{E}_c \rightarrow \mathbb{R}\}$ and the set of situations $\mathcal{O} = \{\omega \mid \omega : B \subseteq \mathcal{E}_u \rightarrow \mathbb{R}\}$. A *scheduling policy* $\pi : \mathcal{S} \times \mathcal{O} \times \mathcal{E}_c \rightarrow \mathbb{R}$ defines a strategy to schedule controllable events based on an adopted control sequence

and the resulting situation. Due to uncertain durations, there might be a non-zero probability of a scheduling policy π violating requirement constraints in a PSTNU, giving rise to the notion of *scheduling risk* (Definition 5.11).

Definition 5.11 (Scheduling risk). *Let π be a scheduling policy for a PSTNU N , and $C \subseteq \mathcal{C}_r$. The scheduling risk for the pair $\langle \pi, C \rangle$ maps each schedule generated by π to the probability that the schedule violates one or more constraints in C .*

One should notice that Definition 5.11 considers violations of subsets of \mathcal{C}_r , the *requirement* constraints, but not \mathcal{C}_u . This is because contingent durations violating their corresponding STCU or PSTC are instances of modeling errors, a source of uncertainty outside the scope of PSTNU's.¹ For that reason, we henceforth assume that all contingent durations take place according to their modeled behavior in a PSTNU.

5.3.1 Computing strong schedules

Following [Tsamardinos, 2002, Fang et al., 2014, Wang and Williams, 2015a, Cimatti et al., 2015, Micheli et al., 2015], we focus our attention on *strong controllability* in Definition 5.12.

Definition 5.12 (Strongly controllable PSTNU). *A PSTNU N is strongly controllable (SC) if, and only if, there exists a complete control sequence δ such that, for all complete situations ω , all requirement constraints \mathcal{C}_r in N are satisfied.*

Consistent with strong controllability for STNU's and PSTN's, a PSTNU N is strongly controllable if one can compute a complete assignment to controllable events regardless of the contingent duration values during execution, and still be guaranteed to fulfill all requirement constraints in N . We shall call such a schedule a *strong policy* π_s . Vidal [Vidal, 1999] introduces rules for checking strong controllability of STNU's for the particular case where $e_1 \in \mathcal{E}_c$ in (5.1), which were later applied to PSTN's in [Fang et al., 2014]. In support of the scheduling risk discussion in Section 5.3.2

¹Considering modeling errors can be useful, should one want to model uncertainty about the uncertainty models themselves.

and our polynomial-time scheduling algorithm in Section 5.4, we now derive linear programming-based necessary and sufficient conditions for PSTNU strong controllability for the general case where e_1 in (5.1) can be controllable or contingent. For that, let $e_2 - e_1 \leq u$, $e_2 - e_1 \geq l$, represent a generic requirement constraint c_r in a PSTNU N , where e_1 and e_2 can be either controllable or contingent events. A strong policy π_s assigns values to elements of \mathcal{E}_c , therefore rendering the satisfaction of c_r a function only of the contingent durations. Let Ω be the set of all possible complete situations, and Ω_i be the subset of a complete situation affecting event e_i (directly or indirectly). From Definition 5.12, a PSTNU N is strongly controllable if, and only if,

$$\forall c_r \in \mathcal{C}_r, \max_{\Omega}(e_2 - e_1) \leq u, \min_{\Omega}(e_2 - e_1) \geq l, \quad (5.2)$$

where e_1 and e_2 are taken from c_r . To compute (5.2), use (5.1) to write

$$e_1 = e_1^c + \sum_{\Omega_1} d_i, \quad e_2 = e_2^c + \sum_{\Omega_2} d_i, \quad (5.3)$$

where $e_i^c \in \mathcal{E}_c$ is a controllable event fixed by the strong schedule, and Ω_i is the “contingent path” of e_i (for controllable e_i , $\Omega_i = \emptyset$ and $e_i = e_i^c$). In the difference $e_2 - e_1$, common contingent durations d_i cancel out, leaving

$$\begin{aligned} \max_{\Omega}(e_2 - e_1) &= e_2^c - e_1^c + \sum_{\Omega_2 \setminus \Omega_1} \max(d_i) - \sum_{\Omega_1 \setminus \Omega_2} \min(d_i), \\ \min_{\Omega}(e_2 - e_1) &= e_2^c - e_1^c + \sum_{\Omega_2 \setminus \Omega_1} \min(d_i) - \sum_{\Omega_1 \setminus \Omega_2} \max(d_i). \end{aligned} \quad (5.4)$$

5.3.2 Computing scheduling risk

For unbounded distributions such as Gaussians, or even bounded distributions on wide intervals, we would expect (5.2) to almost never hold for non-trivial requirements. However, as was previously done in the context of PSTN’s [Tsamardinos, 2002, Fang et al., 2014, Wang and Williams, 2015a], one might be able to “restore” strong controllability to a PSTNU by “squeezing” probabilistic durations, therefore

incurring non-zero amounts of scheduling risk (Definition 5.13).

Definition 5.13 (Scheduling risk for strong policies). *For a PSTNU N , let $C \subseteq \mathcal{C}_r$. Also, let $[l_i, u_i]$ be an externally-imposed bounding interval for the i -th contingent duration d_i in \mathcal{C}_u , so that (5.2) holds for every requirement constraint in C for a particular strong policy π_s . We define the scheduling risk $SR(\pi_s, C)$ of π_s with respect to C as*

$$SR(\pi_s, C) = 1 - \Pr \left(\bigwedge_{i=1}^{|\mathcal{C}_u|} d_i \in [l_i, u_i] \right). \quad (5.5)$$

The joint distribution in (5.5) can be difficult to compute, or even completely unknown. In the next section, we present a scheduling algorithm that can not only minimize a guaranteed upper bound on (5.5), but also optimize other objectives while ensuring a chance constraint $SR(\pi_s, C) \leq \theta$ for a given $\theta \in [0, 1]$.

5.4 Polynomial-time, risk-aware scheduling

We now introduce PARIS, an algorithm leveraging a *linear program* (LP) formulation to extract (or determine the nonexistence of) risk-sensitive strong scheduling policies for PSTNU's. Different from the nonlinear approaches of [Tsamardinos, 2002, Fang et al., 2014, Wang and Williams, 2015a], PARIS' linear formulation allows it to achieve not only dramatic speedups compared to the state of the art in risk-sensitive strong scheduling, but also, to the best of our knowledge, be the first scheduler guaranteed to run in polynomial time.

5.4.1 Assumptions and walk-through

PARIS assumes that all events $e \in \mathcal{E}_u$ in a PSTNU are the endpoints of exactly one element of \mathcal{C}_u . If an event $e' \in \mathcal{E}_u$ is not associated with an uncontrollable duration, we make this event the endpoint of an STCU $[0, \infty)$ starting at the time reference ($t = 0$). Also, if $e' \in \mathcal{E}_u$ is the endpoint of two distinct elements of \mathcal{C}_u ,

we deem this a modeling error. This is because contingent constraints correspond to random continuous durations, and having them share an endpoint is equivalent to forcing random durations to be consistent with each other. Loops involving contingent durations are another modeling error: contingent durations are constrained to be positive, so a loop represents the inconsistent case of an event happening before itself.

Algorithm 5.1: THE PARIS ALGORITHM.

Input: PSTNU N
Output: Strong policy π_s and scheduling risk bound Λ_{SR} .

```

1 Function PARIS( $N$ )
2    $Obj \leftarrow 0, Cts \leftarrow \emptyset$ 
3   for  $c_r \in \mathcal{C}_r$  do
4      $e_1, e_2 \leftarrow \text{StartOf}(c_r), \text{EndOf}(c_r)$ 
5      $d_1, d_2 \leftarrow \text{CtgPath}(N, e_1), \text{CtgPath}(N, e_2)$ 
6      $df_{mn}, df_{mx} \leftarrow \text{DiffMinMax}(N, d_2, d_1)$ 
7      $Cts \leftarrow Cts \cup (df_{mn} \geq \text{Lb}(c_r), df_{mx} \leq \text{Ub}(c_r))$ 
8   for  $c_u \in \mathcal{C}_u$  do
9      $Obj \leftarrow Obj + \text{SqueezeRisk}(c_u)$ 
10     $Cts \leftarrow Cts \cup \text{SqueezeCtrs}(c_u)$ 
11   $sol \leftarrow \text{Minimize } Obj \text{ s.t. } Cts$ 
12   $\pi_s \leftarrow \text{ControllableEventValues}(sol)$ 
13   $\Lambda_{SR} \leftarrow \text{Objective}(sol)$ 
14  return  $\pi_s, \Lambda_{SR}$ 

```

Algorithm 5.2: EVENT'S CONTINGENT PATH.

Input: PSTNU N , event e .
Output: List of duration variables dv .

```

1 Function CtgPath( $N, e$ )
2    $dv \leftarrow \emptyset$ 
3   while  $e \in \mathcal{E}_u$  do
4      $c_u \leftarrow \text{CtgDurationByEnd}(N, e)$ 
5     if  $c_u \in dv$  then
6        $\text{return ERROR}$ 
7     else
8        $dv, e \leftarrow dv \cup c_u, \text{StartOf}(c_u)$ 
9   return  $dv \cup e$ 

```

Algorithm 5.3: MINIMUM AND MAXIMUM IN (5.2).

Input: PSTNU N , duration variables dv_1 and dv_2 as in (5.3).
Output: Minimum and maximum in (5.2)

```

1 Function DiffMinMax( $N, dv_2, dv_1$ )
2    $mx_1 \leftarrow mx_2 \leftarrow mn_1 \leftarrow mn_2 \leftarrow 0$ 
3   for  $i = 0, 1$  do
4     for  $(c_u \in dv_{(i+1)}) \&\& (c_u \notin dv_{(2-i)})$  do
5        $mn_{(i+1)} \leftarrow mn_{(i+1)} + \text{CtgLb}(c_u)$ 
6        $mx_{(i+1)} \leftarrow mx_{(i+1)} + \text{CtgUb}(c_u)$ 
7    $df_{mn} \leftarrow \text{Last}(dv_2) - \text{Last}(dv_1) + mn_2 - mx_1$ 
8    $df_{mx} \leftarrow \text{Last}(dv_2) - \text{Last}(dv_1) + mx_2 - mn_1$ 
9   return  $df_{mn}, df_{mx}$ 

```

The pseudo-code for PARIS is shown in Algorithm 5.1. For each requirement constraint in the PSTNU, line 5 uses Algorithm 5.2 to extract the set of events involved in (5.3) (contingent loops are detected in line 5 of Algorithm 5.2). The maximums and minimums in (5.4) are computed in line 6 by Algorithm 5.3, which are used in line 7 to enforce the necessary and sufficient conditions for strong controllability in (5.2). The functions `CtgLb` and `CtgUb` in Algorithm 5.3 return, respectively, externally-imposed lower and upper bounds for contingent constraints in terms of “squeezing variables”, which are also used by function `SqueezeRisk` in Algorithm 5.1 (line 9) to compute a linear upper bound on (5.5). Line 11 of Algorithm 5.1 solves an LP to determine a risk-bounded strong scheduling policy π_s , and both π_s and an upper bound Λ_{SR} for (5.5) are returned. One should notice that Algorithm 5.2 will always return sequences ending in a controllable event (the e_i^c terms in (5.3)), which are returned by `Last` in lines 7 and 8 of Algorithm 5.3. Details on how these quantities are computed are given next.

5.4.2 A linear scheduling risk bound

Here we present the risk bound used as the objective in Algorithm 5.1 (line 9), allowing us to determine, from all available options, the strong scheduling policy π_s *minimizing this bound*. Section 5.4.5 shows how PARIS can be extended to handle other types of

linear objectives as well.

In order to obtain a linear risk objective, we apply Boole’s inequality to (5.5) and obtain the upper bound

$$\Lambda_{SR}(\pi_s, C) = \sum_{i=1}^{|\mathcal{C}_u|} \Phi_i(l_i) + (1 - \Phi_i(u_i)) \geq SR(\pi_s, C), \quad (5.6)$$

where Φ_i is the cumulative density function (cdf) associated with d_i . Notice that (5.6) holds regardless of whether contingent durations are independent or not, since it is obtained from Boole’s inequality. As it stands, (5.6) is a combination of potentially nonlinear functions. Hence, we now develop efficient linear approximations of $\Phi(l_i)$ and $1 - \Phi(u_i)$ for several common models of contingent durations.

5.4.3 The risk of “squeezing” contingent durations

The $[l_i, u_i]$ bounds in (5.6) are externally imposed on contingent durations to cause (5.2) (strong controllability) to hold. If $[l, u]$ are the true bounds of a contingent duration, we see that imposing $l_i > l$ (squeeze lower bound) causes $\Phi(l_i) > 0$ in (5.6). Analogously, choosing $u_i < u$ yields $(1 - \Phi(u_i)) > 0$. Therefore, *squeezing contingent duration bounds* causes (5.6) to grow. In the following, we show how we can quantify this risk as linear combinations of “squeezing variables” for common types of contingent durations.

Uniform durations

Let $c_u \in \mathcal{C}_u$ be a PSTC representing a random uniform duration $d \sim U(l, u)$. Also, let s_l and s_u be, respectively, the amount by which one squeezes d ’s lower and upper bounds. In this case, we have

$$\begin{aligned} \text{CtgLb}(c_u) &= l + s_l, \text{CtgUb}(c_u) = u - s_u, \\ \text{SqueezeRisk}(c_u) &= \frac{s_l}{u - l} + \frac{s_u}{u - l}, \\ \text{SqueezeCtrs}(c_u) &= s_l, s_u \in [0, u - l], \quad s_l + s_u \leq u - l. \end{aligned} \quad (5.7)$$

The terms $\Phi(l_i)$ and $(1-\Phi(u_i))$ from (5.6) correspond, respectively, to the first and second terms of $\text{SqueezeRisk}(c_u)$ in (5.7). Fortunately, these are already linear functions of the squeeze variables for uniform distributions, so no approximations are required.

Set-bounded durations

Since PSTN's were first introduced, STCU's have sometimes been treated as a particular type of PSTC, such as in [Tsamardinos, 2002]. Unfortunately, with respect to scheduling risk, the latter might not be true. In order to see this, let $c_u \in \mathcal{C}_u$ be an STCU with bounds $[l, u]$. For any nonzero amount of squeezing s_l of its lower bound, the true probability distribution of c_u could concentrate all probability mass in the interval $[l, l+s/2]$, and an analogous statement holds for the upper bound. This, in turn, would cause (5.6) to become a trivial bound $\Lambda_{SR} \geq 1$. Therefore, if (5.6) is to be a guaranteed non-trivial scheduling risk upper bound, *one must constrain STCU squeezing to be zero*, yielding the risk model

$$\begin{aligned} \text{CtgLb}(c_u) &= l, \text{CtgUb}(c_u) = u, \\ \text{SqueezeRisk}(c_u) &= 0, \text{SqueezeCtrs}(c_u) = \emptyset. \end{aligned} \tag{5.8}$$

When \mathcal{C}_u in a PSTNU only contains STCU's, (5.8) turns PARIS into a strong controllability checker for STNU's extending that of [Vidal, 1999] to the case where start events of contingent durations are not necessarily controllable.

Gaussians and other unimodal contingent durations

A key distinction between PARIS and previous methods lies in its linear handling of common *unimodal* distributions for contingent durations, Gaussians being arguably the most common example. Instead of resorting to a nonlinear solver, here we develop *piecewise-linear upper bounds* (see Figure 5-3) for the risk terms in (5.6) involving Φ . Moreover, we exploit the fact that the pdf's of such distributions are monotonic on either side of the mode to derive piecewise-linear approximations of the cdf *without*

integer variables, therefore yielding a *purely linear* formulation.

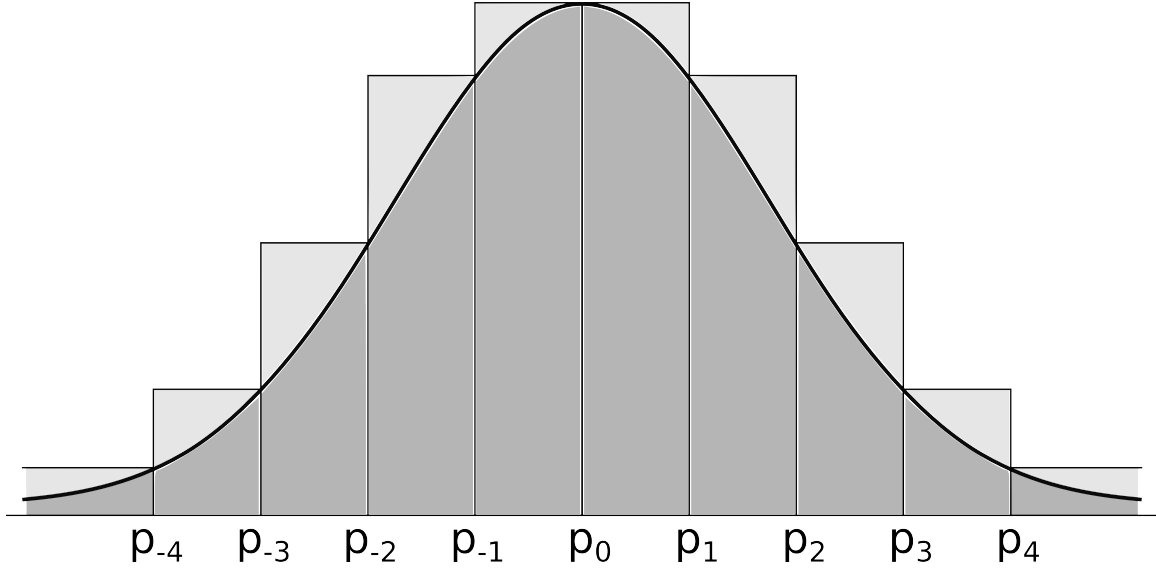


Figure 5-3: Piecewise-constant approximation of a Gaussian pdf allowing $\Phi(l_i)$ and $(1 - \Phi(u_i))$ in (5.6) to be upper bounded by a piecewise-linear function. The p_i 's are given partition points.

Let $f(x)$ be the pdf of a unimodal distribution, and let p_0 be its mode. Also, let $\mathbf{p} = \{p_{-m}, \dots, p_{-1}, p_0, p_1, \dots, p_n\}$ be a partition around p_0 with m segments to the left of the mode and n segments to the right (see Figure 5-3). For $l_i \in [p_j, p_{j+1}]$, $-m \leq j \leq -1$, one can write

$$\Phi(l_i) \leq \left(\Phi(p_{-m}) + \sum_{k=-m}^{j-1} f(p_{k+1})(p_{k+1} - p_k) \right) + f(p_{j+1})(l_i - p_j) \quad (5.9)$$

Similarly, for $u_i \in [p_{j-1}, p_j]$, $1 \leq j \leq n$, we have

$$1 - \Phi(u_i) \leq \left(1 - \Phi(p_n) + \sum_{k=j}^{n-1} f(p_k)(p_{k+1} - p_k) \right) + f(p_j)(p_j - u_i). \quad (5.10)$$

The terms within parentheses in (5.9)-(5.10) are constants, as are all p_i 's. Hence, these are, respectively, *piecewise-linear* upper bounds for $\Phi(l_i)$ and $1 - \Phi(u_i)$. Seeking to incorporate (5.9)-(5.10) into (5.6), let $s_i \in [0, p_{i+1} - p_i]$ be the amount of squeezing in the interval $[p_i, p_{i+1}]$, $-m \leq i \leq n - 1$. The squeezing risk model for unimodal

distributions is given by

$$\begin{aligned}
\text{CtgLb}(c_u) &= p_{-m} + \sum_{i=-m}^{-1} s_i, & \text{CtgUb}(c_u) &= p_n - \sum_{i=0}^{n-1} s_i, \\
\text{SqueezeRisk}(c_u) &= \Phi(p_{-m}) + (1 - \Phi(p_n)) \\
&+ \sum_{i=-m}^{-1} f(p_{i+1})s_i + \sum_{j=0}^{n-1} f(p_j)s_j, \\
\text{SqueezeCtrs}(c_u) &= s_i \in [0, p_{i+1} - p_i], \quad -m \leq i \leq n-1,
\end{aligned} \tag{5.11}$$

The s_i in (5.11) must consistently implement the piecewise-linear behavior from (5.9)-(5.10), i.e., for $i < 0$, one must have $s_{i-1} < p_i - p_{i-1} \Rightarrow s_i = 0$; and, for $i > 0$, $s_i < p_{i+1} - p_i \Rightarrow s_{i-1} = 0$. In general, these rules would have to be enforced through binary variables representing the “activation” of piecewise-linear segments. However, since I) s_i ’s in (5.11) only affect duration bounds through their sum; II) the coefficients of the s_i in **SqueezeRisk** are *monotonic* on either side of the mode; and III) PARIS minimizes (5.6), an optimal solution found by Algorithm 5.1 must necessarily fulfill the aforementioned rules, therefore correctly “squeezing” the distribution. Hence, *no binary variables are needed!*

5.4.4 Improving piecewise approximations

The previous section does not discuss how the partition \mathbf{p} should be chosen. In principle, the specific p_i ’s should not matter, as long as the “box” between p_{i-1} and p_i is chosen so that it always overestimates the pdf (and, thus, the area under the curve). However, Figure 5-3 shows that these boxes can yield a rather crude approximation of the cdf, depending on where partition points are placed. Therefore, given a fixed number of partitions, we now focus on choosing \mathbf{p} so that it approximates the cdf well. For that, let

$$g(\mathbf{p}) = \sum_{i=-m}^{-1} (p_{i+1} - p_i) f(p_{i+1}) + \sum_{j=0}^{n-1} (p_{j+1} - p_j) f(p_j) \tag{5.12}$$

be the total area of the piecewise approximation. Since (5.12) is an upper bound, one might seek to

$$\underset{\mathbf{p}}{\text{minimize}} g(\mathbf{p}) \text{ subject to } p_{i+1} \geq p_i, \forall i, p_{-m}, p_0, p_n \text{ fixed.} \quad (5.13)$$

The objective in (5.13) is not linear or convex in general (e.g., it is neither for Gaussians). Due to the inequalities, one could resort to SUMT [Fiacco and McCormick, 1964], a.k.a. *barrier methods*, to solve a sequence of unconstrained minimization problems that provide increasingly better estimates of a local solution to (5.13). However, these intermediate steps are costly and, as pointed out in [Boyd and Vandenberghe, 2004], computing them exactly is not necessary. Therefore, seeking to keep computational requirements manageable, we resort to a simpler approach based on gradient descent. Starting with $\mathbf{p}^0 = \mathbf{p}_0$, we compute

$$\mathbf{p}^{t+1} = Q \left(\mathbf{p}^t - \mu \nabla g \Big|_{\mathbf{p}=\mathbf{p}^t} \right) \quad (5.14)$$

until $\|\nabla g(\mathbf{p}^t)\| \approx 0$ or a maximum number of iterations is reached. In (5.14), the components of ∇g are

$$\begin{aligned} \frac{\partial g}{\partial p_k} &= f'(p_k)(p_k - p_{k-1}) + f(p_k) - f(p_{k+1}), \quad -m < k < 0, \\ \frac{\partial g}{\partial p_k} &= f'(p_k)(p_{k+1} - p_k) + f(p_{k-1}) - f(p_k), \quad 0 \leq k < n, \end{aligned} \quad (5.15)$$

and 0 for $k \in \{-m, 0, n\}$; μ is a positive constant; and $Q(\cdot)$ is a projection ensuring (5.14) remains in the feasible region of (5.13). In our implementation, $Q(\cdot)$ is the identity if its argument is feasible, and otherwise outputs a random feasible perturbation around \mathbf{p}^t . Thus, at any t , (5.14) is feasible and, for small enough μ , will improve the upper bound (5.12) at every iteration. For a Gaussian $N(\mu, \sigma^2)$, we have

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \quad f'(x) = f(x) \left(\frac{\mu-x}{\sigma^2} \right). \quad (5.16)$$

5.4.5 From minimum risk to other linear objectives

The description of PARIS in Algorithm 5.1 computes π_s minimizing the scheduling risk bound (5.6). However, as pointed out in [Fang et al., 2014, Wang and Williams, 2015a], there might be situations where other types of linear objectives (e.g., the schedule’s makespan) might be preferred over minimizing risk. Thus, let

$$\underset{\mathbf{x}}{\text{minimize}} \ h(\mathbf{x}) \quad \text{subject to} \ Cts(\mathbf{x}) \tag{5.17}$$

be the *target problem*, where $h(\mathbf{x})$ is the *desired objective* (e.g., the makespan) for PARIS; \mathbf{x} is the vector of problem variables (schedule of controllable events, squeezing variables, etc.); and Cts are the same as in line 11 of Algorithm 5.1, plus any additional (linear) constraints required by $h(\mathbf{x})$. In this section, we analyze two risk-motivated settings for solving (5.17): I) a *chance-constrained* setting, in which we solve (5.17) without caring for the actual value of the scheduling risk $SR(\mathbf{x})$, as long as one can ensure that it is bounded by a user-specified tolerable risk level θ ; and II) a *tight risk gap* setting, in which we would like to solve (5.17) while ensuring that our estimate of $SR(\mathbf{x})$ is as good as possible.

Chance-constrained

In the *chance-constrained* setting, (5.17) must be solved while ensuring $SR(\mathbf{x}) \leq \theta$ for a given $\theta \in [0, 1]$. Following Section 5.4.2, we achieve $SR(\mathbf{x}) \leq \theta$ through the sufficient condition

$$\Lambda_{SR}(\mathbf{x}) \leq \theta, \tag{5.18}$$

which should be added as an element of $Cts(\mathbf{x})$ in (5.17). It is worthwhile to notice that (5.18) is sufficient to enforce $SR(\mathbf{x}) \leq \theta$ *even in the absence of binary values* to ensure that the “squeezings” in (5.11) are performed correctly. The reasons are the same as presented in Section 5.4.3: squeeze variables have monotonic coefficients and can only affect bounds through their sums, so a bound $\Lambda'_{SR}(\mathbf{x})$ resulting from “potentially incorrectly squeezed” distributions can only overestimate the bound $\Lambda_{SR}(\mathbf{x})$

generated by the same amount of squeezing, but performed in the correct order. Therefore, since $\Lambda'_{SR}(\mathbf{x}) \leq \theta$ is sufficient for (5.18), a solution of (5.17) is guaranteed to be chance constrained *no matter how the squeezings are performed*.

Tight risk gap

Assume now that, in addition to solving (5.17), we require that the scheduling risk bound given by (5.6) is tight. This is useful, for instance, when no knowledge about the joint distribution of durations is available, so that (5.5) cannot be computed even if all bounds $[l_i, u_i]$ for the contingent durations are given. In this case, (5.6) becomes our best estimate of the scheduling risk for the mission, and we would like to make it as good as possible.

A natural way of achieving this goal would be to *introduce binary variables* in (5.17), so that the piecewise-linear approximations in Section 5.4.3 are correctly implemented. However, that would turn (5.17) into a significantly harder problem to solve, since it would become a Mixed-Integer LP (MILP). Therefore, we propose to replace (5.17) by the surrogate

$$\underset{\mathbf{x}}{\text{minimize}} \Lambda_{SR}(\mathbf{x}) + Mh(\mathbf{x}) \quad \text{subject to } Cts(\mathbf{x}), \quad (5.19)$$

where $\Lambda_{SR}(\mathbf{x})$ is the risk bound (5.6) and M is a finite positive constant. Unlike (5.17), (5.19) *does not require binary variables* due to the introduction of $\Lambda_{SR}(\mathbf{x})$ in the objective. On the other hand, (5.19) no longer optimizes our desired objective $h(\mathbf{x})$. Despite the latter, we now show that M can be chosen so that a solution to (5.19) approximates that of (5.17) with arbitrary precision while requiring no integer variables.

Let \mathbf{x}^* be the optimal solution found by (5.17), and let $h^* = h(\mathbf{x}^*)$ be the corresponding minimal value of $h(\cdot)$ over the convex region defined by $Cts(\mathbf{x})$. Also, let \mathbf{x}' be the solution found by (5.19) over the same convex region, with corresponding desired objective $h(\mathbf{x}') = h^* + \Delta h$. The Δh term is the *objective degradation*, and must be such that $\Delta h \geq 0$, given that h^* is the minimum of $h(\cdot)$ over $Cts(\mathbf{x})$. Any

such degradation in $h(\mathbf{x})$ must be a result of a corresponding decrease $\Delta\Lambda_{SR}(\mathbf{x})$ of the risk bound, and the minimality of (5.19) implies $\Delta\Lambda_{SR}(\mathbf{x}) + M\Delta h \leq 0$. The last step is to notice that the risk bound improvement is such that $-\Delta\Lambda_{SR}(\mathbf{x}) \leq |\mathcal{C}_u|$, which is the difference between the maximum ($\Lambda_{SR} = |\mathcal{C}_u|$) and minimum ($\Lambda_{SR} = 0$) values of Λ_{SR} in (5.6). Therefore, we arrive at the degradation bound

$$\Delta h \leq \frac{|\mathcal{C}_u|}{M}. \quad (5.20)$$

For instance, if $h(\mathbf{x})$ is the schedule's makespan measured in seconds and one chooses $M = 1000|\mathcal{C}_u|$, (5.20) guarantees that the *fully linear* surrogate (5.19) approximates the true minimal makespan h^* with millisecond precision. Also, nothing prevents us from imposing (5.18) on (5.19) to enforce $SR(\mathbf{x}) \leq \theta$. In Section 5.5, this will be referred to as the *tight, chance-constrained* (TCC) setting.

5.4.6 Algorithm properties

This section presents soundness, completeness, and complexity properties for the different formulations of PARIS.

Lemma 5.5. *PARIS is sound.*

Proof: PARIS enforces (5.4), which are necessary and sufficient for strong controllability. Therefore, any schedule found must be a strong scheduling policy. \square

Lemma 5.6. *PARIS runs in polynomial time.*

Proof: the loops in lines 3 and 8 of Algorithm 5.1 run a polynomial number of iterations, and both Algorithms 5.2 and 5.3 run in polynomial time. Also, the scheduling risk models in (5.7), (5.8), and (5.11) create a polynomially-large number of variables and constraints relative to the number of contingent durations, and Karmarkar [Karmarkar, 1984] showed that the LP in line 11 of Algorithm 5.1 can be solved in polynomial time. Finally, the cap on the number of iterations for the partition optimization step (5.14) introduces a maximum overhead that grows linearly with the number of contingent durations. \square

Lemma 5.7. *PARIS is complete when contingent durations are restricted to STCU’s or uniform PSTC’s.*

Proof: the squeezing models (5.7) and (5.8) consider the whole range of possible externally-imposed upper and lower bounds for contingent durations. Therefore, if there are squeezings for which a strong policy exists, PARIS will find it. Otherwise, it will return no solution. \square

Lemma 5.8. *The squeezing model (5.11) for unimodal PSTC’s renders PARIS incomplete.*

Proof: the model (5.11) will fail to return strong schedules requiring upper and lower bounds to be squeezed beyond the extrema of the interval, or not containing the mode. \square

Lemma 5.9. *Chance-constrained PARIS is sound, but incomplete.*

Proof: Soundness of (5.18) follows from $SR(\mathbf{x}) \leq \Lambda_{SR}(\mathbf{x}) \leq \theta$. Incompleteness follows from the fact that enforcing (5.18) to ensure $SR(\mathbf{x}) \leq \theta$ may discard valid solutions when θ is placed in the gap between the minimum of $\Lambda_{SR}(\mathbf{x})$ and the true value of $SR(\mathbf{x})$. \square

5.5 Experiments

The *CAR-SHARING* and *ROVERS* datasets available at

<http://mers.csail.mit.edu/datasets/scheduling>,

along with several other examples, were used in the empirical evaluation of PARIS.

The CAR-SHARING dataset, first made available in [Fang et al., 2014] and later used in [Wang and Williams, 2015a], served the purpose of evaluating PARIS against the state of the art in PSTN scheduling. Experiments from [Wang and Williams, 2015a] on this dataset show that *Rubato*’s conflict-directed, chance-constrained strong controllability checker for PSTN’s outperforms the approach in [Tsamardinos, 2002]

and the *Picard* system presented in [Fang et al., 2014] by nearly an order of magnitude. Therefore, here we compare Rubato’s performance relative to that of PARIS on CAR-SHARING. Rubato is implemented in Common Lisp and uses Ipopt [Wächter and Biegler, 2006] as the nonlinear solver, while PARIS is implemented in Python and uses Gurobi 6.0.4.

The ROVERS dataset consists of 4380 randomly-generated PSTNU instances modeling planetary rover coordination scenarios similar to the one depicted in Figure 5-1a. ROVERS instances feature the coordination between two to ten rovers, which have to complete between one to ten sequential exploration tasks (*drive*, *drill*, *collect*, and *process*) in parallel before reconvening at a relay location and transmitting their data to a satellite within a given visibility window. ROVERS contains all combinations of uncontrollable temporal durations discussed in this chapter and, to the best of the author’s knowledge, cannot be handled by existing scheduling algorithms.

Our first test consisted of running the risk-bound minimization version of PARIS explained in Algorithm 5.1 against Rubato on CAR-SHARING, with results shown in Figures 5-4a and 5-4b (vertical axes in log scale). Out of 1800 instances, PARIS found strong policies for 186 in Figure 5-4a (same number as in [Fang et al., 2014]), while Rubato found 142. Unsurprisingly, we see that strong schedules tend to exist for networks with fewer uncontrollable durations. PARIS handled Gaussian durations using 8 equally-spaced partitions (length equal to σ) on either side of the mean. Notice that Figures 5-4a and 5-4b shows PARIS outperforming Rubato by about 1 or 2 orders of magnitude in most test instances, and up to three or four orders in some of the most difficult problems. Also, while PARIS computes the strong policy, Rubato solves the strictly easier problem of checking if one such policy exists.

For the 186 instances for which PARIS found a strong policy, our next test on CAR-SHARING evaluated how the partition optimization procedure from Section 5.4.4 impacts the gap between the risk bound in (5.6) and the true scheduling risk in (5.5). Since (5.5) is hard to compute in general, we assume, for this particular experiment only, that all contingent durations were independent, in which case (5.5)

becomes

$$SR(\pi_s, C) = 1 - \prod_{i=1}^{|\mathcal{C}_u|} \Pr(d_i \in [l_i, u_i]). \quad (5.21)$$

Gradient descent was allowed a maximum of 12,000 iterations with gradient norm tolerance 10^{-3} and fixed $\mu = 0.03$. Figure 5-4d shows the elapsed time as a function of the number of uncontrollable duration, while Figure 5-4c shows the risk gap between (5.6) and (5.21) for different instances with and without partition optimization. Partition optimization caused the risk gap to improve on all instances, with an average gap improvement of 4.8% (absolute value, not relative). Also, the linear trend in Figure 5-4d confirms that partition optimization *does not affect* PARIS’ polynomial-time complexity.

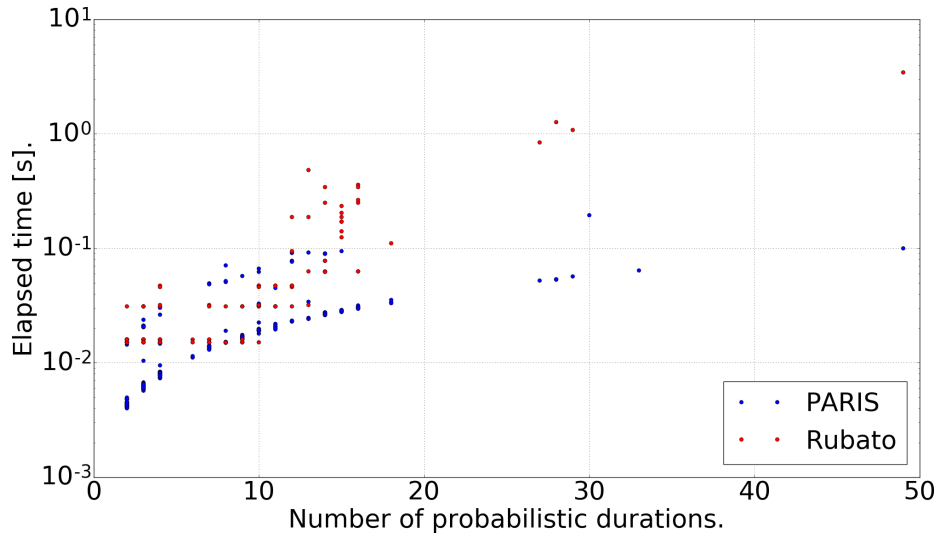
Our last test on CAR-SHARING evaluated the effectiveness of PARIS in optimizing general linear objectives, as explained in Section 5.4.5. For this test, the *schedule makespan* was used as the desired objective in (5.19), with a chance constraint $\theta=30\%$ imposed through (5.18). It was compared against the risk-bound minimization version in Algorithm 5.1, and the results are shown in Figures 5-4e and 5-4f. As expected, using (5.19) with makespan as the desired objective improved the makespan for all test instances, with an average reduction of 8.5 seconds. Also, Figure 5-4f shows that both formulations have similar runtimes.

On the ROVERS dataset, from a total of 4380 PSTNU instances, PARIS found strong policies for 2840 of them. From those, 911 had probabilistic durations squeezed to a single value, i.e., even though a strong policy exists, it is almost guaranteed to fail. Different from CAR-SHARING, Figures 5-5a and 5-5b show that there are instances of ROVERS featuring both strong policies and a large number of uncontrollable durations. Moreover, we observe from these figures the small amount of time required by PARIS to solve instances with and without strong policies, reinforcing the claim that PARIS is suitable for hardware with computational and energy constraints. Finally, the same conclusions from CAR-SHARING regarding partition optimization and makespan optimization hold on ROVERS. For instance, as shown in Figure 5-5c, makespan optimization with a chance constraint $\theta=20\%$ improved the makespan for

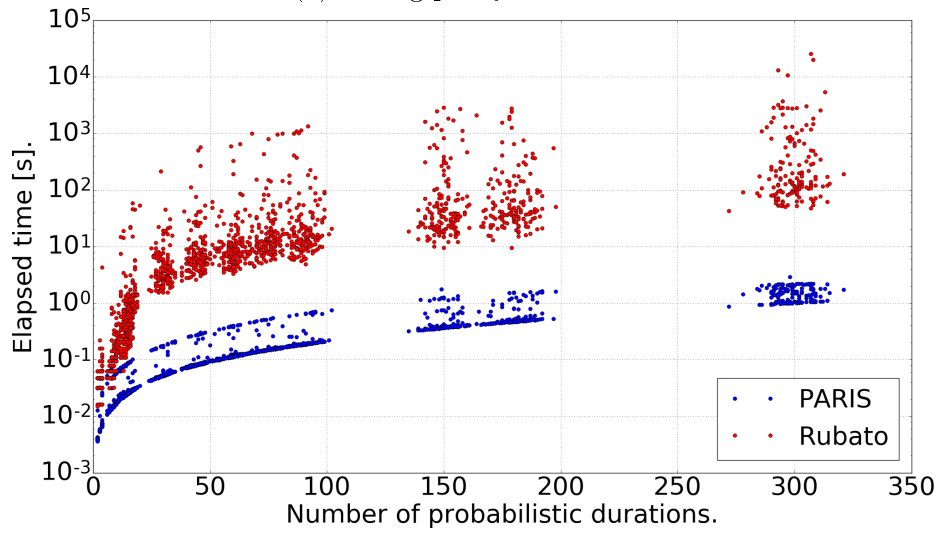
all problem instances, with an average reduction of 37.36 seconds.

5.6 Conclusions

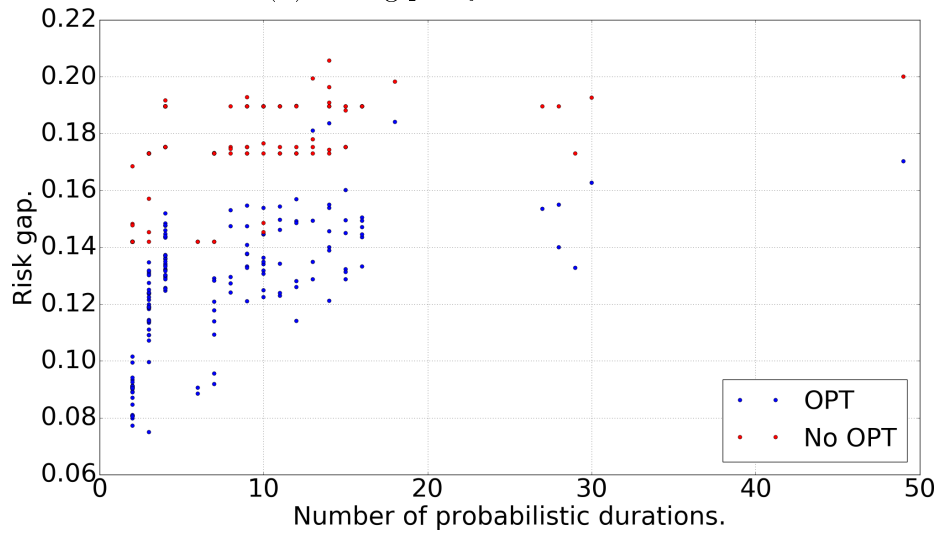
This chapter introduces PSTNU's, a formalism subsuming STNU's and PSTN's, and PARIS, a polynomial-time and sound algorithm for risk-sensitive strong scheduling of PSTNU's that outperforms the current fastest equivalent algorithm for PSTN's by several orders of magnitude. Due to its significantly reduced computational requirements, PARIS endows CLARK with a keen sensitivity to scheduling risk for temporal plans with duration uncertainty. However, PARIS is limited to *unconditional* scheduling problems, i.e., those for which the constraints that compose the temporal network do not depend on real-time observations. Since this thesis' goal is to generate risk-bounded *conditional* temporal plans, the next chapter extends PARIS to strong scheduling of Probabilistic Temporal Plan Networks (PTPN's) containing probabilistic observations.



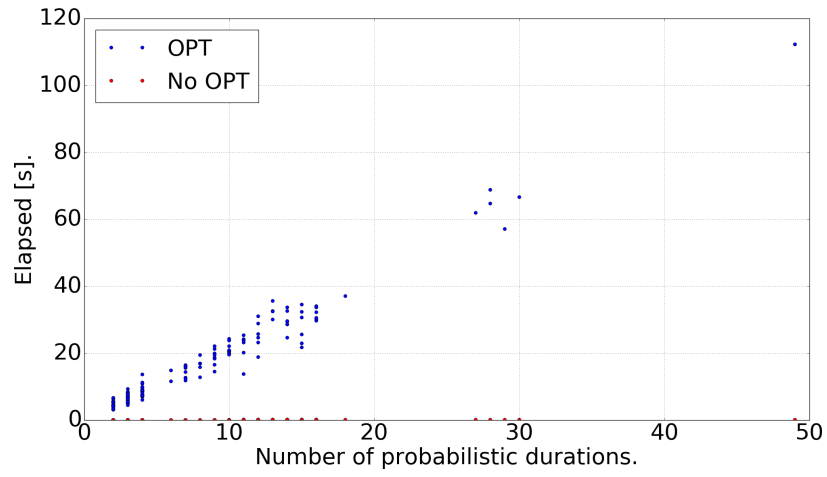
(a) Strong policy was found.



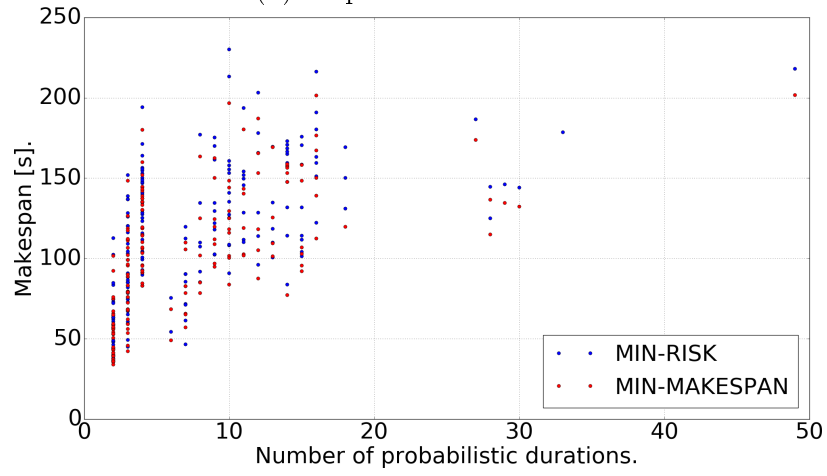
(b) Strong policy was not found.



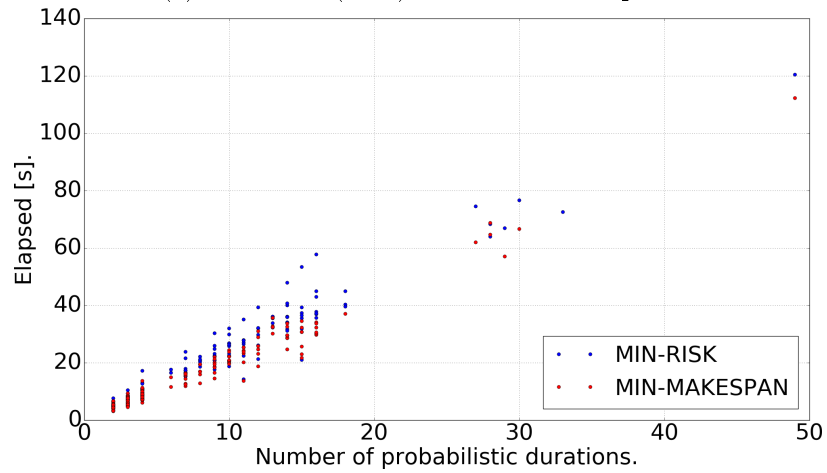
(c) Risk gap with (blue) and without (red) partition optimization.



(d) Elapsed time for 5-4c.

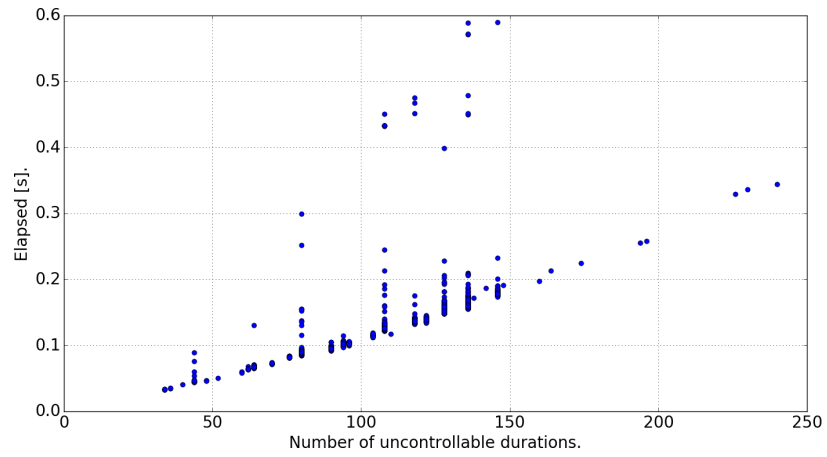


(e) Min. risk (blue) and min. makespan.

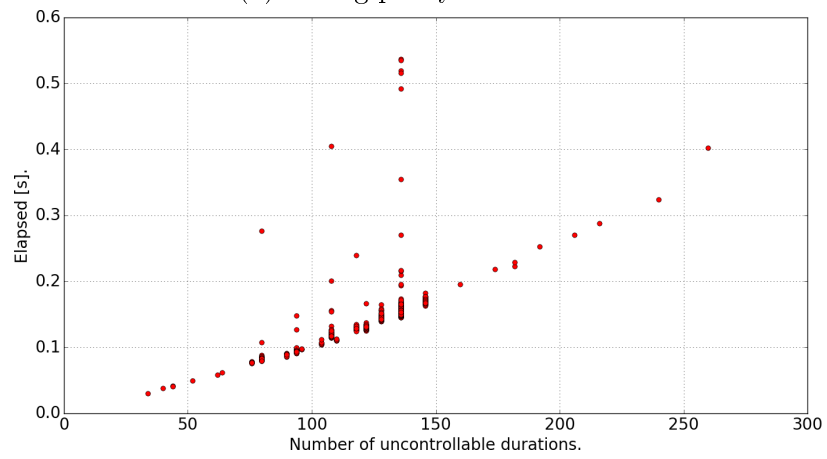


(f) Elapsed time for 5-4e.

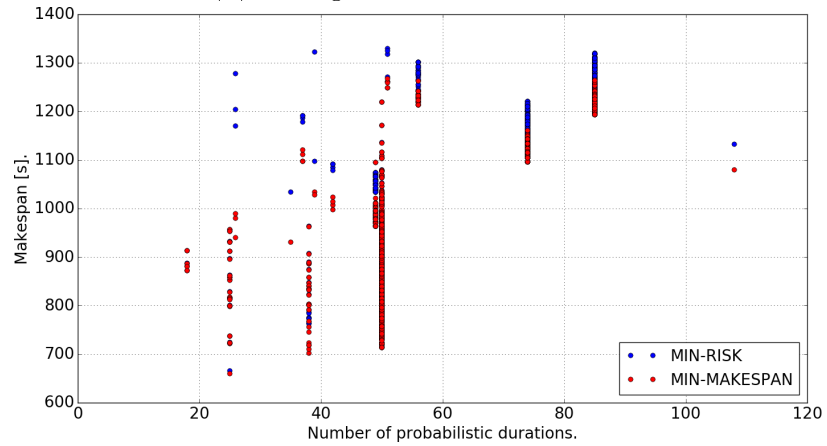
Figure 5-4: Performance of Rubato and PARIS on CAR-SHARING dataset.



(a) Strong policy was found.



(b) Strong policy was not found.



(c) Min. risk (blue) and min. makespan.

Figure 5-5: Performance of PARIS on ROVERS dataset.

Chapter 6

Risk-sensitive scheduling of PTPN's

“There cannot be a crisis next week. My schedule is already full.”

Henry A. Kissinger.

The PARIS algorithm in Chapter 5 allows a PSTNU, a temporal network with a mix of set-bounded and various types of probabilistic temporal durations, to be efficiently tested for the existence of a strong schedule. However, PSTNU's can only be used for *unconditional* scheduling problems, i.e., situations in which all temporal constraints affecting a schedule are known beforehand and with complete certainty. Since this thesis is concerned with risk-bounded *conditional* temporal plans, Chapter 4 introduces Probabilistic Temporal Plan Networks (PTPN's), a graph-based representation of a conditional temporal network that can extend PSTNU's by allowing temporal constraints to be conditional on controllable (agent decisions) and uncontrollable (real-time observations) choice variables.

When CLARK is given a temporal CC-POMDP model or a cRMPL program to execute, the temporal constraints in RAO*'s policy estimate will often - unless the planning problem is unconditional - form a particular type of PTPN where there are only probabilistic uncontrollable choices representing real-time observations. In order to verify the temporal feasibility of this particular type of PTPN, this chapter extends the notions of strong scheduling for PSTNU's from Chapter 5 to *strong consistency* of PTPN's featuring probabilistic observations and uncontrollable temporal durations.

6.1 Introduction

There has been significant effort in the scientific community to help answer the question of whether a feasible schedule exists in the presence of uncontrollable contingencies in a temporal network. Among the most important contributions, [Tsamardinos et al., 2003, Effinger et al., 2009, Venable et al., 2010, Hunsberger et al., 2012, Combi et al., 2013, Cimatti et al., 2014, Cimatti et al., 2016b, Cimatti et al., 2016a] extend the notions of strong, dynamic, and weak controllability (also referred as consistency) originally used in the context of temporal uncertainty [Vidal and Ghallab, 1996, Vidal, 1999, Morris et al., 2001, Morris and Muscettola, 2005, Morris, 2006, Hunsberger, 2009, Hunsberger, 2010, Hunsberger, 2013, Hunsberger, 2014, Morris, 2014] to uncontrollable contingencies. They also present tests that guarantee the existence of feasible solutions under different levels of information about the uncertainty in the plan. These consistency tests, however, all have the caveat of representing uncertainty as set-bounded quantities, i.e., as intervals of values with no associated probability distribution. In order to guarantee feasibility in all possible scenarios, consistency-checking algorithms based on set-bounded uncertainty end up performing a worst-case analysis. When considering situations where uncertainty causes small plan deviations around otherwise “nominal” values, these set-bounded consistency criteria work well and output robust, albeit conservative, schedules. Nevertheless, they have difficulties handling problem instances where uncertainty can potentially lead to infeasible scenarios, often returning that no robust scheduling policy exists. This is most certainly undesirable, since reasonable amounts of risk can usually be tolerated for the sake of not having the autonomous agent sit idly due to its absolute “fear” of the worst.

Given a description of a contingent temporal plan in the form of a PTPN, this chapter improves upon the previously mentioned results on conditional scheduling by extending the notions of weak and strong plan consistency to a risk-bounded setting, and providing efficient algorithms for determining (or refuting) them. Weak and strong consistency are useful concepts when planning missions for agents whose embedded hardware has very limited computation and telecommunication power,

making it hard for them to come up with solutions “on the fly” or for remote operators to intervene in a timely fashion. Chance-constrained weak consistency (CCWC) is a useful concept for missions where agents operate in static or slow changing environments after an initial scouting mission aimed at reducing plan uncertainty. Chance-constrained strong consistency (CCSC), on the other hand, removes the need for a scouting mission and tries to determine the existence of a solution that, with probability greater than some threshold, will succeed irrespective of the outcomes of uncertainty in the plan. Strong consistency is clearly more conservative, but it is appealing to mission managers because strongly consistent policies require little to no onboard sensing and decision making, greatly reducing the agents’ complexity and costs. They also reduce or completely eliminate the need to coordinate between multiple agents. Finally, the robustness of a strongly consistent policy makes it easier to check by human operators before it is approved for upload to the remote agent.

The PTPN representation of contingent temporal plans from Chapter 4 is similar to Temporal Plan Networks with Uncertainty (TPNU’s) [Effinger et al., 2009, Effinger, 2012], Conditional Temporal Plans (CTPs) [Tsamardinos et al., 2003], Disjunctive Temporal Problems with Uncertainty (DTPU’s) [Venable et al., 2010], and the Conditional Simple Temporal Network with Uncertainty (CSTNU) [Hunsberger et al., 2012], but extends them in two important ways. First, PTPN’s support probabilistic choice nodes, as opposed to a purely set-bounded uncertainty representation in DTPU’s and CTP’s. Second, PTPN’s can contain any combination of PSTNU-like temporal constraints, i.e., set-bounded, as well as different types of probabilistic temporal durations. In this thesis, PTPN’s arise both as representations of cRMPL execution traces, and as the conditional temporal constraints in temporal CC-POMDP models.

The algorithms in this chapter reason quantitatively about the probability of different random scenarios and explore the space of feasible solutions efficiently while bounding the risk of failure of a conditional schedule below a user-defined admissible threshold. While state-of-the-art methods in the conditional and stochastic CSP literature rely on a combination of chronological (depth-first) search and inference in

the space of contingencies in order to quickly find satisficing solutions [Fargier et al., 1995, Fargier et al., 1996, Stergiou and Koubarakis, 2000, Gelle and Sabin, 2006, Tarim et al., 2006], in this chapter we introduce a “diagnostic” approach based on Conflict-Directed A^* (CDA*) [Williams and Ragno, 2007]. By continuously learning subsets of conflicting constraints and generalizing them to a potentially much larger set of pathological scenarios, our algorithms can effectively explore the space of robust schedules in best-first order of risk while ensuring that it is within the user-specified bound. For the problem of extracting a strongly consistent policy from a contingent plan description, our numerical results showed significant gains in scalability for our approach.

This chapter is organized as follows. Section 6.2 revisits the simple PTPN scheduling example presented in Section 1.4 to motivate our methods, followed by a formal definition of the notions of chance-constrained consistency in Section 6.3. Next, Section 6.4 presents algorithms for determining chance-constrained weak and strong consistency of PTPN’s. The numerical results in Section 6.5 indicate that our framing of the problem of determining CCSC outperforms the current practice of using chronological search, followed by our final conclusions in Section 6.6.

6.2 Approach in a nutshell

Here we motivate the usefulness of chance-constrained consistency by revisiting the simple commute example from Section 1.4, whose PTPN is shown in Figure 6-1. In this example, we start at home and our goal is to be at work for a meeting in at most 30 minutes. Circles represent temporal events, and temporal constraints are represented by arcs connecting temporal events. For simplicity, we assume that we are given only three possible choices in this PTPN: we can either ride a bike to work, drive a car, or stay home and telecommute. The rewards (R values) model preferences associated with each one of the options. Uncontrollable choices are depicted in Figure 6-1 by double circles with dashed lines. These are random, discrete events that affect our plan and whose probability model is also given in Figure 6-1.

In this example, the uncontrollable choices model what might “go wrong” during

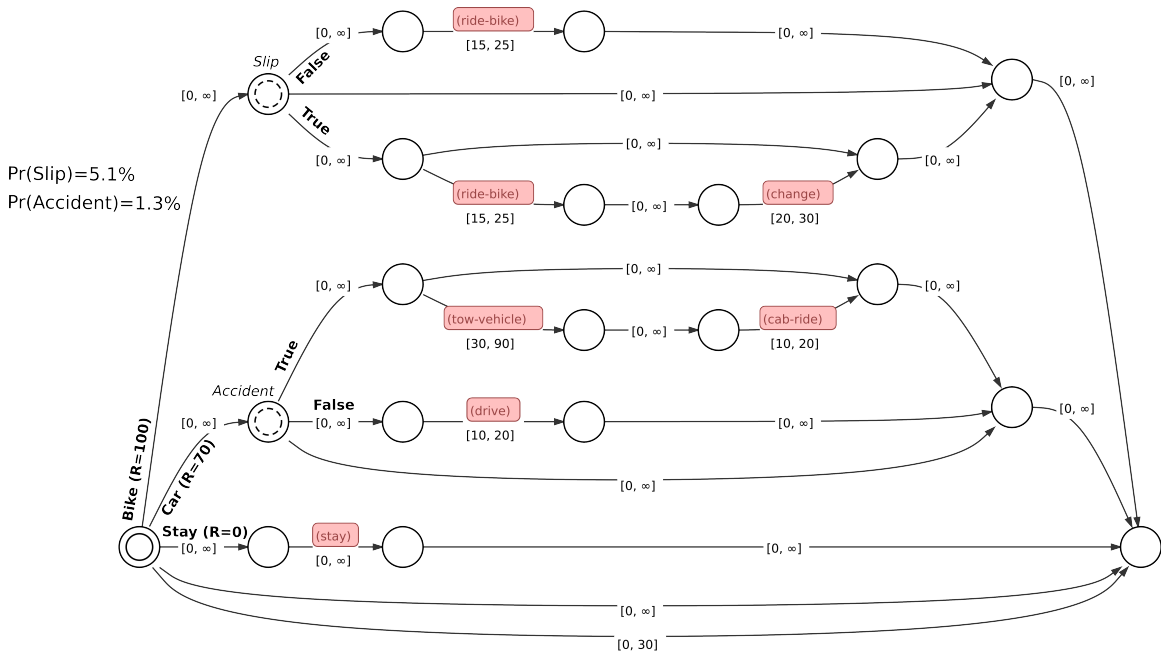


Figure 6-1: A PTPN for a simple plan to get to work (repeated from Figure 1-12 on page 43).

plan execution and the impact of these unexpected events on the overall duration of the plan. For example, if we decide to ride a bike to work (the most preferred option), there is the possibility that we might slip and fall. This event has a minor effect on the duration of the ride, but would force us to change clothes at our workplace because we cannot spend the day in a dirty suit. Since we only have 30 minutes before the meeting starts, the uncontrollable event of slipping would cause the overall plan to be infeasible. A similar situation happens if we choose to drive our car and happen to be involved in an accident.

By ignoring probabilities and using a consistency checker for the PTPN in Figure 6-1 based on a set-bounded representation of uncertainty, we would realize that the PTPN is guaranteed to be consistent. Unfortunately, unless we had a way of telling ahead of time whether we would slip from the bike or be in a car accident, the plan would be to always choose the least preferred option of staying at home. This is because, for the choice of riding a bike or driving to work, there are uncontrollable scenarios that cause the plan to fail, causing the set-bounded consistency checker

to fall back to the safe, albeit undesirable, choice of staying at home. Since this may seem to disagree with common sense, since people try to achieve their goals while acknowledging the uncontrollable nature of their environment, in the following we show how a chance-constrained approach would produce a plan that agrees with what we would expect a “reasonable” agent to do.

Let us consider the case where we accept that our plan might fail, as long as the risk Δ is no more than 2%. Given that riding a bike is the option with the highest reward, our algorithm would deem bike riding the most promising and would start by checking if choosing to ride a bike meets the chance constraint $\Delta \leq 2\%$. If there existed a feasible activity schedule satisfying the temporal constraints for both values of *Slip*, we could pick this schedule and our risk of failure would be zero, which is clearly less than our risk bound. However, our algorithm concludes that the scenario *Slip* = *True* is inconsistent with the overall temporal constraint of arriving at the meeting in less than 30 minutes, so there must exist a nonzero risk of failure in this case. According to the model in Figure 6-1, the probability of slipping is $\Pr(\textit{Slip}) = 5.1\%$, so riding a bike does not meet the chance constraint $\Delta \leq 2\%$. The next best option is riding a car, where now we are subject to the uncontrollable event of being in a car accident. Following a similar analysis, we conclude that the risk of our plan being infeasible in this case is $\Pr(\textit{Accident}) = 1.3\%$, which meets the chance constraint. Therefore, our algorithm would advise to drive to work within the temporal bounds shown in Figure 6-1 for the case where no accident happens. It is worthwhile to notice that choosing $\Delta < 1.3$ would have made staying at home the only feasible alternative. Hence, as in the set bounded approach, a chance constraint may still be too conservative to allow for a feasible solution. Moreover, if the overall temporal constraint of 30 minutes in Figure 6-1 were relaxed to 35 minutes, our algorithm would have been capable of finding a risk free scheduling policy for its first choice of riding a bike. This is because, in this case, there would exist a feasible schedule satisfying all temporal constraints on the upper side of the PTPN, i.e., temporal constraints activated by both *Slip* = *True* and *Slip* = *False*.

Within CLARK, the conditional temporal constraints generated by RAO*’s policy

estimates form a PTPN containing only probabilistic uncontrollable choices, which must be tested for chance-constrained temporal consistency. Since testing feasibility of a chance constraint is a fundamental task, in which estimating risk is costly, our approach frames risk estimation for a PTPN as a process of enumerating the most likely sets of scenarios that incur and do not incur risk (called conflicts and kernels, respectively). We observe that this can be formulated as a symptom-directed, “diagnostic” process, allowing us to leverage an efficient, conflict-directed best-first enumeration algorithm, Conflict-Directed A^* (CDA*) [Williams and Ragno, 2007], to generate kernels and conflicts, and hence feasible and infeasible scenarios.

6.3 Problem statement

This section extends the “risk-averse” notions of weak and strong consistency from [Vidal and Ghallab, 1996, Tsamardinos et al., 2003, Effinger et al., 2009] to a setting where solutions involving some level of risk are accepted. As previously mentioned, allowing plans to contain reasonable levels of risk is usually required for systems operating under uncertainty. If no risk is allowed, a robust scheduling strategy will hardly ever exist.

The combination of assignments to controllable and uncontrollable choices in a PTPN defines “paths” that go through temporal constraints that must be satisfied by the schedule. These paths are encoded in the PTPN by the activation of *labels*, also referred to as *guard conditions*, of PTPN elements. This motivates our definition of controllable and uncontrollable scenarios.

Definition 6.14 (Controllable (Uncontrollable) scenario). *A controllable (uncontrollable) scenario $CS \in \mathbb{CS}$ ($US \in \mathbb{US}$) corresponds to a full assignment $D = d$ ($U = u$) to the controllable (uncontrollable) choices in the PTPN.*

Intuitively, a chance-constrained weakly consistent PTPN is one that, for every possible uncontrollable scenario within a set \mathbb{US}' , there exists at least one controllable scenario CS so that a feasible schedule exists. Therefore, given knowledge about

which uncontrollable scenario is true, we can always choose CS such that a schedule exists. In our chance-constrained formulation, we guarantee that the risk of the true uncontrollable scenario being outside of the feasible set \mathbb{US}' is less or equal to Δ . Strong consistency, on the other hand, requires that at least one controllable scenario CS exists such that there exists at least one common schedule satisfying the temporal constraints generated by all uncontrollable scenarios $US \in \mathbb{US}'$. Once again, our chance-constrained formulation guarantees that our strongly consistent schedule will be feasible with probability at least $1-\Delta$. While weak consistency requires complete prior knowledge of the true uncontrollable scenario prior to choosing CS , strong consistency requires none: we could just “close our eyes” and pick a schedule while completely disregarding the uncontrollable scenario being unfolded.

Let $TC \leftarrow Ac(PTPN, CS, US)$ be a function that returns all active (i.e., with labels evaluating to True) temporal constraints in a PTPN given controllable and uncontrollable scenarios CS and US . Also, let $sched \leftarrow Sol(TC)$ be an algorithm (such as PARIS in Chapter 5) that is able to test for the existence of a schedule $sched$ for TC , where the absence of a schedule is denoted by $sched = \emptyset$. We are now ready to define the notions of chance-constrained weak and strong consistency.

Definition 6.15 (Chance-constrained weak consistency (CCWC)). *A PTPN is said to be CCWC with risk Δ iff there exists a set of uncontrollable scenarios $\mathbb{US}' \subseteq \mathbb{US}$, where $\Pr(\mathbb{US}') > 1-\Delta$, such that $\forall US \in \mathbb{US}'$, there exists a controllable scenario CS such that $Sol(Ac(PTPN, CS, US)) \neq \emptyset$.*

Definition 6.16 (Chance-constrained strong consistency (CCSC)). *A PTPN is said to be CCSC with risk Δ iff there exists a controllable scenario CS and a set of uncontrollable scenarios $\mathbb{US}' \subseteq \mathbb{US}$, where $\Pr(\mathbb{US}') > 1-\Delta$, such that*

$$\bigcap_{US_i \in \mathbb{US}'} Sol(Ac(PTPN, CS, US_i)) \neq \emptyset.$$

Definitions 6.15 and 6.16 become equivalent to the standard definitions of weak and strong consistency if we choose $\Delta = 0$, i.e., no risk. Moreover, they formally

define the concepts of weak and strong chance-constrained consistency in terms of a subset of uncontrollable scenarios $\mathbb{US}' \subseteq \mathbb{US}$ with appropriate probability and feasible assignments to the controllable choices in the plan. Determining those assignments is the topic of the subsequent sections.

6.4 Chance-constrained consistency of PTPN’s

The conditional nature of temporal constraints on a PTPN, which might be activated by different combinations of assignments to controllable and uncontrollable choices, makes the evaluation of chance constraints challenging: chance-constrained weak consistency with risk Δ requires the enumeration of enough uncontrollable scenarios admitting a schedule to cover a probability mass of at least $1 - \Delta$, while chance-constrained strong consistency demands that fixed controllable choices yield a feasible schedule with at least that same probability.

Evaluating the feasibility of the chance constraint is a key step for risk-bounded conditional scheduling, specially for chance-constrained strong consistency. In fact, one important contribution of our approach is the framing of chance constraint evaluation for strongly consistent plans as a discrete “diagnostic” process, where “symptoms” correspond to inconsistencies in temporal constraints and “diagnoses” are pathological uncontrollable scenarios. Recalling our example in Section 6.2, the pathological scenarios for an overall temporal constraint of 30 minutes were $Slip = True$ if we chose to ride a bike to work, or $Accident = True$ if we chose to drive a car. Evaluating chance constraints is performed differently in weak and strong consistency, as detailed in the following. Throughout the algorithms in this section, the function $NextBest(cnfs)$ is implemented using CDA* in order to enumerate candidate solutions in best-first order while taking previously discovered conflicts $cnfs$ into account. A detailed discussion of how to perform this enumeration can be found in [Williams and Ragno, 2007].

Definition 4.5 in Section 4.5.1 introduces a labeled constraint representation of PTPN’s that makes explicit the connection between each temporal constraint and the choices that activate it. In the following, Section 6.4.1 presents an algorithm for

checking chance-constrained weak consistency of a PTPN, followed by our formulation of chance-constrained strong consistency, a more useful concept within CLARK, as a “diagnosis” problem solved by CDA* in Section 6.4.2.

6.4.1 Chance-constrained weak consistency

Determining weak consistency is special in the sense that we are allowed to assume that the true uncontrollable scenario will be revealed to the agent before it has to make any decisions. This is what happens, for example, when we use a smart phone to determine the state of traffic before committing to a specific route to some destination. By doing so, we implicitly assume that the state of the roads will not change during the trip. Since the agent knows which uncontrollable scenario is the true one, the problem is no longer stochastic and the agent can pick the best possible assignment to the controllable choices in terms of reward for that specific uncontrollable scenario. This is exactly the process that Algorithm 6.1 reproduces.

For each possible uncontrollable scenario (US), Algorithm 6.1 searches for assignments to the controllable choices ($candCS$) that maximize reward while yielding a feasible set of constraints. If it is able to find it, it adds US to the set of consistent scenarios (Line 12) and marks CS as the optimal assignment to controllable choices given US (Line 13). However, if the search algorithm runs out of candidate assignments to controllable choices, it marks US as being inconsistent (Line 7). If the probability of inconsistent scenarios ever violates the overall chance constraint Δ , we are guaranteed that the plan is not weakly consistent within the given risk bound. Similarly, if the probability of consistent scenarios surpasses $1-\Delta$, we can return the optimal mapping from uncontrollable to controllable scenarios. It is worthwhile to notice that uncontrollable scenarios are always disjoint, so computing probabilities in Lines 8 and 14 consists of just adding together the probability for each scenario.

Revealing all uncertainty associated with a plan might be as hard and costly as the plan itself, so determining CCWC might be of limited use in some cases. In situations where the cost of sensing and coordination between agents is considerable, it might be better to extract a policy from the plan description that is robust to a

Algorithm 6.1: Chance-constrained weak consistency.

Input: PTPN N , risk threshold Δ .

Output: Mapping $UtoC$ from uncontrollable to controllable scenarios.

```

1 Function CCWC( $N, \Delta$ )
2    $UtoC \leftarrow \emptyset, conUS \leftarrow \emptyset, incUS \leftarrow \emptyset$ 
3   for  $US \in \mathbb{US}$  do
4      $cConf \leftarrow \emptyset, nextUS \leftarrow \text{False}$ 
5     while not  $nextUS$  do
6       if  $(candCS \leftarrow \text{NextBest}(cConf)) = \emptyset$  then
7          $incUS \leftarrow incUS \cup US, nextUS \leftarrow \text{True}$ 
8         if  $\Pr(incUS) > \Delta$  then
9           return FAIL
10        else
11          if  $Ac(PTPN, candCS, US)$  is consistent then
12             $conUS \leftarrow conUS \cup US, nextUS \leftarrow \text{True}$ 
13             $UtoC[US] \leftarrow candCS$ 
14            if  $\Pr(conUS) \geq 1 - \Delta$  then
15              return  $UtoC$ 
16            else
17               $cConf \leftarrow \text{LearnConflict}(cConf, candCS)$ 
18  return  $UtoC$ 

```

large enough fraction of the uncertainty in the plan. This is the topic of the next section.

6.4.2 Chance-constrained strong consistency

Differently from weak consistency, which considers the feasibility of each uncontrollable scenario separately, Definition 6.16 for strong consistency connects all uncontrollable scenarios $US \subseteq \mathbb{US}'$ together by requiring that they share a common schedule. Thus, evaluating the chance constraint for a strongly consistent policy becomes a challenging and key subproblem, since we now have to consider how different subsets of constraints become active and interact across many possible different realizations of the uncontrollable choices. In weak consistency, the given uncontrollable scenario clearly revealed which temporal constraints dependent on uncontrollable choices could

be active in the plan. In the case of strong consistency, the subset of temporal constraints that must be satisfied to meet the chance constraint is not known a priori, and finding it is computationally difficult.

Recall that, within CLARK, the conditional temporal constraints forming a PTPN in RAO*'s policy estimate have all controllable choices assigned their (heuristically estimated) optimal values, and let this assignment to controllable choices be denoted by CS . In this case, the labeled constraints in the PTPN can be split into three mutually exclusive groups, defined in terms of the probability of activation of their labels:

- **Necessarily Active (NA)**: all constraints such that $\Pr(L_i|CS)=1$, which includes empty labels (an empty label is always active). This is the set of all labels L_i made True by CS ;
- **Necessarily Inactive (NI)**: all constraints such that $\Pr(L_i|CS)=0$. This is the set of all labels L_i made False by CS . These constraints are all inactive and do not influence plan feasibility.
- **Potentially Active (PA)**: remaining constraints, for which $0 < \Pr(L_i|CS) < 1$. This is the set of all labels L_i containing assignments to uncontrollable choices that have not been made False by CS .

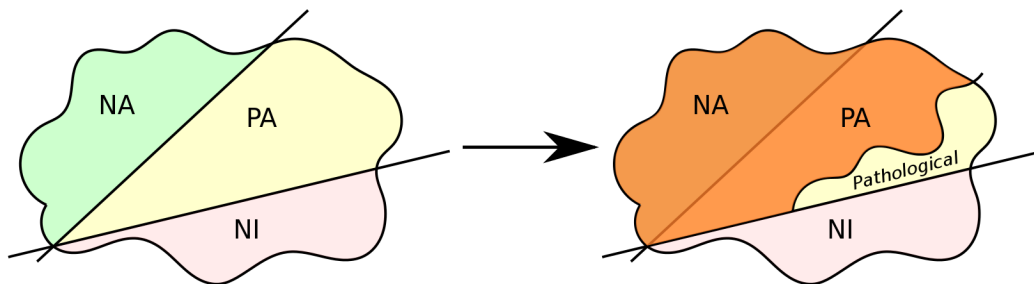


Figure 6-2: Partition of the constraints induced by an assignment to the controllable choices.

This split of the labeled constraints is schematically depicted on the left side of Figure 6-2. Given this partition, the subsequent evaluation of the chance constraint can be graphically represented by the right side of Figure 6-2. In this figure, our

algorithm is trying to find a subset of the temporal constraints in $NA \cup PA$ that is consistent and has probability greater than $1 - \Delta$ of containing the true set of active constraints for any possible uncontrollable scenario. If it succeeds, the strongly consistent schedule for the PTPN can be computed by form the conjunction of temporal constraints contained in this subset. The chance constraint becomes important in this computation, for it allows temporal infeasibility to be resolved by dropping constraints activated by uncontrollable choices. This is done at the expense of increasing the risk of the resulting schedule being inconsistent. The larger the admissible risk bound Δ , the more constraints can be dropped. It is easy to see that this subset of constraints must always cover all of NA in order for a strongly consistent schedule to exist. Also, no effort should be spent trying to cover any portion of NI .

It is usually the case that infeasibility with respect to the constraints in $NA \cup PA$ only manifests itself in a handful of scenarios that activate constraints that are hard to satisfy. Hence, one key contribution from this chapter is to frame the evaluation of the chance constraint as a diagnostic process. The key insight behind the use of a conflict-directed method is to be able to quickly isolate temporal constraints causing infeasibility and evaluate whether they incur a significant amount of risk. The numerical results in Section 6.5 show that our conflict-directed approach is able to detect violation of the chance constraint Δ more efficiently than prior art based on chronological search methods.

We frame the problem of finding a subset of PA that allows the chance constraint to be satisfied as the following Optimal Satisfiability (OpSAT) [Williams and Ragno, 2007] problem:

1. **Decision variables:** Binary $B_i \in \{\text{True}, \text{False}\}$ for each labeled constraints in PA ;
2. **Objective function:** $\min \Pr(L_{B=\text{False}})$, where $L_{B=\text{False}}$ is the set of all labels of constraints such that $B_i = \text{False}$;
3. **Constraints:** $\Pr(L_{B=\text{False}}) \leq \Delta$ and $C_{B=\text{True}} \wedge NA$ is consistent, where $C_{B=\text{True}}$ is the set of all constraints such that $B_i = \text{True}$.

The Boolean variable B_i represents whether the labeled constraints $L_i \Rightarrow C_i$ in the constraint region PA is covered or not. In this formulation, a schedule extracted from a consistent set of temporal constraints $C_{B=\text{True}} \wedge NA$ has risk given by

$$\Pr(L_{B=\text{False}}) = \Pr\left(\bigcup_{i:B_i=\text{False}} L_i\right). \quad (6.1)$$

The probability (6.1) is computed using the Inclusion-Exclusion principle. However, since we are only concerned about the risk being below Δ , it is possible to compute simpler upper and lower bounds for $\Pr(L_{B=\text{False}})$ using the Bonferroni inequalities [Comtet, 1974], as shown in Algorithm 6.2. A particular case of the Bonferroni inequalities is the upper bound $\Pr(L_{B=\text{False}}) \leq \sum_{i:B_i=\text{False}} \Pr(L_i)$, known as Boole's inequality.

Algorithm 6.2: Sequential probability approximations.

Input: Set of labels $L_{B=\text{False}}$, prob. model P , risk threshold Δ .
Output: Whether $\Pr(L_{B=\text{False}})$ is below, above, or is equal to Δ .

```

1 Function Bonferroni( $L_{B=\text{False}}, P, \Delta$ )
2    $bound \leftarrow 0$ 
3   for  $i = 1$  to  $|L_{B=\text{False}}|$  do
4      $[e_1, \dots, e_m] \leftarrow$  All subsets of  $L_{B=\text{False}}$  with  $i$  elements.
5      $probInc \leftarrow \sum_{i=1}^m \Pr(e_i)$ 
6     if  $i$  is odd then
7        $bound \leftarrow bound + probInc$ 
8       if  $bound < \Delta$  then
9         return BELOW
10    else
11       $bound \leftarrow bound - probInc$ 
12      if  $bound > \Delta$  then
13        return ABOVE
14  return EQUAL

```

The worst case performance of Algorithm 6.2 is equivalent to computing (6.1) and comparing it with Δ . The procedure for determining CCSC of PTPN's in Algorithm 6.3 is explained below.

Lines 2,3 : partitions constraints as described in this section. If NA is inconsistent,

terminates with failure;

Lines 5-7 : creates an OpSAT instance for risk minimization as described in this section. Risk bounds are computed with Algorithm 6.2. If the chance constraint is violated, terminates with failure;

Lines 9-12 : if the chance constraint is satisfied with a consistent PSTNU, returns the PSTNU from which the strongly consistent schedule can be extracted. Otherwise, learns a new conflict in terms of infeasible constraints.

Algorithm 6.3: Chance-constrained strong consistency.

Input: PTPN N , controllable scenario CS , risk threshold Δ .

Output: Feasible PSTNU.

```

1 Function CCSC( $N, CS, \Delta$ )
2    $\{NA, NI, PA\} \leftarrow Partition(CS, N)$ 
3   if  $NA$  is inconsistent then
4     return FAIL
5    $bConf \leftarrow \emptyset$ 
6   while ( $candB \leftarrow NextBest(bConf) \neq \emptyset$ ) do
7     if  $\Pr(L_{B=False}(candB)) > \Delta$  then
8       return FAIL
9     if  $C_{B=True}(candB) \wedge NA$  is consistent then
10      return  $C_{B=True}(candB) \wedge NA$ 
11    else
12       $bConf \leftarrow LearnConflict(bConf, candB)$ 
13  return FAIL

```

6.5 Numerical chance constraint evaluation

The results in this section concern Algorithm 6.3, which is used within CLARK to establish if PTPN's are strongly consistent. Figure 6-3 compares the relative average performance of our conflict-directed approach versus the current practice using chronological search (CS) when evaluating the chance constraint for strongly consistent policies on a set of randomly generated PTPN's. Conflict-directed methods

have the additional overhead of learning conflicts whenever infeasibility is detected, so it was not clear whether they would perform better when trying to evaluate the chance constraint.

When dealing with hand-written examples, searching for a CCSC policy using CS or CDA* yielded the same performance. For small instances, there is no practical value on learning conflicts, since CS is able to explore all solutions in a matter of milliseconds. Thus, no useful conclusions could be drawn. Therefore, we randomly generated labeled temporal constraints conditioned on controllable and uncontrollable choices with moderately-sized domains (about 10 elements each). Probability models were also randomly generated. Simple temporal constraints were created between random pairs of nodes with varying bounds. The goal was to either find a CCSC schedule or return that no solution existed. Our implementation of CDA* searches for a strongly consistent policy as explained in Section 6.4.2, while CS tries to find a feasible STN by relaxing temporal constraints in best-first order of risk.

A problem with N conditional constraints induces a search space of size 2^N possible solutions. Both algorithms were run until the first candidate satisfying the risk bound Δ was found or no more solutions were available. Whenever more than one solution existed, CDA* returned the one incurring minimum risk. For relatively small plans with no more than 10 conditional constraints, we see that CS and CDA* showed very similar performances. However, if one increases the size of the problem by a few more constraints, we see a strong exponential growth in the time required by CS to either find a solution or return that no solution exists. Our approach using CDA*, on the other hand, kept its performance virtually unchanged. Despite the exponential trend in Figure 6-4 for CDA*, we see that it happens at a much smaller rate than for CS.

It is worthwhile to mention that CS was able to find feasible schedules quickly when the “hard” temporal constraints causing infeasibility had low probabilities assigned to them. In this cases, it was easy to restore feasibility by relaxing these low probability constraints while meeting the chance constraint. It ran into troubles, however, whenever temporal constraints causing infeasibility were assigned high probabilities. In these cases, CS preferred to explore numerous lower risk alternatives

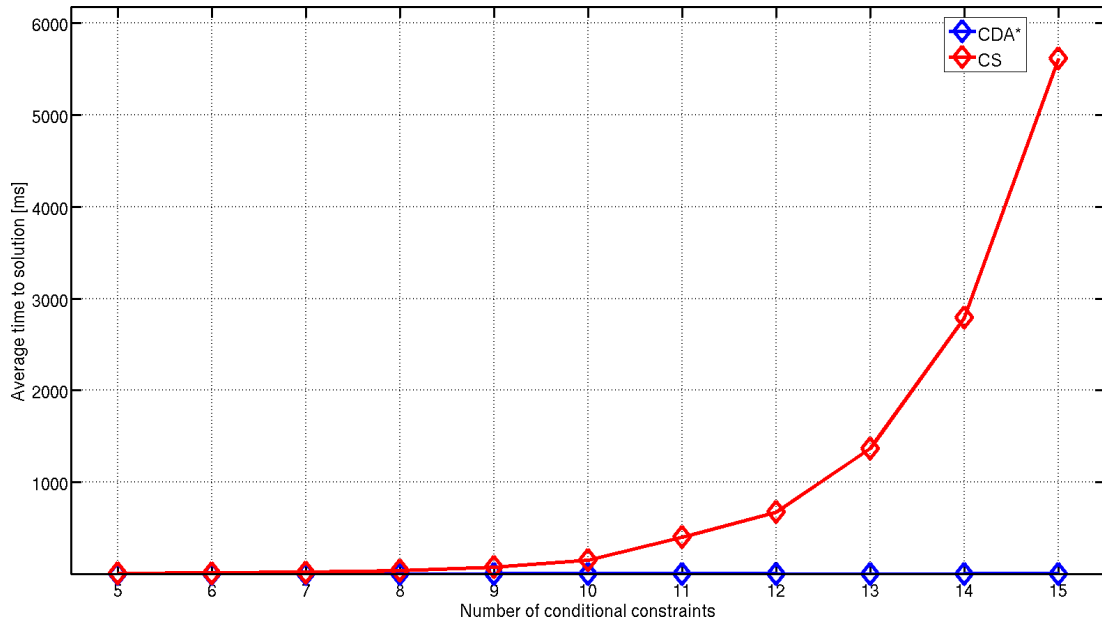


Figure 6-3: Average time to solution for CDA* *versus* CS.

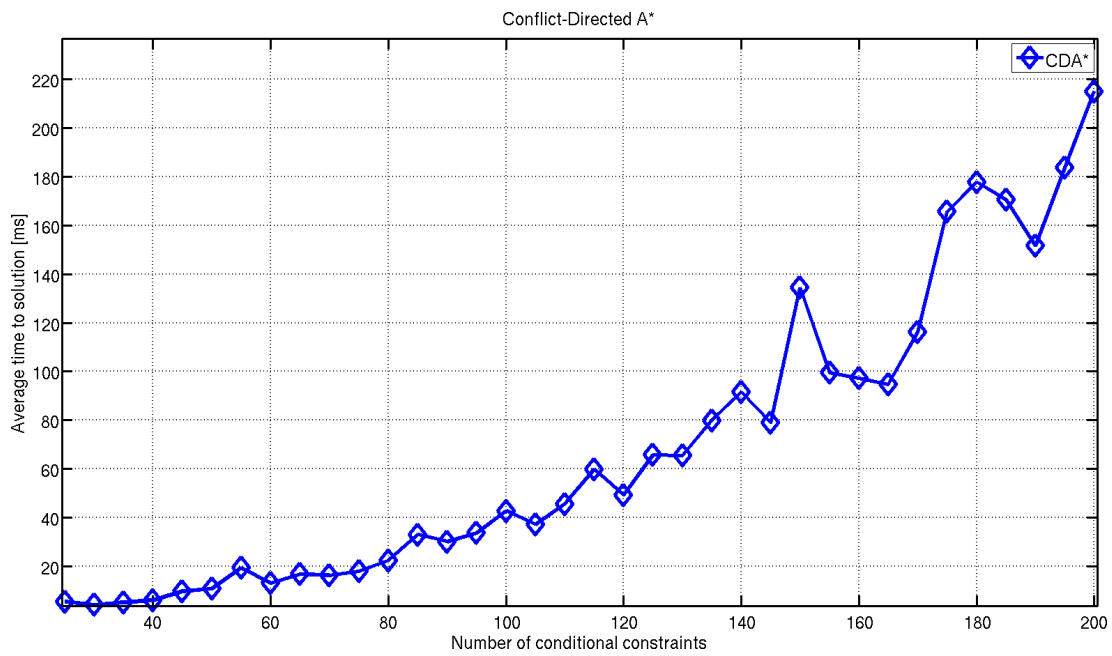


Figure 6-4: Average time complexity growth for CDA*.

before realizing that the source of infeasibility had high probability and, therefore, violated the chance constraint. In these situations, the conflict extraction overhead paid off by quickly revealing the problematic constraints with high probabilities and determining that the chance constraint was infeasible.

6.6 Conclusions

In support of this thesis' goal of generating risk-bounded conditional temporal plans, this chapter extends the results for unconditional scheduling under uncertainty in Chapter 5 to risk-aware scheduling of PTPN's. It also introduces chance-constrained weak and strong consistency, two novel types of risk-aware consistency guarantees for PTPN's. For the efficient determination of chance-constrained strong consistency, we present a "diagnostic" approach based on CDA* [Williams and Ragno, 2007] that identifies sets of conflicting temporal constraints that can be safely ignored in order to restore temporal consistency.

With the risk-aware conditional scheduling methods developed in this chapter, we have all the necessary tools to construct CLARK, a chance-constrained conditional temporal planner for agents that must operate under uncertainty. This is the topic of the next chapter.

Chapter 7

Integrated CLARK experiments

“In theory, there is no difference between theory and practice. In practice, there is.”

Probably Jan L. A. van de Snepscheut

Previous chapters are concerned with the individual components of CLARK, first described in Section 1.4, and how these components relate and compare to the state of the art in their specific fields. In contrast, this chapter pursues a holistic view of CLARK by first revisiting its internal structure in greater detail, and showing how it is integrated within *Enterprise* [Burke et al., 2014, Timmons et al., 2015], a system integration architecture for closed-loop control of autonomous systems. Next, we explain the process through which pSulu [Ono and Williams, 2008, Blackmore et al., 2011, Ono et al., 2012b, Ono et al., 2012a, Ono et al., 2013], a chance-constrained path planner, is incorporated within CLARK as a constraint checker for planning problems featuring uncertain continuous dynamics and collision constraints. Finally, we present CLARK achieving its goal of generating risk-bounded conditional plans over rich sets of mission constraints in two application domains that can greatly benefit from risk-aware autonomy: collaborative human-robot manufacturing and data retrieval agents.

7.1 The CLARK system

As first mentioned in Section 1.4, the **C**onditional **P**lanning for **A**utonomy with **R**isk (CLARK) system is a combination of the different tools developed in this thesis for the generation of chance-constrained, conditional temporal plans for autonomous agents operating under uncertainty. In this section, we provide a detailed description of its different elements, as depicted by the block diagram in Figure 7-1.

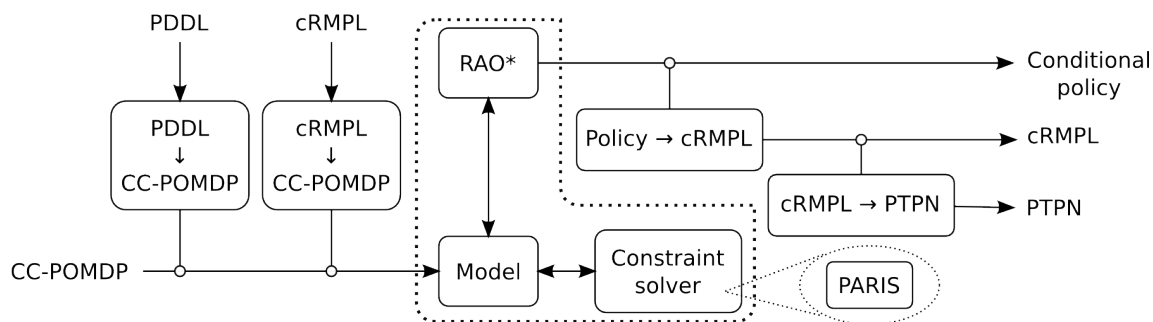


Figure 7-1: Block diagram of the different elements composing CLARK. Inputs are shown to the left of CLARK’s core (surrounded by the dotted line), while outputs are shown to the right (repeated from Figure 1-9 on page 39).

Even though this may be clear from CLARK’s block diagram, it is worthwhile to mention that CLARK and RAO* (Chapter 3) are *not* interchangeable concepts, and should not be confused with each other. The RAO* algorithm is a *component* of CLARK that generates risk-bounded conditional policies from a CC-POMDP description, which could be replaced by any other algorithm that served the same purpose (possible improvements of RAO* are discussed in Chapter 8 and Appendix A). At the same time, the term CLARK is used to refer to the ensemble shown in Figure 7-1, which combines RAO*, risk-aware constraint checkers, and translators for inputs and outputs.

7.1.1 Inputs

As shown at the bottom left corner of Figure 7-1, the input model to CLARK’s core is always a CC-POMDP, which can either be directly supplied to the system,

or generated from some other type of user-provided input. In this section, we detail how CC-POMDP models can be directly specified to CLARK, or generated as an extension to an existing PDDL [McDermott et al., 1998, Fox and Long, 2003, Fox and Long, 2006, Hoffmann and Edelkamp, 2005, Gerevini and Long, 2005] definition. For the mapping between cRMPL execution to CC-POMDP shown in Figure 7-1, please refer to Section 4.5.2.

CC-POMDP

A CC-POMDP input to CLARK is essentially a software entity implementing Definition 3.1, plus a few other helper functions. In the following, let $s \in \mathcal{S}$ be a full state assignment; $a \in \mathcal{A}$ be an action description; $o \in \mathcal{O}$ be an observation symbol; and $C \in 2^{\mathcal{C}}$ be a set of constraints, where $\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{C}$ are as in Definition 3.1.

In order to facilitate understanding, we ground the different functions composing a CC-POMDP input to CLARK on the example shown in Figure 7-2. The functions in a CC-POMDP model are as follows:

- **actions(s)**: returns a set $A \in 2^{\mathcal{A}}$ of applicable actions in state s . In Figure 7-2, the set A at the robot's position is $A = \{\text{up, down, left, right}\}$, each element depicted by an arrow. At the top left corner square, the set would be $A = \{\text{down, right}\}$, while at the goal position G , $A = \emptyset$. A state \mathbf{s} such that $\text{actions}(\mathbf{s}) = \emptyset$ is a *terminal state*.
- **state_transitions(s, a)**: corresponds to the stochastic state transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$. Given a state s and an action a , returns a set of tuples (s_{next}, p) , where $s_{next} \in \mathcal{O}$ is a possible next state from s and $p \in [0, 1]$ is the probability of transitioning from s to s_{next} . On the bottom right quadrant of Figure 7-2, we see a graphical representation of the state transition function for a robot that, when told to move in one direction, may also stay put or slip to the sides. In the particular example in Figure 7-2, we have

$$\begin{aligned} \text{state_transitions}(s_{green}, \text{right}) = \\ \{(s_{red}, 0.8), (s_{green}, 0.1), (s_{blue}, 0.05), (s_{yellow}, 0.05)\}. \end{aligned}$$

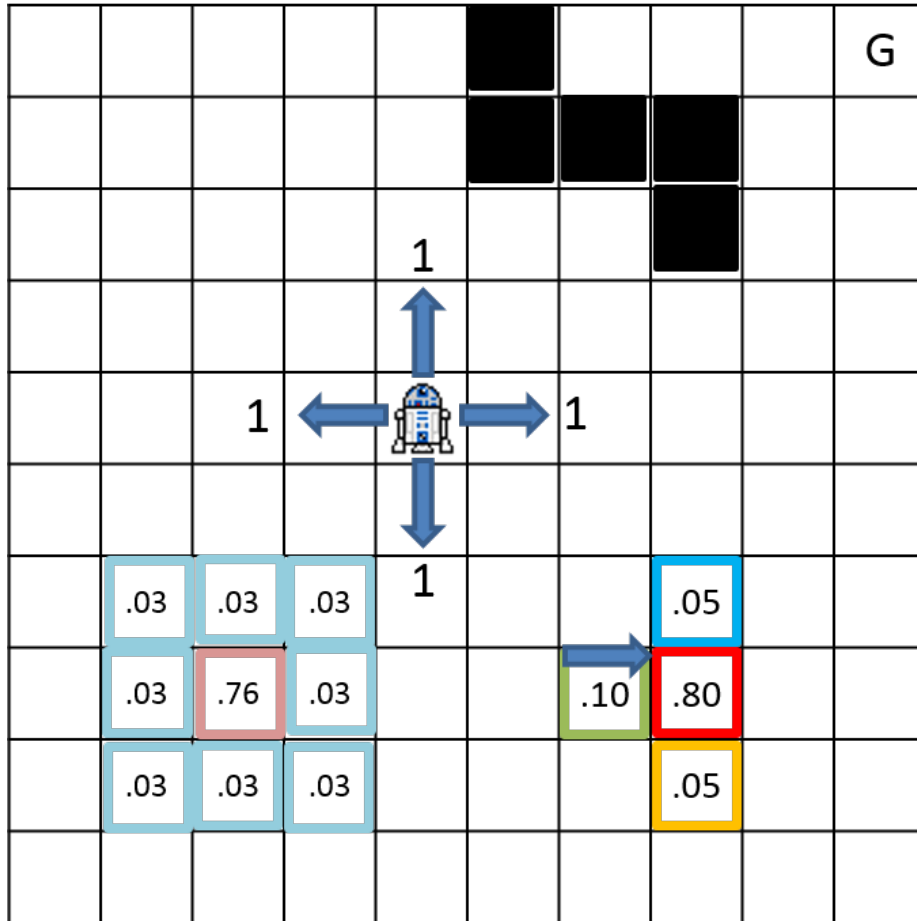


Figure 7-2: Simple scenario that can be modeled as a CC-POMDP: a robot with unreliable movements and noisy position sensors that must move around a grid to get to its goal G . Execution should be carried with bounded risk of colliding against obstacles (black squares).

Note that the probabilities in the set generated by `state_transitions` must *always sum to one*, since they represent a probability distribution over next states.

- `observations(s)`: corresponds to the stochastic observation function $O : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$. Given a state s , returns a set of tuples (o, p) , where $o \in \mathcal{O}$ is a possible observation symbol at s and $p \in [0, 1]$ is the probability of observing o at s . On the bottom left quadrant of Figure 7-2, we see a graphical depiction of a noisy sensor that returns the current position on the grid with high probability, or one of the adjacent cells with uniformly low probability. In this particular

example, let s_{red} be the red square on the bottom left quadrant. In this case, we have

$$\text{observations}(s_{red}) = \{(o_{s_{red}}, 0.76)\} \cup \{(o_{s_{adj}}, 0.03) | s_{adj} \text{ is adjacent to } s_{red}\}.$$

Note that, as with `state_transitions`, the probabilities in the set generated by `observations` must *always sum to one*, since they represent a probability distribution over observation symbols.

- `value(s, a)`: plays the role of the reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, returning the value of executing action a at state s . We use the term *value*, instead of *reward*, because our implementation allows `value(s, a)` to output either rewards, which are maximized, or costs, which are minimized. In Figure 7-2, the numbers next to arrows represent the *cost* of moving in either direction. Therefore, for Figure 7-2, we have

$$\text{value}(s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}.$$

- `heuristic(s)`: this function returns the *admissible* state value heuristic $h_Q(s)$ used to compute the admissible belief value heuristic

$$h_Q(b_{k+1}) = \sum_{s_{k+1}} b(s_{k+1})h_Q(s_{k+1}), \quad (3.32)$$

which in turn is used in the heuristic value-to-go estimate

$$\hat{Q}(b_k, a_k) = \sum_{s_k} R(s_k, a_k)b(s_k) + \sum_{o_{k+1}} \text{Pr}(o_{k+1} | a_k, b_k)h_Q(b_{k+1}). \quad (3.34)$$

In Figure 7-2, since the robot may slide diagonally when moving, a possible admissible state heuristic could be the Chebyshev distance between s and G , i.e.,

$$h_Q(s) = \max(|x_G - x_s|, |y_G - y_s|), \quad (7.1)$$

where (x_s, y_s) and (x_G, y_G) are, respectively, the horizontal and vertical coordinates of a grid state s and the goal cell G .

- **state_risk(s)**: for each set of constraints C , this function returns 1 if state s violates C . Therefore, it plays the role of the constraint violation function $c_v(s_k, C)$ in

$$r_b(b_k, C) = \sum_{s_k \in S} b(s_k) c_v(s_k, C). \quad (3.15)$$

Running into one of the obstacle cells is an example of constraint violation in Figure 7-2.

- **execution_risk_heuristic(s)**: this function returns the *admissible* state execution risk heuristic $h_{er}(s_{k+1}|\pi)$ used to compute the admissible belief execution risk heuristic

$$h_{er}(b_{k+1}|\pi) = \sum_{s_{k+1}} b(s_{k+1}) h_{er}(s_{k+1}|\pi). \quad (3.29)$$

In the lack of a better admissible estimate, one can always choose

$$\text{execution_risk_heuristic}(s) = \text{state_risk}(s).$$

Note that the **state_risk(s)** and **execution_risk_heuristic(s)** functions in a CC-POMDP input encapsulate all constraint theory-dependent quantities necessary to the evaluation of mission risk, causing the higher level planning in CLARK to be performed in a theory-agnostic setting. This property is useful later in this chapter, when CLARK is used to generate solutions for planning domains involving risk-aware path planning and scheduling.

PDDL

As mentioned in Chapter 2, PDDL [McDermott et al., 1998] and its enhancements [Fox and Long, 2003, Fox and Long, 2006, Hoffmann and Edelkamp, 2005, Gerevini and

Long, 2005] have become standard languages for the definition of planning domains, with most current planners implementing either all language features, or subsets useful for specific purposes. Therefore, adding support to PDDL is an important step towards CLARK’s use as a standalone risk-aware planning tool. To the best of our knowledge, however, there is currently no PDDL variant that supports all modeling features for which CLARK is designed, such as probabilistic temporal uncertainty, sensing actions, and chance-constrained planning. At the same time, this section does not seek to provide a complete encoding of all PDDL features into CC-POMDP, which is beyond the scope of this thesis. Instead, here we restrict ourselves to a mapping from PDDL domains with *durative actions* [Fox and Long, 2003] to CC-POMDP that retains the convenient plan operator semantics defined in PDDL, while also adding support, among other things, to probabilistic scheduling and chance-constrained planning.

The key idea behind the conversion from durative PDDL domains to CC-POMDP is to map durative PDDL actions to primitive cRMPL episodes (Section 4.4.1) by leveraging the constraints defined in Section 4.4.2. The steps involved in this conversion are listed in the following.

1. A grounded durative PDDL action a_d becomes a primitive cRMPL episode e_a with suitable duration and action;
2. Each predicate in a_d becomes a Boolean cRMPL state variable;
3. Preconditions and effects in a_d become cRMPL assignment constraints;
4. Preconditions are added as **at start** state constraints in e_a , while effects are added as **at end** state constraints;

Figure 7-3 shows an example of a durative action encoded in PDDL2.1, and Figure 7-4 shows one particular grounding¹ of Figure 7-3: a traversal action from location 11 to location 12 by a rover `rov` that takes 20 units of time on average, with standard

¹Grounded PDDL actions have all their parameters, which start with “?”, assigned constant values.

```

(:durative-action move
 :parameters (?self - rover ?loc1 - location ?loc2 - location)
 :duration (= ?duration (traversal_time ?loc1 ?loc2))
 :condition
  (and
   (at start (at ?self ?loc1))
   (at start (idle ?self))
  )
 :effect
  (and
   (at end (not (at ?self ?loc1)))
   (at end (at ?self ?loc2))))

```

Figure 7-3: Durative action in PDDL2.1 representing a traversal between two locations by a rover.

deviation of $\sqrt{2.5}$ units of time. The episode resulting from running the code in Figure 7-4 is shown in Figure 7-5. Even though the encoding in Figure 7-4 is quite verbose, it exactly corresponds to the aforementioned conversion steps and can be efficiently automated, since cRMPL is an extension of Python.

One important advantage of using cRMPL episodes to encode durative PDDL actions is that one can make use of cRMPL’s support to probabilistic durations and chance constraints to represent actions with uncontrollable durations, and which must be executed under bounded risk. This extended expressiveness of cRMPL is also used in Section 7.1.4, in which we show how risk-bounded path planning can be incorporated into CLARK by representing map traversals as chance-constrained cRMPL episodes.

With this mapping from durative PDDL actions to cRMPL episodes, the final step is to define a CC-POMDP model that operates with these episodes. For that, let $pred(s)$ be the set of true Boolean predicates in state s ; $constr(s)$, the set of constraints in s ; and $event(s)$, the temporal event associated with s . Following the notation from the previous section, the CC-POMDP model is defined as follows.

- **actions(s)**: returns a set $A \in 2^A$ of applicable *episodes* in state s . An episode

```

move_ep = Episode(action='(move rov l1 l2)',
                  duration={'ctype': 'uncontrollable_probabilistic',
                             'distribution': {'type': 'gaussian',
                                                'mean': 20.0,
                                                'variance': 2.5}})

preds = [StateVariable(name='(at rov l1)',
                       domain_dict={'type': 'finite-discrete',
                                      'domain': ['True', 'False']}),
         StateVariable(name='(at rov l2)',
                       domain_dict={'type': 'finite-discrete',
                                      'domain': ['True', 'False']}),
         StateVariable(name='(idle rov)',
                       domain_dict={'type': 'finite-discrete',
                                      'domain': ['True', 'False']})]

start_cts = AssignmentStateConstraint(scope=[preds[0], preds[2]],
                                     values=['True', 'True'])
ends_cts = AssignmentStateConstraint(scope=[preds[0], preds[1]],
                                    values=['False', 'True'])

move_ep.add_start_state_constraint(start_cts)
move_ep.add_end_state_constraint(ends_cts)

```

Figure 7-4: Grounded version of the durative action from Figure 7-3 written in cRMPL. Unlike PDDL, the cRMPL version supports probabilistic uncontrollable durations.

is applicable in s if its preconditions encoded as **at start** constraints are contained within $pred(s)$.

- **state_transitions(s,a)**: in case of deterministic transitions, returns a singleton $\{(s', 1.0)\}$, where $pred(s')$ is obtained by adding and removing, respectively, the positive (true predicates) and negative (false predicates) effects of a ; $constr(s')$ contains all elements from $constr(s)$, plus the duration of a (which may be probabilistic) and a $[0, \infty]$ simple temporal constraint from $event(s)$ to the start event of a ; and $event(s')$ is the end event of a . Probabilistic transitions are defined in a similar way, with the extra step that constraints in $constr(s')$ have an assignment to a probabilistic choice added to their labels (see Definition

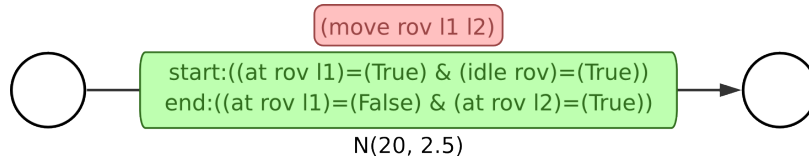


Figure 7-5: Depiction of the episode generated by the cRMPL code in Figure 7-4.

4.5).

- `observations(s)`: for fully observable PDDL models, this function returns a unique identifier for s . If one wishes to model a partially observable extension to the PDDL model, this function can be suitably defined to meet modeling needs.
- `value(s, a)` and `heuristic(s)`: dependent on the particular problem being modeled.
- `state_risk(s)` and `execution_risk_heuristic(s)`: as with the previous two functions, these are dependent on the particular constraints used in the model. In this chapter, we are mostly concerned with probabilistic temporal constraints and collision constraints for path planning.

7.1.2 Outputs

The solution to the input CC-POMDP given to CLARK is an optimal conditional policy that entails a consistent conditional constraint system with high probability. Then, as shown in Figure 7-1, this policy can be converted into an output cRMPL program, which can later be used as an *optimal subroutine* within a hierarchical composition of cRMPL programs. This section explains how this conversion process takes place.

A cRMPL program can serve as both input and output to CLARK. There are, however, significant distinctions in structure and semantics between input and output cRMPL programs. When given as input, cRMPL programs are meant to represent high-level constraints, including goals, that the Plant must satisfy during execution, along with expert “advice” from the cRMPL programmer to the program executive

in the form of flexible choice among subprocedures. Therefore, input cRMPL programs are supposed to represent *families* of potential solutions to the temporal control problem represented by the cRMPL program, which the program executive explores while reasoning about the feasibility of state and temporal constraints. On the other hand, output cRMPL programs, which are derived from the CC-POMDP policy π computed by RAO*, represent a *commitment* to one specific risk-bounded, optimal candidate within this family of potential solutions. Therefore, even though output cRMPL programs can still have uncontrollable choices representing runtime observations that condition program execution, they no longer contain controllable choices, since these are assigned their optimal values by RAO*. Algorithm 7.1 shows the pseudocode used to convert RAO* policies into output cRMPL programs.

Algorithm 7.1: Conversion from CC-POMDP policies to cRMPL programs.

Input: CC-POMDP policy π , hypergraph G from which π was obtained.
Output: cRMPL program p representing the policy.

```

1 Function PolicyToRMPL( $G, \pi$ )
2    $p \leftarrow$  RMPyL(RecursiveConvert( $G, \pi, \text{Root}(G)$ ))
3   return  $p$ 
4 Function RecursiveConvert( $G, \pi, n$ )
5   if  $n$  is terminal then
6     return Episode(action=None)
7   else
8      $ep \leftarrow$  Episode(action= $\pi(n)$ )
9      $he =$  Hyperedge( $G, n, \pi(n)$ )
10     $obs \leftarrow$  Choice(domain=Values( $he$ ), ctype="probabilistic")
11     $children \leftarrow$  Successors( $he$ )
12     $childEps \leftarrow \emptyset$ 
13    for  $c \in children$  do
14       $childEps \leftarrow childEps \cup$  RecursiveConvert( $G, \pi, c$ )
15    return sequence( $ep, \text{observe}(obs, childEps)$ )

```

Finally, the cRMPL program generated by Algorithm 7.1 can be converted into a PTPN (Section 4.5.1) by translating its content according to the XML schema shown in Appendix C. An example of a PTPN generated from a CC-POMDP policy computed by CLARK can be seen in Figure 7-6.

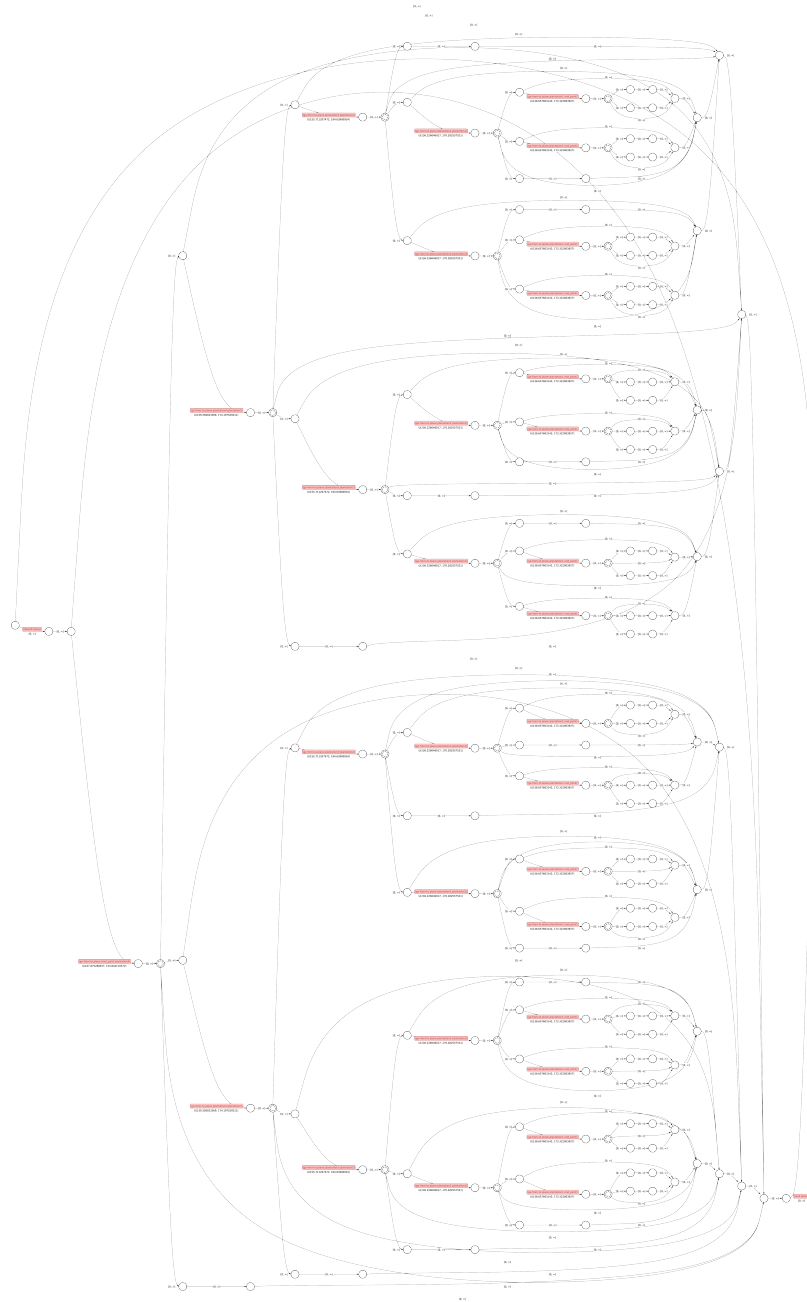


Figure 7-6: Example of a PTPN obtained from a CC-POMDP policy.

7.1.3 Execution on Enterprise

As pointed out in Section 1.4, the execution of PTPN's generated by CLARK, e.g., Figure 7-6, is performed by the Pike dispatcher [Levine and Williams, 2014], and we refer to the combination of CLARK and Pike as the *CLARK executive* (Figure 7-7).

CLARK generates a PTPN for which it can guarantee that a feasible schedule

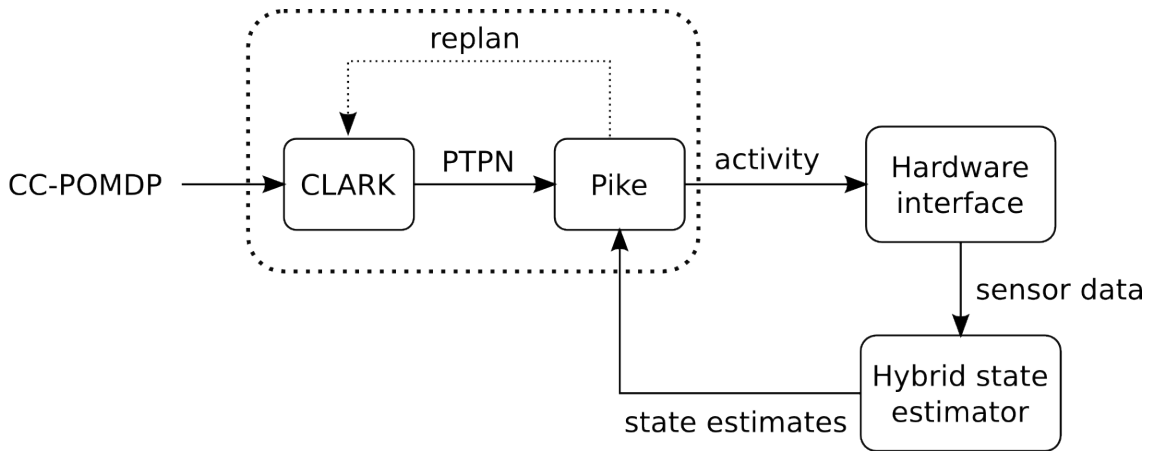


Figure 7-7: The *CLARK executive*, a combination of CLARK and Pike used in closed-loop control applications. The conditional temporal policy generated by CLARK is sent to Pike in the form of a PTPN, which Pike then takes care of dispatching through the physical hardware interface while performing execution monitoring. (repeated from Figure 1-10 on page 41).

exists with high probability, and forwards it to Pike. However, instead of providing a PTPN containing a fixed schedule, CLARK improves the executive’s ability to adapt to disturbances by giving Pike the flexibility to pick times for the temporal events as execution unfolds. In addition to scheduling temporal events in real time, Pike offers an additional line of defense against external disturbances by constantly monitoring plan execution: should Pike detect a disturbance that will cause its current plan to fail, it immediately triggers a replanning signal to CLARK using the current state of the system as the initial belief. This ability is exercised in the first half of Section 7.3, where the CLARK executive’s *reactive robustness through replanning* is assessed in the context of a science retrieval mission for a planetary rover acting under temporal uncertainty and time pressure.

In order to perform its activity-dispatching and execution-monitoring duties, Pike relies on the *hybrid state estimator* in Figure 7-7 to collect sensor readings (camera images, inertial navigation data, point clouds from SLS sensors, etc.) and convert them into discrete logical predicates that Pike uses to monitor execution, such as “the circuit component is on the table” and “the rover is at the relay location”. In support of the hybrid state estimator, this thesis developed a novel robust data fusion

algorithm that is capable of handling measurements from faulty sensors [Santana et al., 2014, Santana et al., 2016a]; and a data-driven algorithm capable of learning expressive hybrid dynamical models that can aid in state uncertainty mitigation when synthesizing CC-POMDP policies [Santana et al., 2015]. We refer the reader to these references and [Timmons, 2013, Lane, 2016] for further details, and to the video at <https://www.youtube.com/watch?v=Fz1s5jAgEew> for a hardware demonstration.

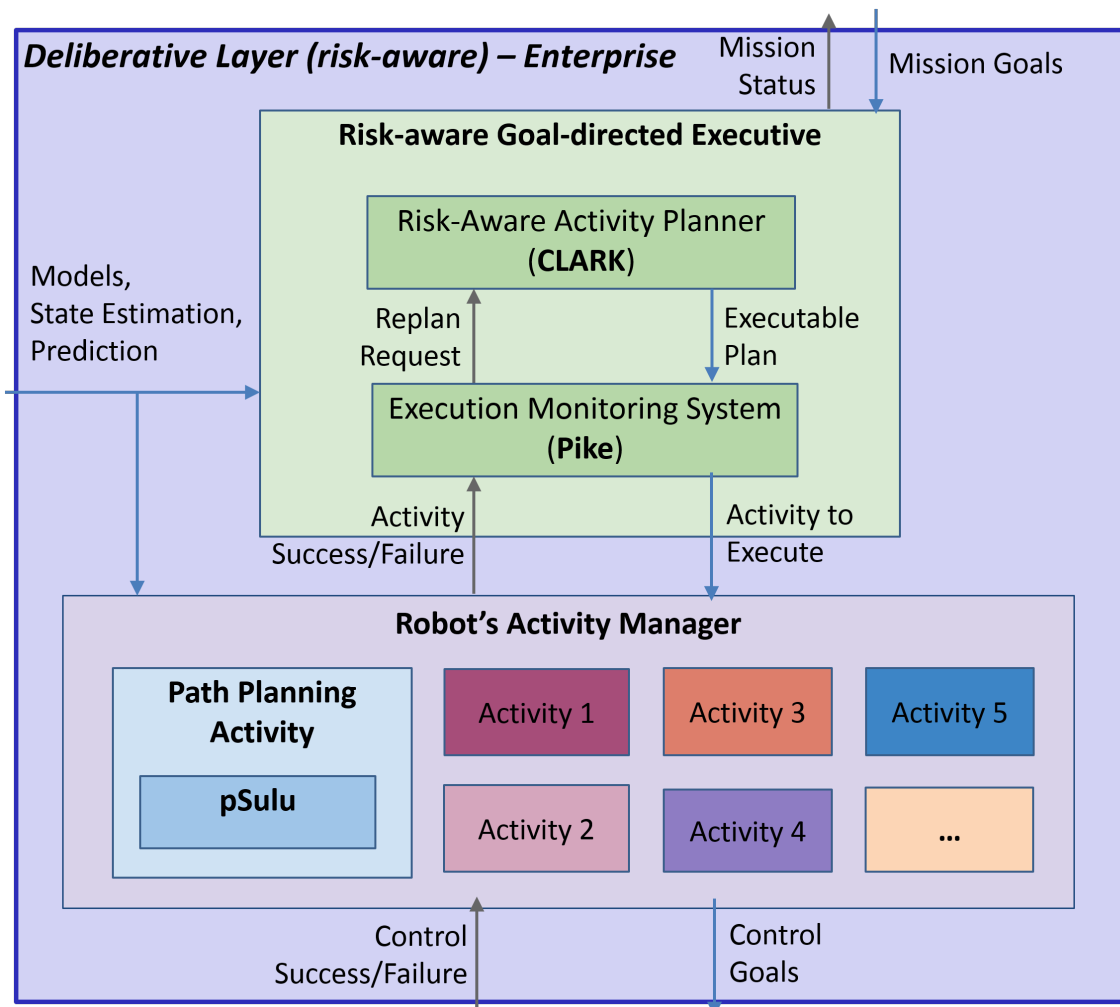


Figure 7-8: CLARK executive as part of Enterprise. The diagram is a courtesy of Catharine McGhan and Tiago Vaquero.

Conditional plans generated by CLARK are executed in simulation or on real hardware through the *Enterprise* [Burke et al., 2014, Timmons et al., 2015] architecture shown in Figure 7-8, which combines the CLARK executive with the hybrid

state estimator from Figure 7-7 and a distributed hardware interface shown as *Robot’s Activity Manager* in Figure 7-8. Enterprise was used for all experiments described in this chapter.

7.1.4 Chance-constrained path planning

The ability to perform path planning for a mobile robot with uncertain dynamics and moving around obstacles is crucial in many robotics domains, including several demonstrations involving CLARK. Moreover, as in the field of Task And Motion Planning [Lozano-Pérez and Kaelbling, 2014, Srivastava et al., 2014, Toussaint, 2015, Fernández-González et al., 2015, Lin et al., 2016], path planning for a robot may not be a goal in itself, but instead a secondary optimization or feasibility check done *in support* of higher level mission goals, such as performing a manufacturing task involving the assembly of several components, or collecting information at different locations around a map. Finally, for robots whose pose and dynamics are uncertain, it may be the case that motion planning can only be realistically performed within non-zero risk of collision. In this section, we explain how pSulu [Ono and Williams, 2008, Blackmore et al., 2011, Ono et al., 2012b, Ono et al., 2012a, Ono et al., 2013], a chance-constrained path planner for robots with uncertain dynamics, is used within CLARK’s CC-POMDP models to handle generation of conditional plans for Plants with continuous dynamics and collision constraints.

The purpose of incorporating pSulu within CLARK can be pictorially understood from Figure 7-9. There, a robot with uncertain pose and dynamics moving on a plane must travel from its initial position (marked “0” at the bottom left corner of the map) to a goal location (marked “10” at the upper middle half) while avoiding collisions with different obstacles, which are shown as gray polygons. However, since the robot’s uncertain pose is represented by Gaussian random variables, there is always a non-zero probability that its true pose might be in collision with one of the obstacles. For that reason, pSulu operates on a risk-bounded path planning framework that, given a risk bound Δ for a traversal, treats Δ as a “resource” that gets distributed amongst

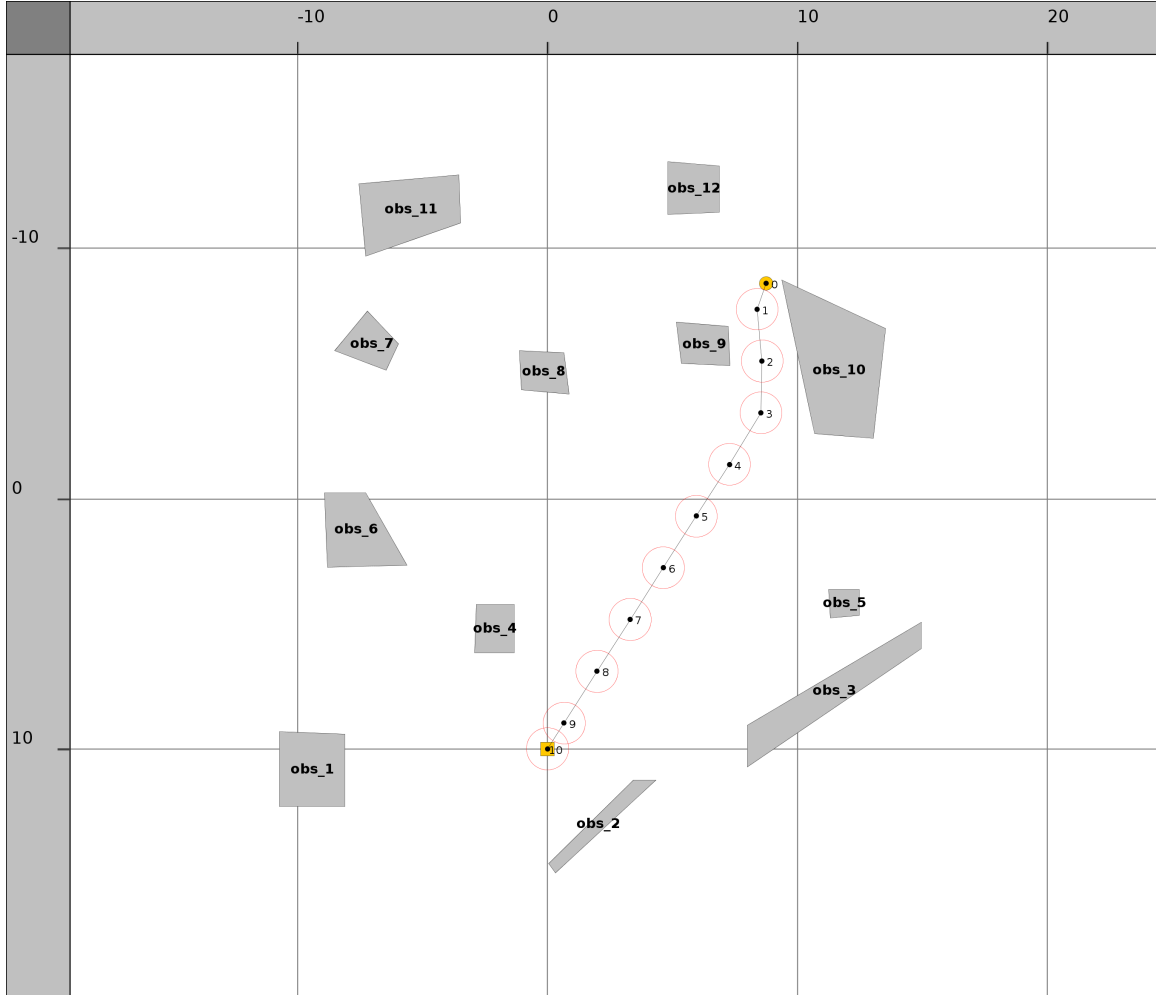


Figure 7-9: A chance-constrained traversal generated by pSulu for a robot with linear dynamics and Gaussian noise. The 3σ ellipses are shown in red.

n trajectory segments such that

$$\sum_{k=0}^n r_{col}(k) \leq \Delta, \quad (7.2)$$

where $r_{col}(k)$ is the probability of colliding at the k -th segment. Equation (7.2) is essentially the same as (3.26) in Section 3.2.4, where we treat different forms of chance constraints that can be imposed on CC-POMDP models.

Within CLARK, we use the pSulu implementation described in [Blackmore et al., 2011, Arantes et al., 2016b]. Letting x_k and u_k denote, respectively, the pose (position and velocity) and continuous control input at time step k for a robot moving on a

plane, the inputs to pSulu are given by:

- the mean x_0 and covariance matrix P_0 for a Gaussian random variable $N(x_0, P_0)$ representing pose uncertainty at the start of the path;
- the goal pose x_g ;
- a linear, time-invariant (LTI) dynamical model with Gaussian disturbances

$$x_{k+1} = \Phi x_k + \Gamma u_k + w_k, w_k \sim N(0, Q), \quad (7.3)$$

where Φ , Γ , and Q are known matrices;

- the number n of trajectory segments;
- a piecewise linear cost function representing fuel consumption, traveled distance, or any other suitable objective;
- a bound Δ on the risk of collision over the complete trajectory;
- collision-avoidance constraints represented as disjunctions of linear constraints for polygonal obstacles on a map (e.g., the obstacles shown in Figure 7-9).

The process through which pSulu is incorporated within CLARK is depicted in Figure 7-10. Given two arbitrary locations of interest A and B on a map with obstacles (e.g., the robot's current position and an interesting site for data collection), pSulu is used to determine the existence of a trajectory with bounded risk of collision in the *continuous*, as opposed to grid-based, space of poses (the method for choosing these risk bounds is explained later in this section). If such a trajectory exists, it is returned as a list of n collision-free trajectory segments. For each one of these segments, the *duration predictor* estimates a nondeterministic duration model for that traversal, depicted in Figure 7-10 as hand-drawn densities for Gaussian and uniform durations. In our implementation, the duration predictor can return any of the uncontrollable duration models presented in Chapter 5: when simulated or real traversal data is available, the duration predictor uses probabilistic models learned

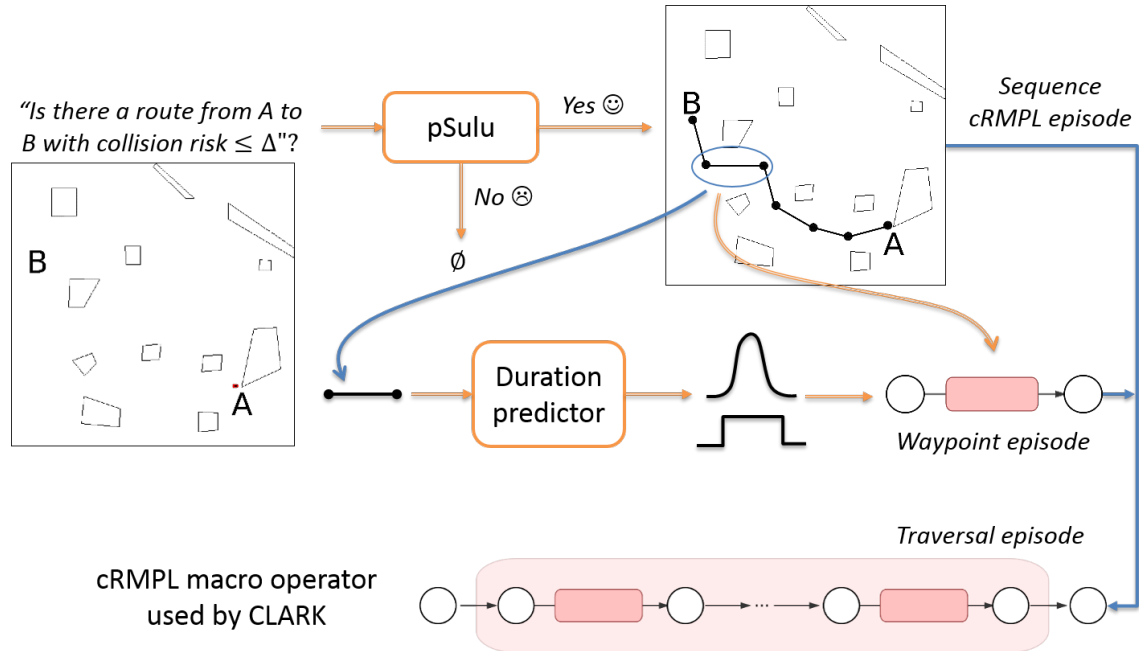


Figure 7-10: Process of converting traversals generated by pSulu into chance-constrained cRMPL episodes.

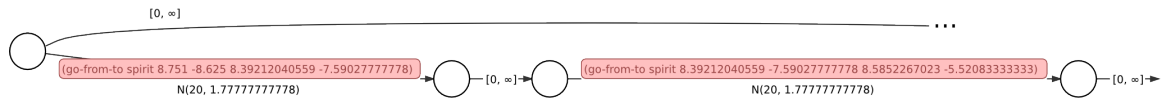


Figure 7-11: First two waypoint episodes for the traversal in Figure 7-9.

from data or specified by experts; when data is not available, the duration predictor returns set-bounded (STNU-like) duration models. The start and end poses of the segment, combined with the uncontrollable duration generated by the duration predictor, are combined to form a *waypoint cRMPL episode*, which are composed using cRMPL’s `sequence` operator to generate a chance-constrained (with chance constraint Δ) composite episode representing the whole traversal. Figure 7-11 shows the first two waypoint episodes for the traversal in Figure 7-9. In our demonstrations, waypoint episodes are dispatched by Enterprise’s hardware interface using an implementation of RRT* [Karaman et al., 2011] to avoid smaller scale obstacles that are not taken into account in the map given to pSulu.

The last aspect of pSulu’s integration within CLARK is how risk bounds $0 < \Delta < 0.5$ are chosen². Figures 7-12 and 7-13 show how the quality of a path, as mea-

²The mathematical model in pSulu restricts the risk bound to be $0 < \Delta < 0.5$ for convexity

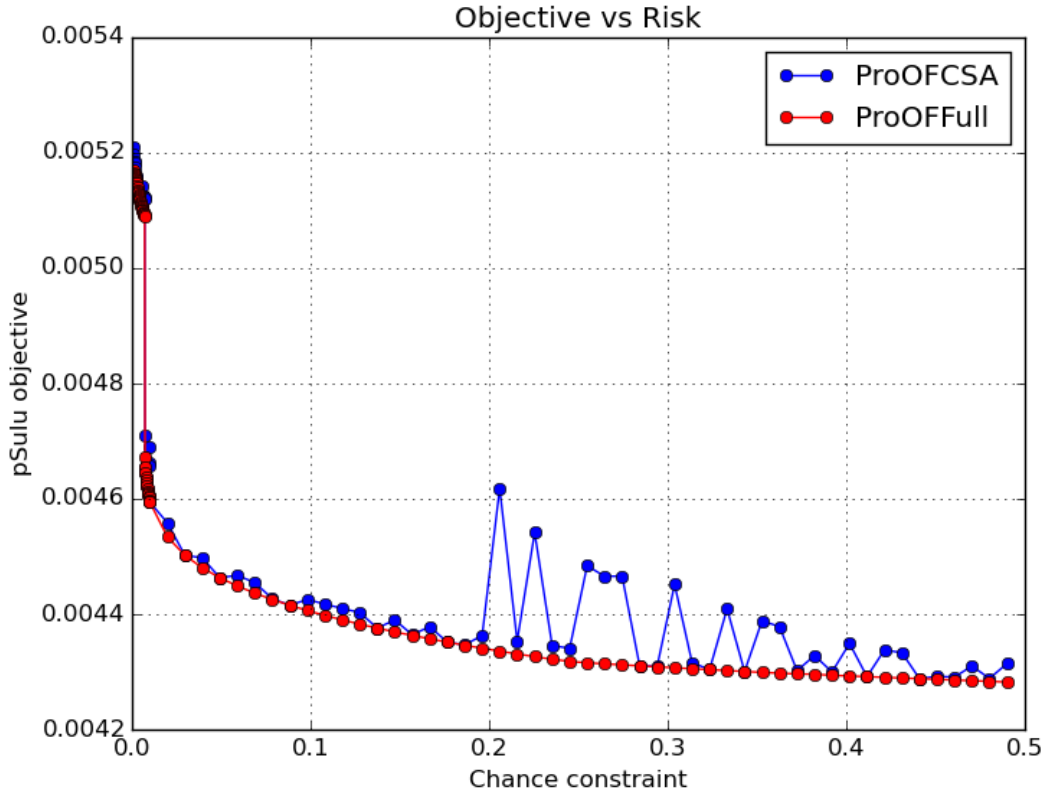


Figure 7-12: Solution quality as a function of the chance constraint Δ for traversals between the two locations in Figure 7-9. The ProOFFull model contains the complete set of constraints for collision avoidance in pSulu, while ProOFFCSA is an approximation of ProOFFull that can be computed faster.

sured by pSulu’s objective function, varies as a function of Δ for the two locations shown in Figure 7-9. In these figures, the labels ProOFFull and ProOFFCSA denote different mathematical models used in the implementation of pSulu: ProOFFull contains the complete set of collision avoidance constraints described in [Arantes et al., 2016b], while ProOFFCSA is a faster, but approximate, model based on the *customized approach* in [Blackmore et al., 2011].

The non-convex nature of the path planning problem around obstacles causes the quality of pSulu’s solution to change discontinuously with Δ , as highlighted in Figure 7-13. This can be intuitively understood by noticing that, despite the fact that path planning is performed on a continuous space in Figure 7-9, there are discrete

reasons.

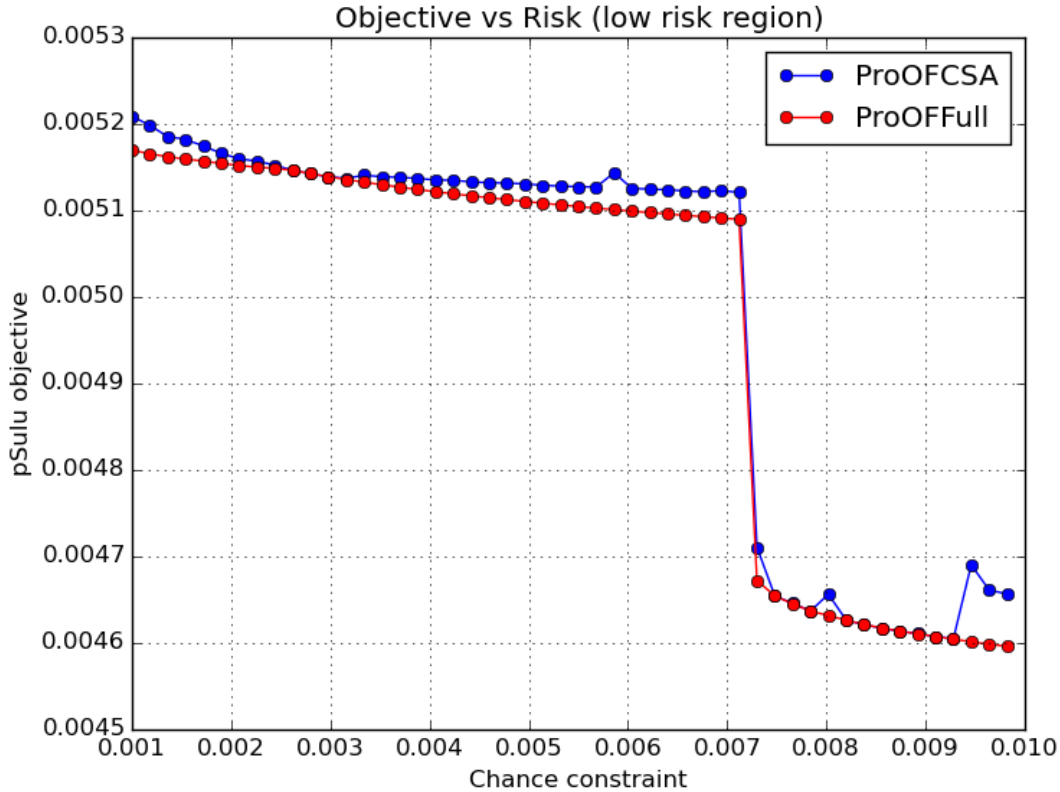


Figure 7-13: A closer look at the low risk portion of Figure 7-12.

“families” of very similar trajectories going around specific sides of the polygonal obstacles. Therefore, for small values of the risk bound Δ , pSulu is forced to take a low quality path that tries to stay as far away as possible from obstacles, while a higher value of Δ allows pSulu to improve the trajectory by taking “shortcuts” that cut through the map. We also notice from Figures 7-12 and 7-13 that, far away from these discontinuities, the quality of the objective remains virtually unchanged, thus forming plateaus.

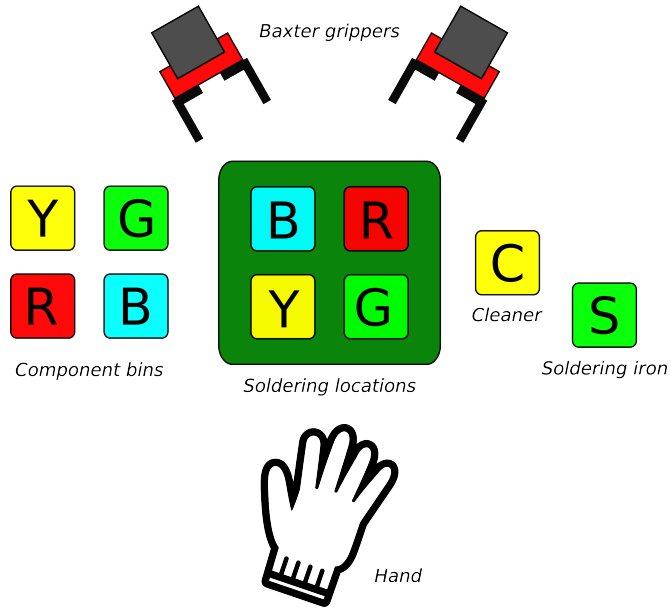
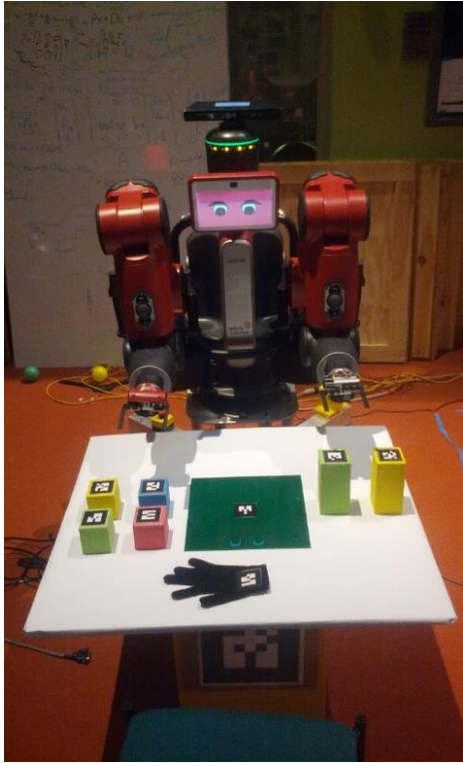
Since CLARK’s CC-POMDP models can only handle discrete actions, we approximate the range of possible values of Δ for traversals between pairs of map sites by discrete bounds $\Delta_1, \dots, \Delta_m$, where Δ_i is taken at the left extremum (lower risk) of the i -th plateau of the objective *versus* Δ curve, where a plateau is defined as a range of values of Δ for which the variation of the objective is less than some user-specified threshold. For a given map and site locations, our experiments performed this dis-

cretization as a preprocessing step for the construction of CC-POMDP models.

7.2 Collaborative manufacturing

Collaborative human-robot manufacturing is a tremendous source of inspiration for this thesis. As motivated at the beginning of Chapter 1, modern manufacturing environments are steadily moving away from a complete separation between humans and robots, towards scenarios where there is synergistic and dynamic collaboration between them. On the one hand, such integration has great potential to boost productivity by leveraging the highly complementary skills of joint human-robot teams [Kaipa et al., 2015]. On the other hand, it raises significant concerns related to safety and reliability: from a human’s perspective, robots must be trustworthy partners that act safely and effectively in an environment that may be constantly changing; from a robot’s perspective, humans are somewhat unpredictable entities that can serve as great sources of help and also disturbances, requiring constant adaptation and strong reliance on sensor information.

The collaborative manufacturing test bed where CLARK was tested is shown in Figure 7-14. It consists of a Baxter robot from Rethink RoboticsTM working in tandem with a human partner to complete a mock electronic component assembly (ECA) task, henceforth referred to as the *ECA scenario* (Figure 7-14b). The goal in the ECA scenario is to correctly place and solder up to four electronic components, represented by small cubes of different colors, on the flat acrylic “circuit board” sitting at the center of the table. In order to solder a component at its correct location on the circuit board, it is required that either the human or the Baxter use the *cleaner* (elongated yellow box) to get rid of any contaminants at that location. Once a location is clean, the corresponding component can be placed and soldered, with the additional requirement that the soldering iron be used by the human. Electronic components are assumed to start at the *component bins* on the left side of Figure 7-14b, while the soldering iron and the cleaner start on the right side. The black-and-white tags seen in Figure 7-14a are used for real-time 3D object tracking using off-the-shelf webcams,



(a) Picture of the ECA scenario with elements not at their designated positions.

(b) Diagram of the ECA scenario showing the designated positions for different elements.

Figure 7-14: Baxter and pieces of a mock electronic component assembly task constituting the *ECA scenario* for collaborative manufacturing demonstrations. Small boxes represent electronic components; the elongated yellow box is a circuit board cleaner; and the elongated green box is a soldering iron.

which look down towards the table. The 3D object pose estimates generated by the camera system, combined with the sensor information coming from the Baxter, are given as inputs to LCARS [Lane, 2016], a hybrid estimator of discrete Boolean predicates (e.g., “block on table” and “hand holding blue block”) used by Pike to monitor plan execution (see Figure 7-7).

Despite the simplicity of the task being performed in the ECA scenario, the attempt to model all different ways in which the human and the environment can interfere with the task, combined with the requirement that the task be executed under time pressure and temporal uncertainty, makes it challenging for CLARK to compute risk-bounded conditional temporal plans. Therefore, in this section, we explore the use of cRMPL programs as expert human guidance for the generation

of risk-bounded conditional temporal plans in collaborative manufacturing domains. For that purpose, we start by focusing on the collaborative pick-and-place portion of the ECA scenario shown in Figure 7-15, in which the Baxter has to constantly adapt to its human coworker while trying to move the components from the left side of the table to their desired locations on the circuit board. We choose to start our analysis by focusing on this subset because it is simple to be written in cRMPL, but the complexity of the solution still grows exponentially with the number of blocks.



Figure 7-15: Simple collaborative pick-and-place task between the Baxter and a human coworker.

The cRMPL control program for the situation in Figure 7-15 is shown in Figure 7-16. It depends on calls to subprocedures shown in Figures 7-17 and 7-18, which also generate cRMPL episodes as their outputs. The English interpretation of the program in Figure 7-16 is the following:

“Start by asking the human whether they would like to start executing the task. If they say no, the task is finished. Otherwise, observe the environment and determine whether the human has made any modifications to it. If yes, record the modification. If not, choose a block to move to its goal

```

def collaborative_pick_and_place(blocks,time_window=-1,dur_dict=None):
    """
    Nominal case, where the robot observes what the human has completed and chooses
    a suitable action.
    """
    agent='Baxter'
    manip='BaxterRight'
    prog = RMPyL(name='run()')
    prog *= prog.sequence(say('Should I start?'),
                          prog.observe({'name':'ask-human',
                                        'ctype':'probabilistic',
                                        'domain':['YES','NO'],
                                        'probability':[0.9,0.1]},
                                        observe_decide_act(prog,blocks,manip,agent,dur_dict),
                                        say('All done!')))
    if time_window>0.0:
        prog.add_overall_temporal_constraint(ctype='controllable',lb=0.0,ub=time_window)
    return prog

```

Figure 7-16: Top level function defining the cRMPL control program for the collaborative pick-and-place task.

```

def observe_decide_act(prog,blocks,manip,agent,dur_dict=None):
    """Models the process of observing what the human has accomplished, and
    choosing a block to pick and place."""
    if len(blocks)>0:
        #Human helped with one of the blocks
        human_help = [observe_decide_act(prog,[ob for ob in blocks if ob!=b],manip,agent) for b in blocks]
        #No help from the human
        no_human_help = prog.sequence(
            prog.decide(
                {'name':'block-choice',
                 'domain':blocks,
                 'utility':range(len(blocks))},
                *[prog.sequence(
                    pick_and_place_block(prog,b,b+'Bin',b+'Target',manip,agent,dur_dict),
                    observe_decide_act(prog,[ob for ob in blocks if ob!=b],manip,agent)) for b in blocks]))
        #All episodes
        all_episodes = human_help+[no_human_help]

        #Observe each one of the blocks
        observations=blocks+['none']
        say_text = 'Checking if the human moved %s components or %s'%(','.join(observations[:-1]),observations[-1])
        return prog.sequence(
            say(say_text),
            prog.observe({'name':'observe-human-%d'%(len(blocks)),
                        'ctype':'probabilistic',
                        'domain':observations,
                        'probability':([0.3/len(blocks)]*len(blocks))+[0.7]},
                        *all_episodes))
    else:
        return say('All done!')

```

Figure 7-17: Recursive function modeling the process of the Baxter observing whether the human modified the environment, choosing the next block to move, and repeating the process until no more blocks are left.

position that has not been moved yet. Repeat this process until there are no more blocks to move.”

```

def say(text):
    """Utters message."""
    return Episode(action=('say \"%s\"'%text).lower())

def pick_and_place_block(prog,block,pick_loc,place_loc,manip,agent,dur_dict=None):
    """Picks a block and places it somewhere."""
    obj=block+'Component'
    duration = dur_dict if dur_dict != None else {'ctype':'controllable','lb':0.0,'ub':float('inf')}
    return prog.sequence(say('Going to pick %s'%obj),
                        Episode(action=('pick %s %s %s %s'%
                                       (obj,manip,pick_loc,agent)).lower(),
                               duration=duration),
                        Episode(action=('place %s %s %s %s'%
                                       (obj,manip,place_loc,agent)).lower(),
                               duration=duration))

```

Figure 7-18: The say activity is primitive for the Baxter, and causes the string passed as an argument to be read by a text-to-speech module. The pick_and_place_block activity, on the other hand, is represented by a composite sequence episode composed of the primitive activities pick and place.

The video at http://mers.csail.mit.edu/video-files/AFOSR/baxter_crmp1_explained.mp4 shows the aforementioned example being dispatched in real time on physical hardware by the CLARK executive for the particular case in which just the red and green blocks must be moved to their appropriate locations. An intuition for understanding how the cRMPL program in Figure 7-16 is dispatched is to consider the execution policy as a sequence of if statements representing the conditions in the program. However, the cRMPL program also causes the CLARK executive to reason over timing constraints, otherwise difficult to capture with conventional control structures such as if statements.

Blocks	Activities	Events	Decisions	Observations	CD	UD
1	8	22	1	2	29	2
2	33	88	5	6	117	12
3	198	522	31	32	695	78
4	1583	4164	249	250	5545	632

Table 7.1: Number of elements involved in the scheduling of the collaborative pick-and-place task as a function of the number of blocks. *Decisions* are controllable choices; *observations* are probabilistic uncontrollable choices; CD are controllable durations represented as simple temporal constraints; and UD are uncontrollable probabilistic durations.

Table 7.1 shows the number of elements involved in the scheduling of the collaborative pick-and-place task as a function of the number of blocks being manipulated.

These numbers are obtained by unraveling the cRMPL program in Figure 7-16 into a PTPN (see Appendix C), and counting the relevant elements. Figure 7-19 shows the outcome of this unraveling for the case where there are two blocks.

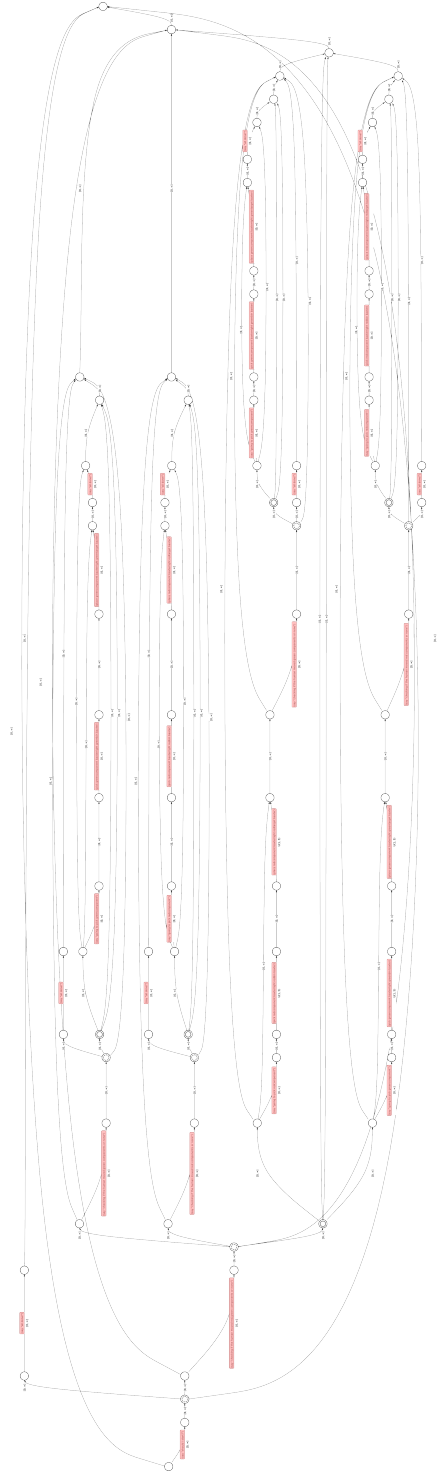


Figure 7-19: PTPN for the collaborative pick-and-place task featuring two blocks.

As indicated by the presence of observation nodes in Table 7.1, execution policies for the cRMPL program in Figure 7-16 are instances of *conditional temporal plans*, precisely the type of problem that CLARK is designed to generate. Therefore, our first goal was to assess the effectiveness of CLARK’s mapping from cRMPL execution to CC-POMDP (Section 4.5.2), combined with the probabilistic scheduling algorithms described in Chapters 5 and 6, relative to the state of the art.

If given a PTPN with controllable and uncontrollable choices and asked to “compile it”, the Pike dispatcher [Levine and Williams, 2014] (the same used as part of the CLARK executive in Figure 7-7) leverages the efficient constraint-labeling scheme introduced by Drake [Conrad et al., 2009, Conrad, 2010, Conrad and Williams, 2011] in order to determine the existence of a feasible schedule. Due to implementation details, Pike is limited to verifying the temporal consistency of PTPN’s containing only controllable temporal durations. Furthermore, if given a PTPN with controllable choices, Pike will try to verify the temporal feasibility of all possible assignments to them. This is in contrast with CLARK’s CC-POMDP strategy, which halts the search for an execution policy the moment it can verify that its current candidate is optimal and temporally feasible within risk bounds. Therefore, in order to perform a fair comparison with Pike, we wrote a version of the cRMPL program in Figure 7-16 without the robot’s decision of which block to manipulate (it is forced to pick the first one in alphabetical order), and with uncontrollable durations converted into controllable ones. For this simplified cRMPL program, the unraveling into a PTPN generates the numbers shown in Table 7.2.

Blocks	Activities	Events	Observations	Constraints
1	8	20	2	27
2	24	58	5	81
3	94	224	18	317
4	466	1106	87	1571

Table 7.2: Number of elements in the PTPN’s given to Pike, which contain no decisions (controllable choices) and only simple temporal constraints (controllable durations). Refer to the numbers in Table 7.1 for a comparison of relative complexity.

The results for our first test comparing CLARK’s CC-POMDP compilation strat-

Blocks	CLARK		Pike	
	CD	UD	CD	UD
1	0.03 s	0.03 s	0.50 s	<i>NA</i>
2	0.20 s	0.30 s	0.50 s	<i>NA</i>
3	2.97 s	3.04 s	<i>TO</i>	<i>NA</i>
4	414.14 s	418.63 s	<i>TO</i>	<i>NA</i>

Table 7.3: Performance comparison between CLARK and Pike in the collaborative pick-and-place scenario. The complexity of the scheduling problems for CLARK and Pike are described, respectively, in Tables 7.1 and 7.2. The CD columns are used for cRMPL programs containing only controllable durations, while UD columns are for programs containing both controllable and uncontrollable (probabilistic) durations. We use *NA* to denote that Pike cannot handle uncontrollable durations, and *TO* to represent a compilation timeout (ran beyond 1 hour without returning a result). Numbers are averages over ten runs.

egy with Pike’s constraint labeling are summarized in Table 7.3, with the complexity of the scheduling problems for CLARK and Pike described, respectively, in Tables 7.1 and 7.2. In these tests, the total execution time was not constrained, so a feasible schedule always existed. Pike verified the temporal consistency of its input PTPN using Drake’s labeled All Pairs Shortest Path algorithm, while CLARK leveraged the mapping from cRMPL execution to CC-POMDP from Section 4.5.2, and the algorithms for strong scheduling under uncertainty described in Chapters 5 and 6. Note that, while the PTPN’s given to Pike had no decisions, CLARK had to incrementally unravel the cRMPL program in Figure 7-16, assigning controllable choices and branching on uncontrollable ones, while using the aforementioned probabilistic scheduling algorithms to check for temporal consistency along the way.

The results for 1 and 2 blocks in Table 7.3 indicate that, as expected, CLARK and Pike have comparable performance for conditional temporal plans involving a small number of constraints and choices (the higher number for Pike is explained by a small communication overhead caused by ROS before returning a solution). However, in the transition from 2 to 3 blocks, where the complexity of the problem increases significantly for both CLARK and Pike, we see that CLARK only takes a

few seconds to find a conditional temporal plan with a feasible schedule, while Pike times out at 1 hour without returning a solution. The increase in complexity is even more dramatic for CLARK in the transition from 3 to 4 blocks, where the runtime increases by two orders of magnitude. Note, however, that this is consistent with the exponential behavior observed in Chapter 6 for strong conditional scheduling, which CLARK has to repeat many times as it incrementally constructs the execution policy for the cRMPL program.

In Table 7.3, the inclusion of probabilistic durations does not seem to have a significant impact on search performance, as indicated by the CD and UD columns under CLARK. This observation is consistent with two important facts: first, the PARIS algorithm in Chapter 5 leverages linear programming and an efficient commercial solver to handle both probabilistic and fully controllable scheduling; second, the lack of a temporal constraint limiting execution duration allows feasible schedules for probabilistic temporal networks to be found without the need for “squeezings”. Figure 7-20 shows the impact of time windows on CLARK’s performance: for overly constrained temporal problems (narrow time window), it is easy to determine the infeasibility of the schedule. As the time window widens and causes the scheduling risk to drop, we observe an increase in the search effort, but it tends to stabilize as the problem becomes temporally unconstrained (wide time window).

It is worthwhile to draw a parallel between these results and the performance of Murphy [Effinger, 2012], a pRMPL executive that greatly inspired this thesis. Unlike CLARK, which is currently restricted to the generation of strong risk-bounded schedules by leveraging the probabilistic scheduling algorithms from Chapters 5 and 6, Murphy is able to generate dynamic schedules for pRMPL programs by pursuing a time discretization strategy, in which continuous time is represented by discrete increments Δt , which are then directly embedded into the discrete state space of an MDP.

Due to this time discretization requirement, the complexity of the MDP solved by Murphy grows exponentially as the scheduling resolution is increased, i.e., Δt is shrunk. For the *Wounded Soldier Example* in [Effinger, 2012], a scenario modeled

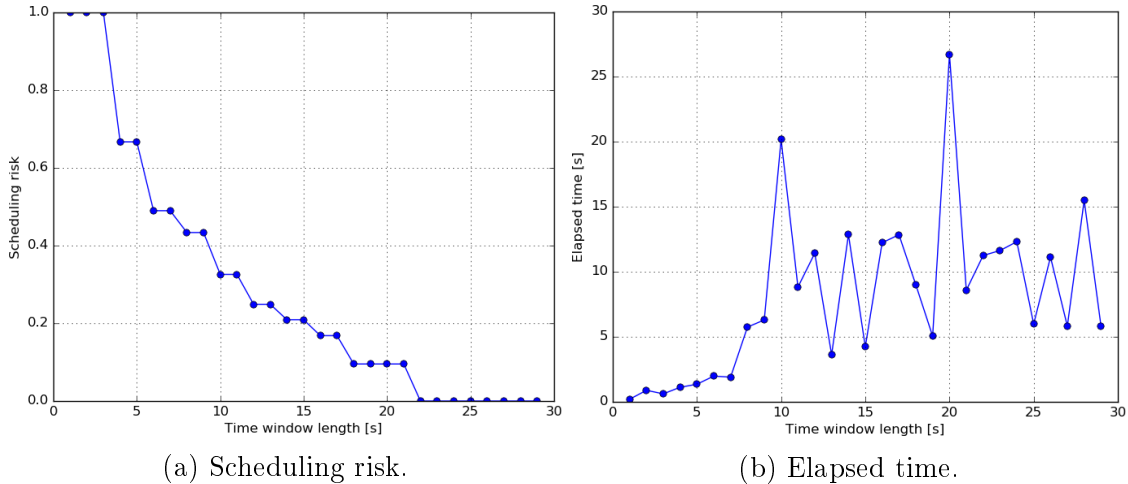


Figure 7-20: Scheduling risk and time to compute a temporally feasible execution of the collaborative pick-and-place task with 3 blocks, as a function of the width of the time window.

with a pRMPL program featuring 5 activities, 2 observations, and 1 decision, the search for an execution policy could take several minutes up to hours for resolutions Δt less than 10 seconds. CLARK, on the other hand, can compute execution policies for cRMPL programs containing thousands of activities and hundreds of choices (see Tables 7.1 and 7.3) through a combination of RAO*'s heuristic forward search and performing probabilistic scheduling by reasoning over *continuous-time* temporal networks. Even though this is not exactly a fair comparison, given that the dynamic schedules generated by Murphy are harder to compute than the risk-bounded strong schedules currently supported by CLARK, the improved scalability of the results in Table 7.3 suggests that CLARK's approach of representing scheduling problems as networks of temporal constraints over continuous time, and leveraging efficient constraint checkers to test for the existence of a feasible schedule, has greater potential of being able to scale to the complexity of real-world problems. In an effort to extend CLARK to the generation of dynamic schedules leveraging a continuous time representation, we sought to incorporate the state-of-the-art dynamic scheduling algorithm for CSTNU's described in [Cimatti et al., 2016a]. In a private communication with one of the authors of the aforementioned work on April 9th, 2016, we were told that "for performance reasons, in the next release the maximum number of conditions [con-

ditions are observations] in each CSTNU will be limited to 26". Even though 26 is about one order of magnitude less conditions than CLARK can currently handle for conditional temporal plans with strong schedules, it seems a significant improvement over the size of the conditional temporal problems that can be handled through temporal discretization in a reasonable amount of time. Unfortunately, we did not have enough time to complete the integration of the algorithm in [Cimatti et al., 2016a] or a probabilistic extension thereof within CLARK, but suggest it as key future work in Section 8.2.

7.3 Data retrieval missions

In addition to collaborative human-robot manufacturing, *data retrieval missions* are another application domain that can greatly benefit from risk-aware autonomy. For instance, autonomous agents can be employed to explore interesting sites currently out of reach for humans, such as Earth's deep ocean waters and regions beneath its polar ice caps; visit other celestial bodies in our solar system; and automate the process of collecting information on the environments that surround us, such as urban traffic and farm crops. In this context, this section discusses how CLARK was used as part of the Resilient Space Systems (RSS) project between JPL, Caltech, WHOI, and MIT, and how CLARK's risk-aware temporal planning capabilities enabled the achievement of RSS' midyear review goals last spring. To the best of our knowledge, CLARK is the first planner capable of handling the RSS demonstration scenario, which requires joint handling of probabilistic scheduling constraints and path planning with uncertain dynamics.

In close connection with this thesis' goal, the RSS project aims at developing autonomous agents that can reason quantitatively about uncertainty and mission risk, and incorporate notions of safety to their decision-making while dynamically executing tasks in partially known, and potentially hazardous, environments. Towards this goal, the technology demonstration that took place as part of RSS' midyear review on April 2016 incorporated the CLARK executive and Enterprise (Figure 7-8) as part of

the Resilient Spacecraft Executive (RSE) architecture [McGhan et al., 2015, McGhan and Murray, 2015] diagrammatically shown in Figure 7-21.

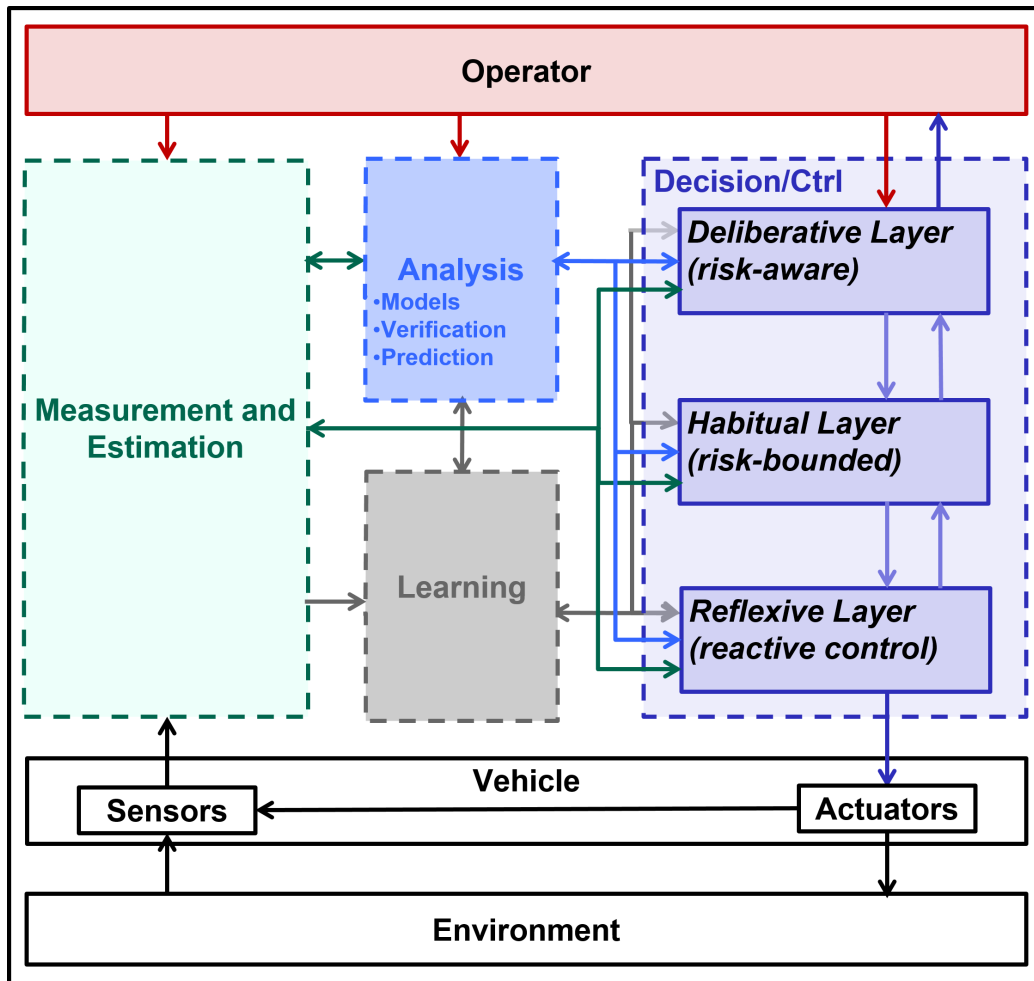


Figure 7-21: Resilient Spacecraft Executive (RSE) architecture. In the RSS demonstration, the role of the risk-aware *deliberative layer* was performed by the CLARK executive within Enterprise (Figure 7-8). The diagram is a courtesy of Catharine McGhan and Tiago Vaquero.

In Figure 7-21, the *deliberative layer* consists of “deliberative” reasoning behavior used to make decisions and perform risk-aware plan execution. In this demonstration, the role of the deliberative layer is played by the CLARK executive within Enterprise (Figure 7-8). At a more concrete level of abstraction, the *habitual layer* consists of “habitual” behaviors that are performed by rote once learned through repetition, such as trajectory planning. Finally, and further down the abstraction hierarchy, the *reflexive layer* consists of “reflexive” low-level behaviors, such as leveraging PID

controllers to align a Mars rover to trajectory segments computed by the habitual layer, while making sure not to bump against previously unseen obstacles. The left side of Figure 7-21 shows the *state measurement* block, where raw sensor data is processed, and which includes analysis and learning capabilities. In the remainder of this section, we focus on CLARK as part of the deliberative layer.

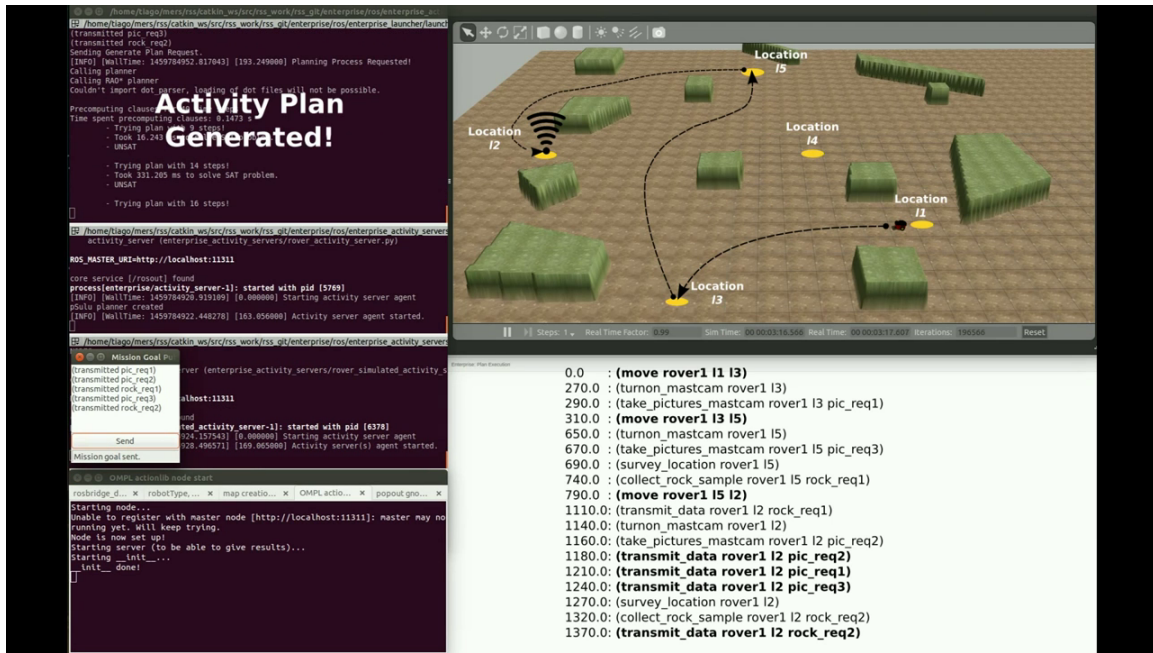


Figure 7-22: Snapshot of the RSS demonstration environment.

A snapshot of the demonstration environment running in the Gazebo simulator [Koenig and Howard, 2004] is shown in Figure 7-22, and a video of the demonstration can be found at http://mers.csail.mit.edu/video-files/rss/Resilient_Space_Systems_Midyear_Review_April_5th_2016.mp4. In a map with obstacles containing five locations of interest (1 through 5), a Mars rover is tasked with taking pictures at locations 2, 3, and 5, as well as collecting rock samples from two out of three potential collection sites at locations 2, 4, and 5. The rover is equipped with two cameras: *mastcam* is a high resolution camera specifically designed to take panoramic pictures, while *hazcam* is a lower resolution camera whose purpose is to perform visual detection of unanticipated obstacles on the rover’s path. However, if *mastcam* is unavailable, we allow *hazcam* to be used to take pictures of map locations due to the lack of a better option. Upon completion of its data collection,

the rover is required to drive to either location 2 or location 4 in order to relay its information to an orbiting satellite, which is visible from these locations within a limited time window $[50, 2200]$ measured in seconds from the start of the mission. There are six types of durative actions available to the rover, which are summarized in Table 7.4: (I) perform a traversal between two locations; (II) activate a camera, which gets automatically deactivated after use; (III) use an active camera to take a picture; (IV) survey an area, which confirms or refutes the presence of a desired rock sample; (V) collect a rock that has been detected by a survey operation; and (VI) transmit a request (picture or rock sample analysis) to the satellite, which can only be done during the period of time in which the satellite is visible. Appendix D contains the PDDL model used in this demonstration, which was augmented with the uncontrollable temporal durations in Table 7.4 according to the procedure described in Section 7.1.1.

In addition to CLARK, tBurton [Wang and Williams, 2015b, Wang, 2015], POPF [Coles et al., 2010], and OPTIC [Benton et al., 2012] were considered as temporal planning options. Except for CLARK, the most important limiting factor for the other planners was their inability to handle uncontrollable temporal durations. Since COLIN and OPTIC are capable of handling planning with continuous resources, we considered following [Ono et al., 2012a] and modeling risk as a “resource” consumed by different actions, but the nonlinear nature of the “squeezings” of probabilistic temporal constraints precluded this option. Even the temporal planner in [Cimatti et al., 2015], which adapts COLIN [Coles et al., 2009, Coles et al., 2012] to handle strong scheduling with STNU constraints, could not be used, due to its inability to handle risk-aware scheduling.

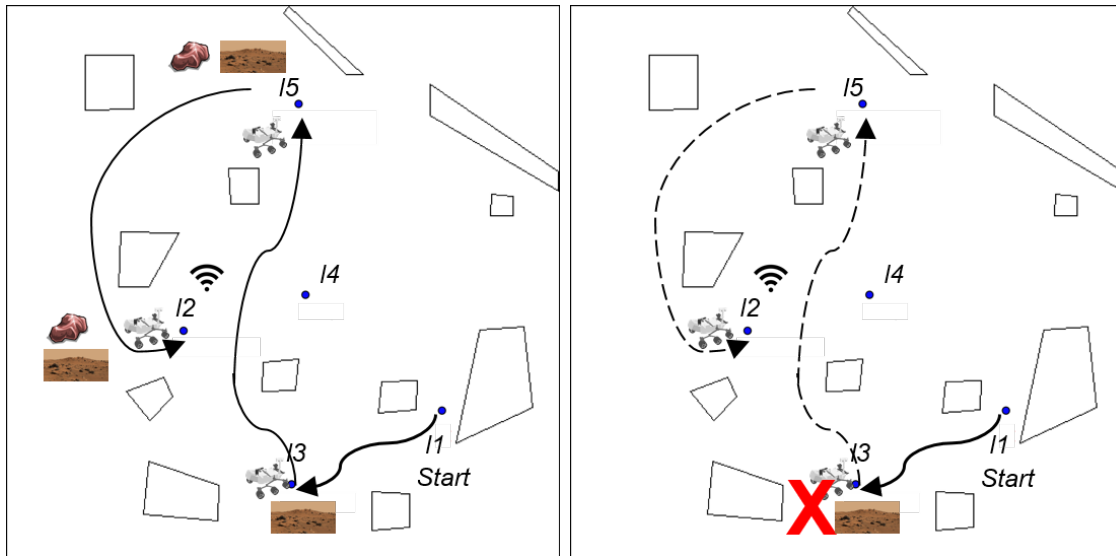
Since the RSS scenario had been originally designed with unconditional temporal planners in mind, one of its goals was to demonstrate *reactive robustness through replanning*. On the one hand, unlike Sections 3.5 and 7.2, this demonstration requirement has the downside of not exercising CLARK’s ability to achieve robustness through *conditional probabilistic planning*, which allows an agent’s current decisions to be influenced by anticipated future execution scenarios (e.g., picking up an un-

Action	Duration type	Representation
Take picture with mastcam	set-bounded	$u[5,20]$
Take picture with hazcam	set-bounded	$u[5,20]$
Collect rock sample	set-bounded	$u[10,50]$
Survey location	set-bounded	$u[10,50]$
Turn mastcam on	uniform	$U(5,20)$
Turn hazcam on	uniform	$U(5,20)$
Transmit data	uniform	$U(10,30)$
Move $l_1 \leftrightarrow l_2$	Gaussian	$N(270,100)$
Move $l_1 \leftrightarrow l_3$	Gaussian	$N(220,100)$
Move $l_1 \leftrightarrow l_4$	Gaussian	$N(220,100)$
Move $l_1 \leftrightarrow l_5$	Gaussian	$N(270,100)$
Move $l_2 \leftrightarrow l_3$	Gaussian	$N(250,100)$
Move $l_2 \leftrightarrow l_4$	Gaussian	$N(230,100)$
Move $l_2 \leftrightarrow l_5$	Gaussian	$N(270,100)$
Move $l_3 \leftrightarrow l_4$	Gaussian	$N(240,100)$
Move $l_3 \leftrightarrow l_5$	Gaussian	$N(290,100)$
Move $l_4 \leftrightarrow l_5$	Gaussian	$N(237,100)$

Table 7.4: Duration models used in the RSS demonstration. Set-bounded durations $u[a, b]$ are those found in STNU’s and represent random variables that take values within the interval $[a, b]$ with *unknown* probability distribution. Similar to set-bounded durations, a uniform duration $U(a, b)$ also takes values in the interval $[a, b]$, but with known density $(b - a)^{-1}$ anywhere within the interval. Finally, a Gaussian duration $N(\mu, \sigma^2)$ has mean μ and variance σ^2 .

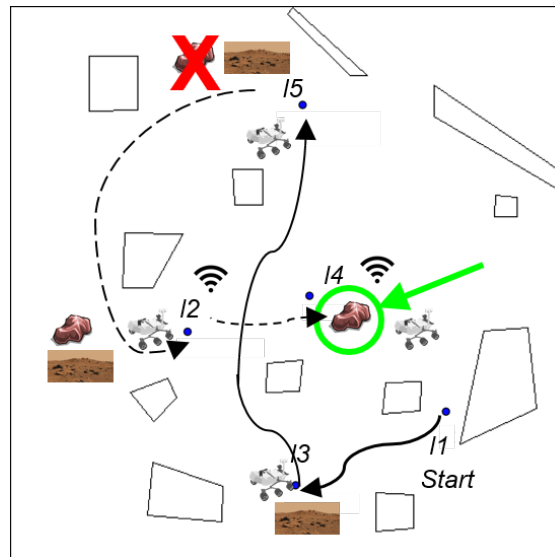
brella if there is a possibility of rain later in the day). On the other hand, by requiring replanning at reactive timescales, the RSS scenario fostered, and empirically demonstrated, CLARK’s ability to quickly generate *emergency contingency* temporal plans, which often take the form of unconditional sequences of actions that seek to protect the agent and its environment from harm [Arantes et al., 2015, Arantes et al., 2016a]. As first mentioned in Section 1.4 and reinforced in Section 7.1.3, being able to monitor execution and react to failure conditions not anticipated by the CC-POMDP model is key to enhancing the CLARK executive’s robustness to external disturbances. This is because planning models, CC-POMDP and otherwise, are always incomplete approximations of reality, thus requiring plan executives for risk-aware autonomous agents to necessarily be ready to handle unmodeled contingencies. Moreover, even restricted to unconditional temporal planning, the RSS scenario still could not be handled by

existing temporal planners, due to its requirement of risk-aware scheduling in the presence of set-bounded and probabilistic uncontrollable durations. In Section 7.3.1, we extend the model used in the RSS demonstration to handle the risk from path planning collisions.



(a) First stage: nominal temporal plan.

(b) Second stage: mastcam fault.



(c) Third stage: rock-not-found fault.

Figure 7-23: Different stage of the RSS demonstration available at the aforementioned video link. The “nominal temporal plan” shown in 7-23a is the same one depicted on the bottom right quadrant of Figure 7-22. The figures are a courtesy of Tiago Vaquero and Catharine McGhan.

Figure 7-23 depicts three important stages in the RSS demonstration video referred

to earlier in this section. First, during the “nominal plan” stage in Figure 7-23a, CLARK decides to visit locations 3, 2, and 5 in this order; collect rock samples at 5 and 2; and transmit its data to the satellite at location 2. However, the moment the rover arrives at location 3, a mastcam fault is manually injected into the system, preventing the rover from being able to take a picture at that location and causing Pike to trigger replanning within the CLARK executive (Figures 7-7 and 7-8). Taking into account the elapsed time since the beginning of execution, the time windows for communication with the satellite are updated, and CLARK generates a new plan from the rover’s current state that uses hazcam as a replacement for mastcam (Figure 7-23b), while preserving all the other plan actions (but not their schedule). Once the rover arrives at location 5, a fault related to the survey failing to detect the rock sample is injected into the system, causing Pike to trigger replanning once again. A graphical representation of the new plan generated by CLARK is shown in Figure 7-23c, in which the rover chooses to collect the rock sample at location 4, and to transmit data at both locations 2 and 4.

Start time	Activity
0.0 :	(move rover1 l1 l3)
270.0 :	(turnon_mastcam rover1 l3)
290.0 :	(take_pictures_mastcam rover1 l3 pic_req1)
310.0 :	(move rover1 l3 l5)
650.0 :	(turnon_mastcam rover1 l5)
670.0 :	(take_pictures_mastcam rover1 l5 pic_req3)
690.0 :	(survey_location rover1 l5)
740.0 :	(collect_rock_sample rover1 l5 rock_req1)
790.0 :	(move rover1 l5 l2)
1110.0 :	(transmit_data rover1 l2 rock_req1)
1140.0 :	(turnon_mastcam rover1 l2)
1160.0 :	(take_pictures_mastcam rover1 l2 pic_req2)
1180.0 :	(transmit_data rover1 l2 pic_req2)
1210.0 :	(transmit_data rover1 l2 pic_req1)
1240.0 :	(transmit_data rover1 l2 pic_req3)
1270.0 :	(survey_location rover1 l2)
1320.0 :	(collect_rock_sample rover1 l2 rock_req2)
1370.0 :	(transmit_data rover1 l2 rock_req2)

Table 7.5: Possible schedule for the risk-aware “nominal” temporal plan generated by CLARK for the RSS demonstration scenario.

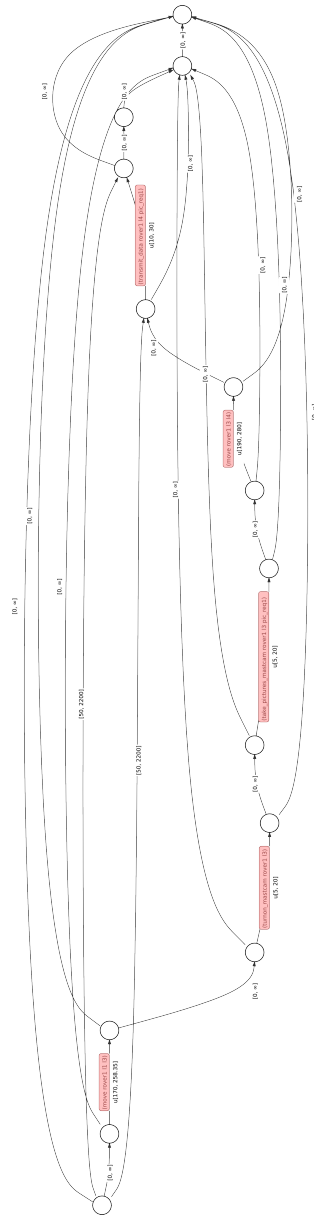


Figure 7-24: PTPN generated by CLARK and sent to Pike for the simple case where only a picture at location 3 is needed. The actual scheduling of activities is performed by Pike in real time as it dispatches the plan.

Figure 7-24 shows what the PTPN that CLARK sends to Pike looks like for the simple case where only a picture at location 3 is needed³. Instead of generating a fixed schedule for the plan, which is standard practice for temporal planners handling controllable temporal durations, the PTPN in Figure 7-24 gives Pike the flexibility to dispatch the plan in real time, thus improving the CLARK executive's robustness

³A PTPN for the complete scenario cannot legibly fit on the page.

to temporal disturbances. Table 7.5 shows one possible schedule for the actions in the risk-aware “nominal” temporal plan generated by CLARK for the complete RSS demonstration scenario.

The durative actions in CLARK’s CC-POMDP are constructed from the PDDL model in Appendix D and Table 7.4 by following the procedure in Section 7.1.1. States in RAO*’s search graph contain references to the temporal constraints entailed by the current policy, i.e., the uncontrollable durations of all actions that are currently part of the policy, plus the time window constraints associated with the transmit actions. Combined with the strong scheduling constraints (5.2) (page 156), these form the set \mathcal{C} of constraints in the CC-POMDP. The PARIS scheduler from Chapter 5 serves as both the execution risk heuristic for RAO* and the constraint violation function c_v in the CC-POMDP: for intermediate search states, it computes an *admissible* estimate of the scheduling risk bound by *minimizing* it over the *current set* of entailed temporal constraints. At terminal search states, it computes this risk bound for the final set of temporal constraints in the policy and returns it as the effective mission scheduling risk, which then gets propagated by RAO*’s backup operations up to the root node of the policy. A chance constraint $er(b_0, C|\pi) \leq \Delta = 0.1\%$ is used to limit the scheduling risk over the complete execution, and the goal in the CC-POMDP is to minimize the number of plan actions.

Since repeated evaluations of probabilistic temporal consistency is a costly operation, we implemented an approach similar to [Ivankovic et al., 2014] and guided RAO*’s search by computing sets of “helpful actions” and minimum length relaxed (non-temporal) plans using a SAT-based subplanner inspired by SatPlan [Kautz and Selman, 2006]. We chose a SAT-based encoding of the relaxed planning problem because it allowed easy generation of the set of all helpful actions at a state: after each call to the SAT-based subplanner, we recorded the first action in the plan and added the negation of its Boolean variable to the planning clauses, repeating this process until the subplanner returned no solution. States stemming from helpful actions at their parents were given preference in RAO*’s expansion, while other were labeled for “delayed” expansion.

Goal combinations	PL	TTS [s]	ER	MR-MS [s]	CC-MS [s]
g_1	5.00	0.19	1.15E-06	630.00	608.35
g_2	4.00	0.11	5.73E-07	390.00	378.05
g_3	5.00	0.18	1.15E-06	710.00	688.35
g_4	4.00	0.12	5.73E-07	400.00	388.05
g_5	4.00	0.13	5.73E-07	400.00	388.05
g_1, g_2	8.00	0.40	1.15E-06	710.00	688.35
g_1, g_3	9.00	0.65	1.72E-06	1070.00	1038.66
g_1, g_4	8.00	0.41	1.15E-06	760.00	738.35
g_1, g_5	8.00	0.41	1.15E-06	760.00	738.35
g_2, g_3	8.00	0.40	1.15E-06	780.00	758.35
g_2, g_4	7.00	0.31	5.73E-07	520.00	508.05
g_2, g_5	7.00	0.31	5.73E-07	520.00	508.05
g_3, g_4	8.00	0.46	1.15E-06	840.00	818.35
g_3, g_5	8.00	0.49	1.15E-06	840.00	818.35
g_4, g_5	8.00	0.47	1.15E-06	810.00	788.35
g_1, g_2, g_3	12.00	1.06	1.72E-06	1140.00	1108.66
g_1, g_2, g_4	11.00	0.84	1.15E-06	840.00	818.35
g_1, g_2, g_5	11.00	0.81	1.15E-06	840.00	818.35
g_1, g_3, g_4	12.00	1.40	1.72E-06	1200.00	1168.66
g_1, g_3, g_5	12.00	1.36	1.72E-06	1200.00	1168.66
g_1, g_4, g_5	12.00	1.63	1.72E-06	1190.00	1158.66
g_2, g_3, g_4	11.00	0.93	1.15E-06	910.00	888.35
g_2, g_3, g_5	11.00	0.97	1.15E-06	910.00	888.35
g_2, g_4, g_5	11.00	2.21	1.15E-06	880.00	858.35
g_3, g_4, g_5	11.00	0.92	1.15E-06	970.00	948.35
g_1, g_2, g_3, g_4	15.00	5.72	1.72E-06	1270.00	1238.66
g_1, g_2, g_3, g_5	15.00	6.02	1.72E-06	1270.00	1238.66
g_1, g_2, g_4, g_5	15.00	9.59	1.72E-06	1260.00	1228.66
g_1, g_3, g_4, g_5	15.00	8.95	1.72E-06	1330.00	1298.66
g_2, g_3, g_4, g_5	14.00	3.20	1.15E-06	1040.00	1018.35
g_1, g_2, g_3, g_4, g_5	18.00	28.39	1.72E-06	1400.00	1368.66

Table 7.6: CLARK’s performance on the RSS demonstration for different combinations of goals. PL is the *Plan Length* (number of actions); TTS is the *Time to Solution*, i.e., the amount of time to generate a temporal plan with a schedule that meets the chance constraint; ER is RAO* *execution risk*, which corresponds to the scheduling risk for these plans; MR-MS is the *Minimum Risk Makespan*, i.e., the temporal span to the temporal plan when scheduling risk is minimized; and CC-MS is the *Chance-Constrained Makespan*, where the risk bound $\Delta = 0.1\%$ is exploited to reduce the total execution time.

Let the different goals in the RSS demonstration scenario be denoted by the following aliases:

- g_1 : (transmitted pic_req1);
- g_2 : (transmitted pic_req2);
- g_3 : (transmitted pic_req3);
- g_4 : (transmitted rock_req1);
- g_5 : (transmitted rock_req2).

With this notation, Table 7.6 shows CLARK’s performance on the RSS demonstration scenario as a function of the combination of goals that must be achieved by the mission. In all test cases, CLARK generated optimal (minimum number of actions) temporal plans handling all the different types of uncontrollable durations in Table 7.4, and respecting the risk bound $\Delta = 0.1\%$. In fact, as shown by the fourth column in Table 7.6, the scheduling risk of the temporal plans are quite far away from the aforementioned risk bound. Therefore, as a post-processing step, temporal plans generated by CLARK with uncontrollable durations (e.g., Figure 7-24) and execution risk $\Delta' < \Delta$ were given back to PARIS so that it could minimize its makespan with a chance constraint $\Delta - \Delta'$ (see Section 5.4.5). In other words, we used the remaining “risk budget” to optimize a secondary plan objective (plan makespan) that was not part of the original objective in the CC-POMDP. The results of this optimization are shown on the last column of Table 7.6, where we see that all plan makespans were reduced.

Finally, the third column (time to solution) in Table 7.6 indicates that CLARK achieved its goal of providing risk-aware temporal planning with expressive temporal uncertainty models within reactive time scales. This is certainly not a claim that CLARK is at the same level of efficiency as specialized temporal planners for PDDL2.2 and above, which would require a full translation from PDDL2.2 into the CC-POMDP framework, and the incorporation of special optimizations for temporal PDDL planning that neither RAO*, nor its available CC-POMDP models, currently possess. At the same time, to the best of our knowledge, there are currently no temporal planners capable of handling the probabilistic scheduling constraints required

by the RSS demonstration, and our results in Table 7.6 show that CLARK and its CC-POMDP models attain good performance for scenarios of practical interest.

7.3.1 Extending RSS with risk-bounded path planning

The previous discussion of the RSS demonstration does not consider the risk stemming from collisions with map obstacles, and how this risk of collision impacts the rover’s choice of activities and schedule for a mission that must be carried under bounded risk. Therefore, in this section, we extend the CC-POMDP model for the RSS demonstration to handle risk-bounded path planning, as explained in Section 7.1.4.

The CC-POMDP model in this new experimental setting conserves all attributes from the previous RSS demonstration, and incorporates the environment map (Figure 7-23) and vehicle dynamics. States in RAO*’s search graph now contain a mean μ and covariance matrix P representing the Gaussian uncertainty about the rover’s true position on the map, and pSulu is used to check the existence of a path between two regions on the map that can be traversed with bounded risk of collision. Following the procedure in Section 7.1.4, the `move` action in the PDDL model for the RSS demonstration is split into two types of traversal activities: `move-low` is the low risk version of the traversal, with a bound on the risk of collision of 0.01%, while `move-high` tolerates a risk of collision of up to 0.04%. Since a higher risk of collision allows the rover to pursue higher quality paths on the map, preference to `move-high` is given over `move-low`, whenever mission risk permits. As in the previous demonstration, a chance constraint $er(b_0, C|\pi) \leq \Delta = 0.1\%$ is used to bound the risk of the rover colliding with obstacles *or* missing its communication window with the satellite.

An example of a conditional plan generated by CLARK in this new setting is shown in Figure 7-25, where the only goals are to have pictures of locations 3 and 5 (g_1 and g_3). Ellipses represent belief states in the conditional policies, and edges are the probabilities of transitioning between belief states. Since this is a fully observable model, each belief state contains a single particle with probability 1 (thus the reason why *Entropy* is 0 in all of them). In each belief state, fields R , ER , and ERB are,

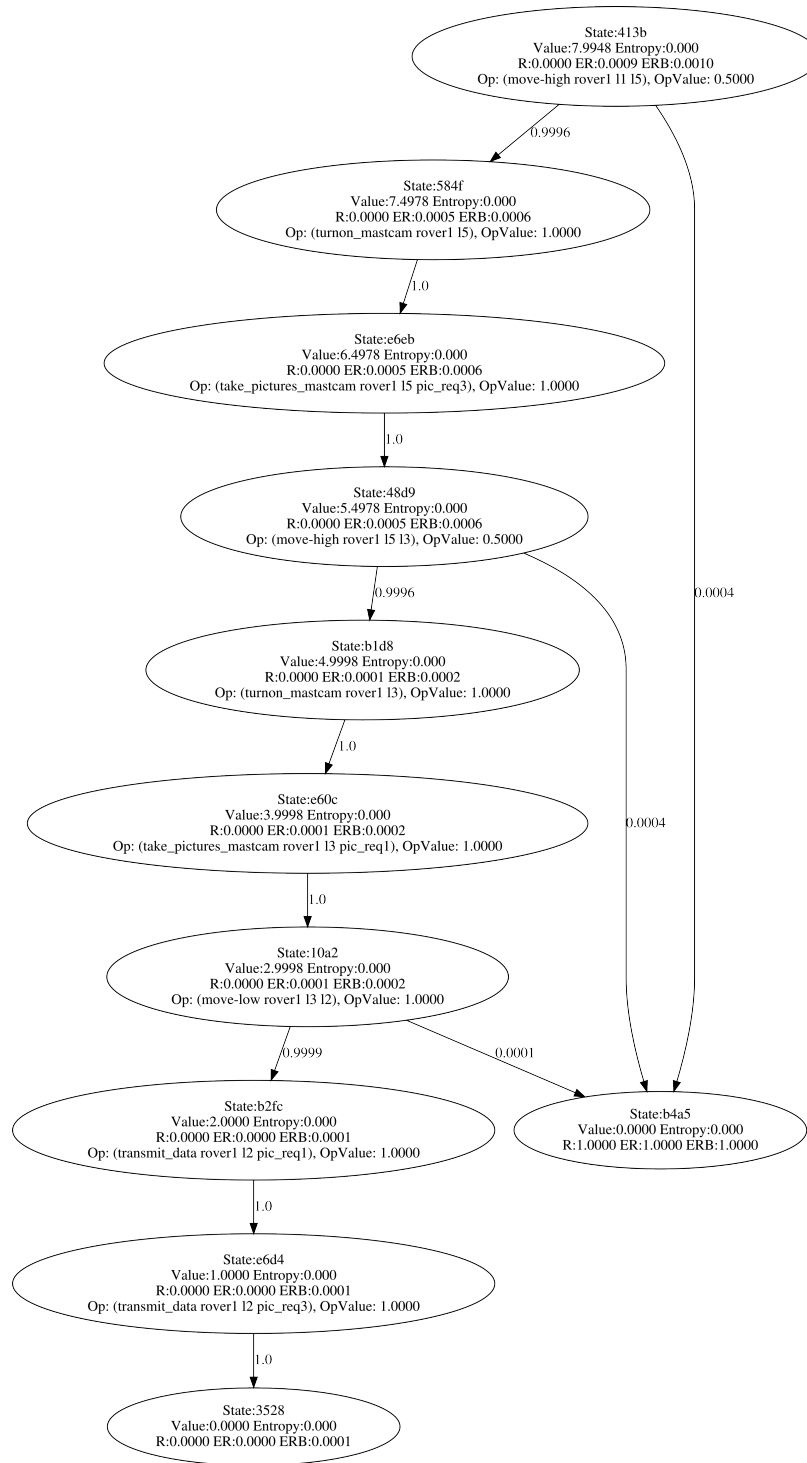


Figure 7-25: CLARK policy for the RSS scenario with picture requests at locations 3 and 5, and two types of move actions with bounded risk of collision: move-low (risk bound of 0.01%) and move-high (risk bound of 0.04%).

respectively, the immediate risk $r_b(b_k)$, the execution risk $er(b_k|\pi)$, and the execution risk bound Δ_k at that belief state. The optimal activity at each belief state is shown as Op (for Operator), and the value of executing this activity at the belief state is shown as $OpValue$ (in this case, the value is a cost).

Starting with the risk bound $er(b_0, C|\pi) \leq \Delta = 0.1\%$ at the root belief, it gets “consumed” by the path planning and scheduling risks according to (3.28) (page 88). However, the scheduling risk is so small compared to the collision risk that only the latter is reflected within the finite precision of the ERB fields in Figure 7-25. Note that `move-high` is used for the first two traversals, but it gets pruned by RAO* due to its excessive risk in the low risk traversal from location 3 to location 2.

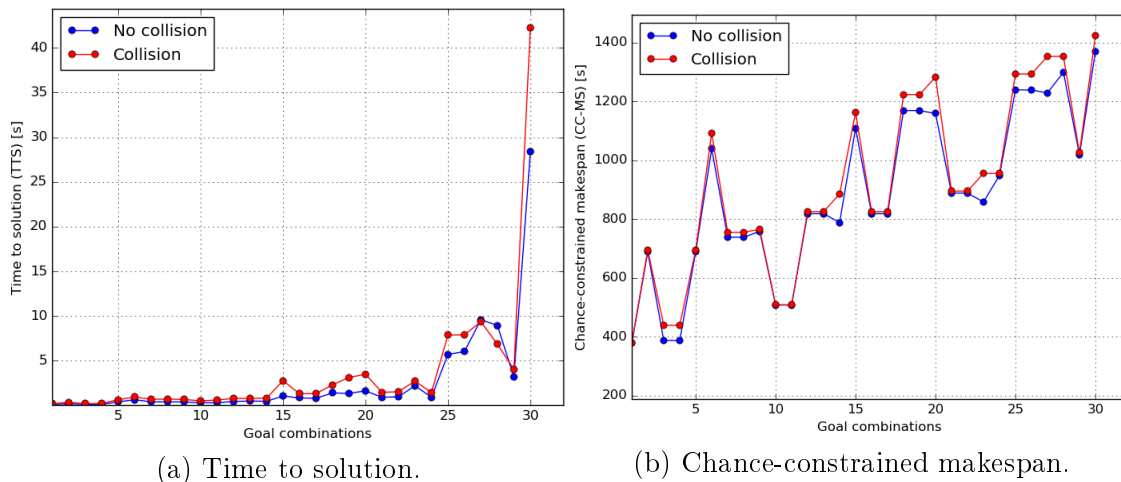


Figure 7-26: Comparison between RSS scenarios with (label *Collision*) and without (label *No collision*) collision handling. The order of goal combinations on the horizontal axes follows the first column of Table 7.6.

The same set of goal combinations from Table 7.6 were evaluated in this new setting of the RSS demonstration, and comparative results between the two settings of the RSS demonstration are shown as plots in Figure 7-26. With these experiments, our goal was to assess whether the inclusion of path planning constraints would have a significant impact on CLARK’s performance, and whether the extra risk from collisions would affect activity scheduling.

Since pSulu is not an algorithm developed in the context of this thesis, the times in Figure 7-26a do not take into account the preprocessing step that computes the

existence of routes between pairs of locations on the map at different levels of risk, which are used by CLARK as a lookup table. However, Figure 7-26a does take into account the overhead generated by the duplication of `move` actions, and the added complexity of RAO* constructing policies with probabilistic branching. From the observation of the two curves in Figure 7-26a, we conclude that the more expressive CC-POMDP planning model that takes path planning risk into account did not incur significant overhead, except on the last example. The reason behind the latter is that our SAT-based subplanner for helpful actions incorporates no information about risk, therefore causing RAO* to backtrack a few times on the grounds of chance constraint infeasibility. Nevertheless, despite the added complexity, CLARK is still able to compute a feasible risk-bounded temporal plan in less than a minute for the complete RSS scenario taking collision risk into account.

Our last experiments concerning the impact of collision risk on activity scheduling are shown in Figure 7-26b and confirm our expectations: since the chance constraint $er(b_0, C|\pi) \leq \Delta = 0.1\%$ is being used to bound the probability of joint scheduling and path planning constraints, a higher risk “consumption” by collision constraints leaves less room for improvement of the mission schedule, as indicated by the *Collision* curve always being above *No collision* in Figure 7-26b.

7.4 Conclusions

This chapter revisits the CLARK system first introduced in Section 1.4, and builds upon concepts presented in previous chapters to provide a detailed, holistic view of the different components of CLARK’s architecture (Figure 7-1), the CLARK executive (Figure 7-7), and the integration of the latter within Enterprise (Figure 7-8). Moreover, Sections 7.2 and 7.3 present experimental results in application domains where CLARK receives inputs, respectively, as cRMPL programs and augmented PDDL models, thus complementing the experimental results from Chapter 3, where inputs to CLARK are given directly as CC-POMDP’s. Therefore, with this chapter, this thesis addresses all forms of inputs and outputs shown in Figure 7-1.

In view of the results presented so far, the next chapter summarizes this thesis' main contributions, and briefly discusses some of the many potential avenues of future research that could stem from this work.

Chapter 8

Conclusions

“None of my inventions came by accident. I see a worthwhile need to be met and I make trial after trial until it comes. What it boils down to is one per cent inspiration and ninety-nine per cent perspiration.”

Thomas A. Edison, 1929.

The first half of 2016 brought with it two sharply distinct moments for the interaction between artificial intelligence and humankind. While the world¹ watched in awe as AlphaGo [Silver et al., 2016] convincingly beat Lee Sedol, considered one of the best Go players of all time, in a historic five-game match akin to Deep Blue’s [Campbell et al., 2002] victory over World Chess Champion Garry Kasparov in 1997, it also witnessed the first fatal accident ruled as being caused by a self-driving vehicle [Vlasic and Boudette, 2016]. As we move towards a reality where autonomous agents are increasingly put in charge of high-stakes mission that often include the protection of human life, matters of safe and trustworthy autonomy come into the spotlight. It is within this context that this thesis places its contributions.

8.1 Summary of contributions

In this thesis, we argued that trusting autonomous agents with safety-critical tasks requires them to develop a keen sensitivity to risk and to incorporate uncertainty

¹Or, at least, the more computationally-inclined portion of it.

into their decision-making. Towards this goal, we proposed that autonomous agents must be able to execute risk-bounded conditional plans while operating under time pressure and other types of constraints. By being conditional, the policy allows the agent’s choice of activity to depend on its current belief about the true state of the world. Moreover, ensuring safety through bounds on mission risk allows such conditional plans to move away from the conservatism of risk-minimizing approaches, while keeping the autonomous agent strictly within acceptable safety levels.

As a formalism for conditional risk-bounded planning under uncertainty, this thesis proposed Chance-Constrained Partially Observable Markov Decision Processes (CC-POMDP’s) in Chapter 3. The key feature of CC-POMDP’s is a novel dynamic measure of mission safety in terms of *execution risk*, which addresses shortcomings in the literature related to the treatment of risks as unit costs or a “resource” that is monotonically spent throughout execution. In order to solve CC-POMDP’s, we introduced Risk-bounded AO* (RAO*) in Chapter 3, an HFS-based algorithm that searches for solutions to a CC-POMDP by leveraging admissible utility and risk heuristics to simultaneously guide the search and perform early pruning of overly-risky policy branches.

In an effort to facilitate the specification of risk-bounded behavior by human modelers, this thesis also presented the Chance-constrained Reactive Model-based Programming Language (cRMPL) in Chapter 4, a novel variant of RMPL [Williams et al., 2003, Ingham, 2003, Effinger, 2012] that incorporates chance constraints as part of its syntax. By defining the execution semantics of cRMPL in terms of CC-POMDP’s and using RAO* to compute optimal, risk-bounded execution policies, we were able to exploit the language’s hierarchical structure and steer away from the intractability of previous approaches that required the explicit unraveling of all possible execution traces. From a software engineering standpoint, our choice to implement cRMPL as an extension of Python made it easy to integrate cRMPL within the popular Robot Operating System (ROS) [Quigley et al., 2009] framework for robotics applications, a key capability for modern agent programming languages [Ziafati et al., 2012].

Influenced by Planning Modulo Theories (PMT) [Gregory et al., 2012], semantic attachment [Dornhege et al., 2012], and [Ivankovic et al., 2014], which promote a synergistic combination of state-of-the-art methods for symbolic planning and constraint satisfaction, Chapter 5 developed PARIS, the current fastest algorithm for risk-aware scheduling of Probabilistic Simple Temporal Networks with Uncertainty (PSTNU’s), a temporal reasoning formalism that subsumes both PSTN’s and STNU’s. In Chapter 6, PARIS is used in the development of the first risk-aware conditional scheduling algorithm for Probabilistic Temporal Plan Networks (PTPN’s) featuring PSTNU constraints.

The different tools and algorithms developed in the context of this thesis were combined to form the Constrained Planning for Autonomy with Risk (CLARK) system, a risk-aware conditional planning system that can generate chance-constrained, dynamic temporal plans for autonomous agents that must operate under uncertainty. In addition to experimental results involving the different components of CLARK in Chapters 3, 5, and 6, several demonstrations in Chapter 7 empirically showed that CLARK achieves this thesis’ goal of generating safe, conditional plans over rich sets of mission constraints, while working in tandem with other building blocks of resilient autonomous systems.

8.2 Future work

Extend RAO* to compute infinite horizon, risk-bounded policies

The temporal nature of the applications that this thesis is concerned about, in which autonomous agents must complete their tasks under time pressure and temporal uncertainty, was the motivation behind our focus on finite horizon policies in this thesis’ development of RAO*. However, the modeling power of CC-POMDP’s extends much beyond risk-aware temporal applications. In situations where an agent’s actions in the environment are not bound by any particular deadline, infinite horizon policies may be desired as an approximation of time-independent behavior.

An important matter concerns the execution risk recursion

$$er(b_k|\pi) = r_b(b_k) + (1 - r_b(b_k)) \sum_{o_{k+1}} \Pr^{sa}(o_{k+1}|\pi(b_k), b_k) er(b_{k+1}^{sa}|\pi) \quad (3.21)$$

from Chapter 3 as the horizon $h \rightarrow \infty$. Since (3.21) can be interpreted as a discounted Bellman backup equation with finite costs $r_b(b_k) \in [0, 1]$ and discount factor $(1 - r_b(b_k)) \in [0, 1]$, we conclude that $er(b_k|\pi)$ converges to a point within $[0, 1]$ even as $k \rightarrow \infty$. However, it is very easy to think of situations in which $er(b_k|\pi) \rightarrow 1$ as $k \rightarrow \infty$, which means that no risk-bounded policies from belief b exist. For instance, think of a robot moving down a hallway that, at each step, may collide with the walls with probability p . In this case, $er(b_k|\pi) = 1 - (1 - p)^{(h-k)}$, which tends to 1 as $h \rightarrow \infty$ for all values of k . This may a trivial example, but it raises two important questions: under which conditions can one impose non-trivial bounds $0 \leq \Delta < 1$ on (3.21) and still be able to compute a chance-constrained policy? Can this be determined without having to exhaust the search space? We conjecture that such questions could potentially be answered efficiently by extending the notion of a Strongly Connected Component (SCC) [Tarjan, 1972] to a “Strongly Safe Connected Component” (SSaCC), and modifying Topological Value Iteration (TVI) [Dai et al., 2011] to compute (3.21) within these SSaCC’s. These computations could be incorporated into an LAO*-like [Hansen and Zilberstein, 2001] algorithm to produce RLAO*².

Handle partially-enumerated beliefs and observations, and sampling

This is the topic of Appendix A, in which we investigate possible ways of extending RAO*’s risk-aware search to planning domains that are intractably large to be fully enumerated, while retaining strict safety guarantees. The idea is that large groups of unlikely particles in a belief state, as well as unlikely observation branches on the policy search graph, could be appropriately lumped, respectively, into “macro” particles and observations, whose expansion would be delayed to the future. Generating these sets of unlikely particles and observations could either be achieved through effi-

²Or some other more suitably-named algorithm.

cient best-first enumeration algorithms [Timmons, 2013], or through sampling, as in (L)RTDP [Barto et al., 1995, Bonet and Geffner, 2003]. In the latter case, Hoeffding’s inequality [Hoeffding, 1963] could provide a guaranteed bound on the probability of the empirical mean falling far from the true expected value.

Support risk-aware, conditional and unconditional dynamic scheduling

In order to perform risk-aware scheduling under temporal uncertainty, this thesis presented PARIS (Chapter 5) and its extension to conditional scheduling (Chapter 6). However, both of these perform *strong scheduling*, i.e., they determine precomputed schedules for controllable temporal events that are guaranteed to be feasible with high probability, regardless of the particular outcomes of uncertainty (both observations and uncontrollable durations) in the plan. Therefore, even though temporal plans generated by CLARK are dynamic in terms of action selection (the choice of action depends on the current belief state), their schedules are static.

Unfortunately, to the best of our knowledge, there are currently no conditional or unconditional risk-aware dynamic scheduling algorithms that can incorporate information about probabilistic observations and temporal durations. As mentioned in Chapter 2, there are several approaches to dynamic scheduling of STNU’s [Morris et al., 2001, Morris and Muscettola, 2005, Morris, 2006, Hunsberger, 2009, Hunsberger, 2010, Hunsberger, 2013, Hunsberger, 2014, Morris, 2014], and some very recent attempts at extending the STNU paradigm to conditional scheduling [Combi et al., 2013, Cimatti et al., 2014, Cimatti et al., 2016b, Cimatti et al., 2016a], where both temporal durations and uncontrollable choices have set-bounded, non-probabilistic uncertainty. However, unlike existing strong schedulers for PSTN’s and PSTNU’s [Tsamardinos, 2002, Fang et al., 2014, Wang and Williams, 2015a, Santana et al., 2016c], none of these dynamic schedulers incorporate any notion of scheduling risk arising from probabilistic durations and real-time observations.

Risk-aware, conflict-directed search

It is known that conflict-directed search methods [Stallman and Sussman, 1977, De Kleer and Williams, 1987, De Kleer et al., 1992, Williams and Ragno, 2007] provide superior performance in large, constrained search spaces when compared with chronological backtracking, as seen in Chapter 6. However, since the constraint violation function c_v in CC-POMDP’s (Definition 3.1) does not output any conflicts, RAO*’s mechanism for revising overly risky partial policies is not conflict-directed. Inspired by Conflict-Directed A^* [Williams and Ragno, 2007] (CDA*), a potentially worthwhile extension of the results in this thesis would be to develop risk-aware conflict extraction methods that could be used to propose the unimaginatively-named CD-RAO*. Examples of such risk-aware conflict extraction methods are developed in [Wang, 2013, Wang and Williams, 2015a] in the context of strong scheduling for PSTN’s.

Extend the range of input languages to CLARK

The current interface to CLARK is written mostly in Python, thus facilitating its integration within ROS-based robotic applications and software systems that leverage Python’s extensive set of scientific programming libraries. There is also a Common Lisp version, which is intended to be integrated with an ongoing implementation of cRMPL as an extension of Common Lisp. However, to improve CLARK’s usage as a standalone chance-constrained conditional planner, it would be helpful to implement interfaces that could generate language-agnostic CC-POMDP descriptions from existing probabilistic planning languages, such as PPDDL [Younes and Littman, 2004], RDDDL [Sanner, 2010], and PomdpX [APPL, 2014], among others. Another option would be to extend cRMPL and its future improvements to be able to specify CC-POMDP models directly.

Hardware deployment combined with model learning

All components of CLARK are model-based, i.e., they operate on the assumption that their input models, including probability distributions, are precise abstractions of the real world. Moreover, all experiments involving CLARK so far have considered CC-POMDP models with probabilistic transition and observation models that do not depend on the continuous state of the Plant. However, in support of this thesis, we have shown that the incorporation of continuous state information into discrete probabilistic models can be beneficial, and provide in [Santana et al., 2015] a data-driven algorithm for the unsupervised learning of Probabilistic Hybrid Automata (PHA) models. Therefore, in order to allow CLARK to control systems whose dynamical models change as they age, it would be interesting to combine the algorithm in [Santana et al., 2015] to the CLARK executive architecture shown in Figure 1-10.

Appendix A

Extending RAO* to partially-enumerated beliefs and policies

In this appendix, we address challenges related to the handling of large belief states and policy branching in Chapter 3. More specifically, we are concerned with the computation of

$$\bar{b}(s_{k+1}) = \sum_{s_k} T(s_k, a_k, s_{k+1}) \hat{b}(s_k), \quad (\text{A.1})$$

$$\hat{b}(s_{k+1}) = \frac{1}{\eta} O(s_{k+1}, o_{k+1}) \bar{b}(s_{k+1}), \quad (\text{A.2})$$

$$\Pr(o_{k+1} | o_{1:k}, a_{0:k}) = \sum_{s_{k+1}} O(s_{k+1}, o_{k+1}) \bar{b}(s_{k+1}), \quad (\text{A.3})$$

in situations in which the size of belief spaces and the number of observations at nodes render exact computations intractable. Our goal here is to understand how approximations of (A.1)-(A.3) can impact execution risk and policy optimality.

A.1 Partial enumeration of belief states

Henceforth, let the term *particle* denote a state-probability with nonzero probability. The set of all particles at some belief state b_k will be denoted by $A_p(b_k) = \{s_k : b(s_k) \neq 0\}$.

Due to the potential sheer size of \mathcal{S} , the number of elements in $A_p(b_k)$ might be intractably large to enumerate. Hence, our proposed approach to attain scalability is to focus our attention on a subset of particles concentrating most of the probability in $A_p(b_k)$. Let us call this subset of particles concentrating most of the probability the *Enumerated Particles* (EP) set, denoted by $E_p(b_k)$. The remaining particles composing the belief state are called *Lumped Particles* (LP) and denoted by $L_p(b_k)$. We say that the remaining particles are “lumped” because they are not explicitly enumerated by the algorithm. Instead, they are lumped together into meta particles representing sets of unenumerated particles (“clouds” in Figure A-1). A visual depiction of these sets can be found in Figure A-1, in which belief states correspond to layers on the tree. Squares on a belief layer correspond to particles in E_p , while the set of cloud nodes represent the lumped particles in L_p . A belief state featuring a non-empty L_p is called a partially-enumerated belief state (PEBS).

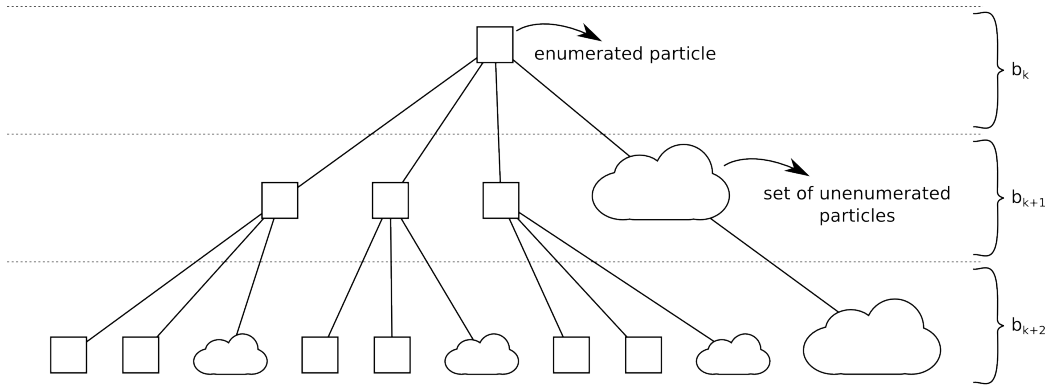


Figure A-1: Example PEBS. Squares on the tree represent enumerated particles with non-zero probabilities; clouds represent unenumerated (lumped) particles; and layers represent belief states.

The intuition behind E_p and L_p is very simple. If we were to deal with the complete belief states in our search for a policy, the computational load would quickly

become intractable as the number of particles grows quickly in (A.1)-(A.2). Hence, in order to be able to compute a chance-constrained policy, we will choose to only consider a subset of true belief states, namely E_p , and *ignore* the particles in L_p . Ignoring particles is clearly something “risky” to do, but it is feasible due to the leeway provided by chance constraints. Hence, as long as we can show that the risk of ignoring particles meets our chance constraints, the policy is still guaranteed to be safe. Another consequence of ignoring particles is the potential loss of optimality, as measured by the objective function that the policy is trying to optimize.

For the moment, let us assume that there exists an algorithm, such as Bones [Timmons, 2013], capable of computing $E_p(b_k)$ for some particular belief state b_k . It is worthwhile to stress an important fact that might not be obvious at first glance: given a belief state b_k , we might have the same state s_k represented in both the enumerated and the lumped particle sets. In other words, we can always write

$$b(s_k) = b_e(s_k) + b_l(s_k), \tag{A.4}$$

where $b(s_k)$ is the true probability (if we could perfectly compute as probabilities) of state s_k in belief state b_k ; $b_e(s_k)$ is our estimate of $b(s_k)$; and $b_l(s_k)$ is the non-negative error in our estimate, which corresponds to the probability sum of all cloud particles containing s_k . The source of the error term $b_l(s_k)$ is made evident in Section A.1.1.

Finally, given that probabilities in a belief state must always sum to 1, it is easy to see that

$$\Pr(L_p(b_k)) = 1 - \Pr(E_p(b_k)) = 1 - \sum_{s_k \in E_p(b_k)} b_e(s_k). \tag{A.5}$$

Hence, even if we just propagate the particles in $E_p(b_k)$, we can still recover the total probability within $L_p(b_k)$ from (A.5).

A.1.1 Predicting PEBS's

We start by considering the problem of propagating PEBS's forward according to the state transition model. From (A.1), we get

$$\begin{aligned}
\bar{b}(s_{k+1}|a_k) &= \sum_{s_k \in A_p(\hat{b}_k)} T(s_k, a_k, s_{k+1}) \hat{b}(s_k), \\
&= \sum_{s_k \in E_p(\hat{b}_k)} T(s_k, a_k, s_{k+1}) \hat{b}_e(s_k) + \sum_{s_k \in L_p(\hat{b}_k)} T(s_k, a_k, s_{k+1}) \hat{b}_l(s_k), \\
&= \bar{b}_e(s_{k+1}|a_k) + \bar{b}_l(s_{k+1}|a_k), \tag{A.6}
\end{aligned}$$

where the summation in (A.6) was split between the enumerated and lumped portion of the belief of each state s_k . The term

$$\bar{b}_e(s_{k+1}|a_k) = \sum_{s_k \in E_p(\hat{b}_k)} T(s_k, a_k, s_{k+1}) \hat{b}_e(s_k), \tag{A.7}$$

which is obtained through forward propagation of $E_p(\hat{b}_k)$ according to the state transition model, can be computed as usual. The other term corresponds to $\bar{b}_l(s_{k+1}|a_k)$ and cannot be computed exactly. This is due to the fact that we neither have access to individual particles within $L_p(\hat{b}_k)$, nor we have an exact value for the error $\hat{b}_l(s_k)$. For a given s_{k+1} , there is no guarantee that there exists no s_k within cloud nodes in \hat{b}_k for which $T(s_k, a_k, s_{k+1}) \neq 0$. In simpler words, there might be some potential parents of s_{k+1} “lost” in the cloud nodes and whose probabilities are not being taken into account when computing $\bar{b}_e(s_{k+1}|a_k)$ in (A.6). Hence, $\bar{b}_e(s_{k+1}|a_k)$ will *always* be an underestimate (lower bound) of the true probability $\bar{b}(s_{k+1}|a_k)$, given that the error term $\bar{b}_l(s_{k+1}|a_k)$ is clearly non-negative.

One simple, but important, consequence of (A.6) is that the probability mass within the set of enumerated particles will always tend to decrease as a result of the prediction step. This can be easily seen from

$$\Pr(E_p(\hat{b}_k)) = \sum_{s_k \in E_p(\hat{b}_k)} \hat{b}_e(s_k) \geq \sum_{s_k \in E_p(\hat{b}_k)} T(s_k, a_k, s_{k+1}) \hat{b}_e(s_k) = \Pr(E_p(\bar{b}_{k+1})). \tag{A.8}$$

Thus, we conclude from (A.5) and (A.8) that $\Pr(L_p(\bar{b}_{k+1}))$ will never decrease when compared to $\Pr(L_p(\hat{b}_k))$.

Risk bounds for predicted PEBS's

We now consider the impact of the underestimates in (A.6) on how we measure risk using

$$r_b(b_k, C) = \sum_{s_k \in S} b(s_k) c_v(s_k, C). \quad (\text{A.9})$$

Using the notation from this section, we can rewrite (A.9) as

$$r_b(\bar{b}_{k+1}) = \sum_{s_{k+1} \in A_p(\bar{b}_{k+1})} \bar{b}(s_{k+1}) c_v(s_{k+1}) = \sum_{s_{k+1} \in A_p(\bar{b}_{k+1})} (\bar{b}_e(s_{k+1}) + \bar{b}_l(s_{k+1})) c_v(s_{k+1}) \quad (\text{A.10})$$

Similar to (A.6), we can only compute the probabilities $\bar{b}(s_{k+1})$ up to some error. Hence, we must find ways to approximate (A.10) so as to get, at least, upper and lower bounds on the risk of a PEBS. For now, let us settle for the simple solution found by replacing $c_v(\cdot)$ for the terms multiplying $\bar{b}_l(s_{k+1})$ by a constant, state-independent probability of violating hard plan constraints. Let c be such a constant. Under this assumption, we can rewrite (A.10) as

$$r_b(\bar{b}_{k+1}) = \sum_{s_{k+1} \in E_p(\bar{b}_{k+1})} \bar{b}_e(s_{k+1}) c_v(s_{k+1}) + c \left(\sum_{s_{k+1} \in L_p(\bar{b}_{k+1})} \bar{b}_l(s_{k+1}) \right), \quad (\text{A.11})$$

where the probability term within parentheses is exactly

$$\Pr(L_p(\bar{b}_{k+1})) = 1 - \sum_{s_{k+1} \in E_p(\bar{b}_{k+1})} \bar{b}_e(s_{k+1}).$$

Let c_v^{\max} and c_v^{\min} be, respectively, state-independent maximal and minimal prob-

abilities of violating constraints in a given domain. In this case, we get the bounds

$$r_b^{up}(\bar{b}_{k+1}) = \sum_{s_{k+1} \in E_p(\bar{b}_{k+1})} \bar{b}_e(s_{k+1})c_v(s_{k+1}) + c_v^{\max} \Pr(L_p(\bar{b}_{k+1})), \quad (\text{A.12})$$

$$r_b^{lo}(\bar{b}_{k+1}) = \sum_{s_{k+1} \in E_p(\bar{b}_{k+1})} \bar{b}_e(s_{k+1})c_v(s_{k+1}) + c_v^{\min} \Pr(L_p(\bar{b}_{k+1})), \quad (\text{A.13})$$

where $r_b^{up}(\bar{b}_{k+1})$ and $r_b^{lo}(\bar{b}_{k+1})$ are upper and lower bounds, respectively, for $r_b(\bar{b}_{k+1})$. The simplest choices for these bounds are $c_v^{\max} = 1$ and $c_v^{\min} = 0$, which should hold regardless of the planning domain.

A.1.2 Updating partially-enumerated beliefs

The previous section analyzes how a partially-enumerated belief state impacts (A.1) and its corresponding measures of risk. In this section, we focus on determining how (A.2) is affected by partial enumeration of both predicted belief states and observations.

Once again, we will focus on computing updated beliefs for particles in $E_p(\hat{b}_{k+1})$ and leave the probability of $L_p(\hat{b}_{k+1})$ to (A.5). From (A.2) and (A.6), we get

$$\hat{b}(s_{k+1}|a_k, o_{k+1}) \propto O(s_{k+1}, o_{k+1})\bar{b}_e(s_{k+1}|a_k) + O(s_{k+1}, o_{k+1})\bar{b}_l(s_{k+1}|a_k), \quad (\text{A.14})$$

Different from (A.6), the probability in (A.14) can only be computed up to some normalization factor z_{k+1} , called the partition function. In the case of (A.14), we have $z_{k+1} = \Pr(o_{k+1}|a_{1:k}, o_{1:k})$. There are two important shortcomings related to (A.14). First, the probability $\bar{b}_l(s_{k+1}|a_k)$ is unknown, which prevents us from even computing the true value of the right-hand side of (A.14). Second, even if we could compensate for this error, it is not sufficient to only determine probability terms up to some constant factor. This is because we are trying to offer hard guarantees on the risk of execution, so we need to know exactly - or at least be able to bound - the probability of different outcomes when executing a policy.

Given the unity sum constraint for probability distributions, we can use (A.14) to

write the partition function as

$$z_{k+1} = \sum_{s_{k+1} \in A_p(\bar{b}_{k+1})} O(s_{k+1}, o_{k+1}) \bar{b}_e(s_{k+1}|a_k) + O(s_{k+1}, o_{k+1}) \bar{b}_l(s_{k+1}|a_k), \quad (\text{A.15})$$

which, once again, cannot be computed exactly due to the partial enumeration of belief states. Let us assume, for the moment, that we have a way of computing (A.15). Under this assumption, we can rewrite (A.14) as

$$\begin{aligned} \hat{b}(s_{k+1}|a_k, o_{k+1}) &= \frac{O(s_{k+1}, o_{k+1}) \bar{b}_e(s_{k+1}|a_k)}{z_{k+1}} + \frac{O(s_{k+1}, o_{k+1}) \bar{b}_l(s_{k+1}|a_k)}{z_{k+1}}, \\ &= \hat{b}_e(s_{k+1}|o_{k+1}, a_k) + \hat{b}_l(s_{k+1}|o_{k+1}, a_k), \end{aligned} \quad (\text{A.16})$$

which looks a lot like (A.6). Simple bounds for (A.15) can be written as

$$z_{k+1}^{up} = \sum_{s_{k+1} \in E_p(\bar{b}_{k+1})} O(s_{k+1}, o_{k+1}) \bar{b}_e(s_{k+1}|a_k) + \Pr(o_{k+1})^{\max} \Pr(L_p(\bar{b}_{k+1})), \quad (\text{A.17})$$

$$z_{k+1}^{lo} = \sum_{s_{k+1} \in E_p(\bar{b}_{k+1})} O(s_{k+1}, o_{k+1}) \bar{b}_e(s_{k+1}|a_k) + \Pr(o_{k+1})^{\min} \Pr(L_p(\bar{b}_{k+1})), \quad (\text{A.18})$$

where $\Pr(o_{k+1})^{\max}$ and $\Pr(o_{k+1})^{\min}$ are, respectively, upper and lower bounds¹ for the probability $O(s_{k+1}, o_{k+1})$, $\forall s_{k+1}$. Substituting (A.17)-(A.18) in (A.15) gives us, respectively, upper and lower bounds for $\hat{b}_e(s_{k+1}|o_{k+1}, a_k)$ equal to

$$\hat{b}_e^{up}(s_{k+1}|o_{k+1}, a_k) = \frac{O(s_{k+1}, o_{k+1}) \bar{b}_e(s_{k+1}|a_k)}{z_{k+1}^{lo}}, \quad (\text{A.19})$$

$$\hat{b}_e^{lo}(s_{k+1}|o_{k+1}, a_k) = \frac{O(s_{k+1}, o_{k+1}) \bar{b}_e(s_{k+1}|a_k)}{z_{k+1}^{up}}. \quad (\text{A.20})$$

Completing the recursion with Section A.1.1 requires $\hat{b}_e(s_{k+1}|o_{k+1}, a_k)$ to be assigned values, which could in turn be used during the next iteration of (A.6). Due to our inability to compute the partition function z_{k+1} exactly, we have to settle for the bounds (A.19) and (A.20) of the true $\hat{b}_e(s_{k+1}|o_{k+1}, a_k)$. Thus, in order to remain

¹For discrete observations, choosing $\Pr(o_{k+1})^{\max} = 1$ and $\Pr(o_{k+1})^{\min} = 0$ is always guaranteed to be consistent.

consistent with (A.4) and make sure that we do not violate chance constraints, we must make the approximation

$$\hat{b}_e(s_{k+1}|o_{k+1}, a_k) \leftarrow \hat{b}_e^{lo}(s_{k+1}|o_{k+1}, a_k), \forall s_{k+1} \in E_p(\hat{b}_{k+1}), \quad (\text{A.21})$$

which also assigns $\Pr(L_p(\hat{b}_{k+1})) \leftarrow \Pr(L_p(\hat{b}_{k+1}))^{up}$ by moving errors in the term $\hat{b}_e^{lo}(s_{k+1}|o_{k+1}, a_k)$ to the cloud particles in $L_p(\hat{b}_{k+1})$. The reasons behind the need for a lower bound $\hat{b}_e(s_{k+1}|o_{k+1}, a_k)$ are detailed in the following sections, where the computation of utility and risk bounds require the availability of such lower bounds.

Risk bounds for updated PEBS's

Following the same steps as before, we can write the risk $r_b(\hat{b}_{k+1})$ for the updated belief state as

$$r_b(\hat{b}_{k+1}) = \sum_{s_{k+1} \in E_p(\hat{b}_{k+1})} \hat{b}_e(s_{k+1})c_v(s_{k+1}) + \left(\sum_{s_{k+1} \in L_p(\hat{b}_{k+1})} \hat{b}_l(s_{k+1})c_v(s_{k+1}) \right). \quad (\text{A.22})$$

Reality comes to tear our hopes and dreams the moment we realize that *none* of the terms in (A.22) can be computed exactly. This is because updating PEBS superimposes the intractability of (A.15) and the errors introduced by partial enumeration, which is itself due to the intractable size of \mathcal{S} . Similar to Section A.1.1, we now investigate ways to deal with these multiple approximations in order to still be able to compute policies with hard guarantees on risk.

Let c_v^{\max} and c_v^{\min} be as defined in Section A.1.1. Similar to (A.12)-(A.13), we can write the bounds

$$r_b(\hat{b}_{k+1}) \leq r_b^{up}(\hat{b}_{k+1}) = \sum_{s_{k+1} \in E_p(\hat{b}_{k+1})} \hat{b}_e(s_{k+1})c_v(s_{k+1}) + c_v^{\max} \Pr(L_p(\hat{b}_{k+1})), \quad (\text{A.23})$$

$$r_b(\hat{b}_{k+1}) \geq r_b^{lo}(\hat{b}_{k+1}) = \sum_{s_{k+1} \in E_p(\hat{b}_{k+1})} \hat{b}_e(s_{k+1})c_v(s_{k+1}) + c_v^{\min} \Pr(L_p(\hat{b}_{k+1})). \quad (\text{A.24})$$

From the previous subsection, we know that updated belief states cannot be com-

puted exactly. Therefore, the probability terms in (A.23) and (A.24) must be replaced by the bounds in (A.19) and (A.20). A simple differential analysis shows us that

$$\frac{\partial(r_b^{up}(\hat{b}_{k+1}))}{\partial\hat{b}_e(s_{k+1})} = c_v(s_{k+1}) - c_v^{\max} \leq 0, \quad \frac{\partial(r_b^{lo}(\hat{b}_{k+1}))}{\partial\hat{b}_e(s_{k+1})} = c_v(s_{k+1}) - c_v^{\min} \geq 0. \quad (\text{A.25})$$

Hence, both an upper bound for (A.23) and a lower bound for (A.24) are attained at a lower bound $\hat{b}_e^{lo}(s_{k+1})$ of $\hat{b}_e(s_{k+1})$. Using (A.20), we can compute an upper bound for $\Pr(L_p(\hat{b}_{k+1}))$ as

$$\Pr(L_p(\hat{b}_{k+1}))^{up} = 1 - \sum_{s_{k+1} \in E_p(\hat{b}_{k+1})} \hat{b}_e^{lo}(s_{k+1}|o_{k+1}, a_k), \quad (\text{A.26})$$

which can be substituted into (A.23)-(A.24) in order to give the computable bounds

$$r_b^{up}(\hat{b}_{k+1}) = \sum_{s_{k+1} \in E_p(\hat{b}_{k+1})} \hat{b}_e^{lo}(s_{k+1})c_v(s_{k+1}) + c_v^{\max} \Pr(L_p(\hat{b}_{k+1}))^{up}, \quad (\text{A.27})$$

$$r_b^{lo}(\hat{b}_{k+1}) = \sum_{s_{k+1} \in E_p(\hat{b}_{k+1})} \hat{b}_e^{lo}(s_{k+1})c_v(s_{k+1}) + c_v^{\min} \Pr(L_p(\hat{b}_{k+1}))^{up}. \quad (\text{A.28})$$

A.1.3 Computing approximate execution risks

So far, we have only considered the impact that partial enumeration of particles in a belief state has on the point-wise measure of risk $r_b(b_k)$. However, our discussion about chance constraints in Chapter 3 showed that the execution risk

$$er(b_k|\pi) = r_b(b_k) + (1 - r_b(b_k)) \sum_{o_{k+1}} \Pr^{sa}(o_{k+1}|\pi(b_k), b_k) er(b_{k+1}|\pi) \quad (\text{A.29})$$

is more useful for imposing chance constraints on a conditional plan with sensing actions.

This section addresses two challenges associated with the computation of (A.29), namely: the errors caused by the partial enumeration of particles (cloud nodes in Figure A-1); and the potentially intractable number of possible observations o_{k+1} , creating the need to truncate their enumeration as well. We, then, analyze how these

approximations propagate into our estimates of the execution risk $er(b_k|\pi)$. In the following, we develop approximations for $\Pr(o_{k+1}|a_k)$, which can be readily extended to $\Pr(o_{k+1}|a_k)^{sa}$ by replacing predicted PEBS's by *safe* predicted PEBS's.

In our search for approximations for (A.29), we start by investigating the impact that PEBS's have on (A.3) by writing

$$\begin{aligned} \Pr(o_{k+1}|a_k) &= \sum_{s_{k+1} \in A_p(\bar{b}_{k+1})} O(s_{k+1}, o_{k+1}) \bar{b}(s_{k+1}|a_k), \\ &= \sum_{s_{k+1} \in E_p(\bar{b}_{k+1})} O(s_{k+1}, o_{k+1}) \bar{b}_e(s_{k+1}|a_k) \\ &\quad + \left(\sum_{s_{k+1} \in L_p(\bar{b}_{k+1})} O(s_{k+1}, o_{k+1}) \bar{b}_l(s_{k+1}|a_k) \right). \end{aligned} \quad (\text{A.30})$$

Similar to what has been seen before, individual belief values for the terms within parentheses in (A.30) are not available due to the partial enumeration of belief states. Therefore, our goal is to come up with upper and lower bounds for (A.30) that could be subsequently used in approximating execution risks. Following the exact same procedure as in (A.11)-(A.13), we get the bounds

$$\Pr(o_{k+1}|a_k)^{up} = \sum_{s_{k+1} \in E_p(\bar{b}_{k+1})} O(s_{k+1}, o_{k+1}) \bar{b}_e(s_{k+1}|a_k) + \Pr(o_{k+1})^{\max} \Pr(L_p(\bar{b}_{k+1})), \quad (\text{A.31})$$

$$\Pr(o_{k+1}|a_k)^{lo} = \sum_{s_{k+1} \in E_p(\bar{b}_{k+1})} O(s_{k+1}, o_{k+1}) \bar{b}_e(s_{k+1}|a_k) + \Pr(o_{k+1})^{\min} \Pr(L_p(\bar{b}_{k+1})), \quad (\text{A.32})$$

where $\Pr(o_{k+1})^{\max}$ and $\Pr(o_{k+1})^{\min}$ are, respectively, upper and lower bounds for the probability $O(s_{k+1}, o_{k+1})$, $\forall s_{k+1}$; and $\Pr(o_{k+1}|a_k)^{up}$ and $\Pr(o_{k+1}|a_k)^{lo}$ are, respectively, upper and lower bounds for $\Pr(o_{k+1}|a_k)$. In the case of discrete observations, choosing $\Pr(o_{k+1})^{\max} = 1$ and $\Pr(o_{k+1})^{\min} = 0$ is guaranteed to be consistent. This is the simplest possible type of approximation for (A.30), but serves as a starting point for approximating execution risks.

Previous sections provide upper and lower bounds for $r_b(b_k)$, while (A.31) and (A.32) give us a way to bound $\Pr(o_{k+1}|\pi(b_k))$. In the following, we consider an additional approximation of (A.29), namely the truncation of the sum over all children of b_k , which helps reduce the branching factor at that node. In the same spirit of $E_p(b_k)$ and $L_p(b_k)$ from Section A.1, here we define $E_o(b_k)$ and $L_o(b_k)$, respectively, as the enumerated and lumped sets of observations at the belief state b_k . Given these sets, we can rewrite (A.29) as

$$\begin{aligned}
er(b_k|\pi) &= r_b(b_k) + (1 - r_b(b_k))ER_k, \\
ER_k &= \left(\sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1}|\pi(b_k))^{sa} er(b_{k+1}|\pi) + \right. \\
&\quad \left. \sum_{o'_{k+1} \in L_o(b_k)} \Pr(o'_{k+1}|\pi(b_k))^{sa} er(b_{k+1}|\pi) \right). \tag{A.33}
\end{aligned}$$

From previous sections, we know that we can choose bounds $r_b^{up}(b_k)$ and $r_b^{lo}(b_k)$ for $r_b(b_k)$. The fact that $r_b(b_k)$ appears both positively and negatively in (A.33) might make it harder to realize how bounds on $r_b(b_k)$ affect bounds on $er(b_k|\pi)$. This confusion is eliminated by looking at the differential

$$\frac{\partial (er(b_k|\pi))}{\partial r_b(b_k)} = 1 - ER_k \geq 0, \tag{A.34}$$

where we must have $0 \leq ER_k \leq 1$ (it is the expected value of execution risks, which are always within the interval $[0, 1]$). Hence, we conclude that $er(b_k|\pi)$ is a non-decreasing function of $r_b(b_k)$. In other words, choosing an overestimate (underestimate) of $r_b(b_k)$ causes $er(b_k|\pi)$ to be overestimated (underestimated) as well.

Next, we look into the problem of finding a suitable approximation for the true value of ER_k in (A.33). Similar to what we have done before, let

$$er^{\max} = \max_b er(b|\pi), \quad er^{\min} = \min_b er(b|\pi), \tag{A.35}$$

be, respectively, upper and lower bounds of $er(b_{k+1}|\pi)$, $\forall b_{k+1}$. With those definitions

at hand, we can write the bounds

$$ER_k \leq \sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1} | \pi(b_k))^{sa} er(b_{k+1} | \pi) + er^{\max} \Pr(L_o(b_k)), \quad (\text{A.36})$$

$$ER_k \geq \sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1} | \pi(b_k))^{sa} er(b_{k+1} | \pi) + er^{\min} \Pr(L_o(b_k)), \quad (\text{A.37})$$

which do not depend on the unknown terms of the second summation in (A.33). As discussed in Chapter 3, the true value of $er(b_{k+1} | \pi)$ can only be obtained once all paths stemming from b_{k+1} have been expanded to terminal nodes. Hence, during most of the search process, we only have access to upper and lower estimates $h_{er}^{up}(b_{k+1} | \pi)$ and $h_{er}^{lo}(b_{k+1} | \pi)$ of $er(b_{k+1} | \pi)$. Incorporating these estimates into (A.36) and (A.37) yields

$$ER_k \leq \sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1} | \pi(b_k))^{sa} h_{er}^{up}(b_{k+1} | \pi) + er^{\max} \Pr(L_o(b_k)) = ER'_k, \quad (\text{A.38})$$

$$ER_k \geq \sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1} | \pi(b_k))^{sa} h_{er}^{lo}(b_{k+1} | \pi) + er^{\min} \Pr(L_o(b_k)) = ER''_k. \quad (\text{A.39})$$

We know from (A.30) that $\Pr(o_{k+1} | \pi(b_k))^{sa}$ cannot be computed exactly. Instead, we must settle for upper and lower bounds, such as (A.31) and (A.32). From

$$\Pr(L_o(b_k))^{sa} = 1 - \sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1} | \pi(b_k))^{sa},$$

we see that these probability terms appear both positively and negatively in (A.38)-(A.39). Hence, in order to decide which bound for $\Pr(o_{k+1} | \pi(b_k))^{sa}$ should be used, we perform a simple analysis similar to (A.34). Differentiating (A.38) and (A.39) with respect to $\Pr(o_{k+1} | \pi(b_k))^{sa}$ yields

$$\begin{aligned} \frac{\partial ER'_k}{\partial \Pr(o_{k+1} | \pi(b_k))^{sa}} &= h_{er}^{up}(b_{k+1} | \pi) - er^{\max} \leq 0, \\ \frac{\partial ER''_k}{\partial \Pr(o_{k+1} | \pi(b_k))^{sa}} &= h_{er}^{lo}(b_{k+1} | \pi) - er^{\min} \geq 0. \end{aligned} \quad (\text{A.40})$$

The inequalities in (A.40) show that (A.38) is a non-increasing function of the probabilities $\Pr(o_{k+1}|\pi(b_k))^{sa}$. Since we want to convert (A.38) into a guaranteed upper bound of G_k , we must replace $\Pr(o_{k+1}|\pi(b_k))^{sa}$ by $\Pr(o_{k+1}|\pi(b_k))^{sa,lo}$ in order not to cause (A.38) to drop below its true value. A similar analysis shows that the same substitution should be made in (A.39), since the previous inequalities show that (A.39) is a non-decreasing function of $\Pr(o_{k+1}|\pi(b_k))^{sa}$. Hence, we can write the computable bounds

$$ER_k \leq ER_k^{up} = \sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1}|\pi(b_k))^{sa,lo} h_{er}^{up}(b_{k+1}|\pi) + er^{\max} \Pr(L_o(b_k))^{sa,up}, \quad (\text{A.41})$$

$$ER_k \geq ER_k^{lo} = \sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1}|\pi(b_k))^{sa,lo} h_{er}^{lo}(b_{k+1}|\pi) + er^{\min} \Pr(L_o(b_k))^{sa,up}, \quad (\text{A.42})$$

where

$$\Pr(L_o(b_k))^{sa,up} = 1 - \sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1}|\pi(b_k))^{sa,lo}. \quad (\text{A.43})$$

Combining all the results in this section, we can derive upper and lower bounds for the execution risk $er(b_k|\pi)$ in the form

$$er^{up}(b_k|\pi) = r_b^{up}(b_k) + (1 - r_b^{up}(b_k))ER_k^{up}, \quad (\text{A.44})$$

$$er^{lo}(b_k|\pi) = r_b^{lo}(b_k) + (1 - r_b^{lo}(b_k))ER_k^{lo}. \quad (\text{A.45})$$

It should be noticed that the analysis in (A.34) and (A.40) allowed the bounds (A.41)-(A.45) to be (maybe only slightly) tighter than the ones obtained by mere maximization and minimization of every term in the equations.

A.1.4 Approximate forward-propagation of execution risks

The RAO* algorithm in Chapter 3 propagates risk upper bounds forward for the early detection of policy branches that are too risky to meet the chance constraints in the plan. Since the pruning of a policy branch must be based on a sufficient condition for infeasibility (or, conversely, a necessary condition for feasibility), we must provide an upper bound for (3.28) based on the terms that we can compute using a PEBS. One such upper bound can be computed as

$$er(b'_{k+1}|\pi) \leq \frac{\left(\frac{\tilde{\Delta}_k - r_b(b_k)^{lo}}{1 - r_b(b_k)^{lo}} - \sum_{o_{k+1} \neq o'_{k+1} \in E_o(b_k)} \Pr(o_{k+1}|\pi(b_k))^{sa,lo} er^{lo}(b_{k+1}|\pi) \right)}{\Pr(o'_{k+1}|\pi(b_k))^{sa,lo}}, \quad (\text{A.46})$$

where $r_b(b_k)^{lo}$, $\Pr(o_{k+1}|\pi(b_k))^{sa,lo}$, and $er^{lo}(b_{k+1}|\pi)$ are computed according to (A.28), (A.32), and (A.45), respectively. The reason for choosing $r_b(b_k)^{lo}$ in (A.46) is due to the fact that

$$\frac{\partial}{\partial r} \left(\frac{\tilde{\Delta} - r}{1 - r} \right) = \frac{\tilde{\Delta} - 1}{(1 - r)^2} \leq 0,$$

which means that an upper bound for the first term within parentheses is attained at a lower bound for $r_b(b_k)$.

A.1.5 Computing approximate utilities

Previous sections show how the approximation of belief states impacts our ability to compute precise values for the execution risk (A.29). In this section, we look at the very similar problem of how to compute the value

$$Q(b_k, a_k) = \sum_{s_k} R(s_k, a_k) b(s_k) + \gamma \sum_{o_{k+1}} \Pr(o_{k+1}|a_k) Q(b_{k+1})^* \quad (\text{A.47})$$

of taking action a_k at some belief state b_k , as defined by the CC-POMDP model in Chapter 3. Given the current best estimate $\hat{Q}(b_k, a_k)$ of the true value $Q(b_k, a_k)$, we select actions for the current estimate $\hat{\pi}$ of the optimal policy π^* according to

$$\hat{\pi}(b_k) = \arg \max_{a_k} \hat{Q}(b_k, a_k), \forall k. \quad (\text{A.48})$$

Similar to previous sections, there are two main challenges involved in the computation of (A.47) and (A.48):

- partial enumeration of belief state particles and observations prevent us from being able to compute the expectations in (A.47) exactly;
- the optimal values $Q(b_{k+1})^*$ are not available and usually cannot be computed forward (as opposed to goal-regression in dynamic programming).

We will address these two challenges leveraging the same strategy used for the execution risk: we will use heuristic estimates of $Q(b_{k+1})^*$ in order to compute reasonable predictions of the true value of (A.47); and we will use partially enumerated belief state particles and observations to compute upper and lower bounds of (A.47) so as to perform action selection in (A.48) by means of interval comparisons. Towards this goal, let

$$q_{a_k} = \sum_{s_k} R(s_k, a_k) b(s_k), \quad (\text{A.49})$$

$$EQ_{a_k} = \sum_{o_{k+1}} \Pr(o_{k+1} | a_k) Q(b_{k+1})^*. \quad (\text{A.50})$$

Upper and lower bounds for (A.49) can be obtained similarly to what was done for risk in (A.12)-(A.13). Thus, we get

$$q_{a_k}^{up} = \sum_{s_k \in E_p(b_k)} b_e(s_k) R(s_k, a_k) + R^{\max} \Pr(L_p(b_k)), \quad (\text{A.51})$$

$$q_{a_k}^{lo} = \sum_{s_k \in E_p(b_k)} b_e(s_k) R(s_k, a_k) + R^{\min} \Pr(L_p(b_k)), \quad (\text{A.52})$$

where R^{\max} and R^{\min} are, respectively, maximal and minimal values for the reward $R(s_k, a_k)$, $\forall s_k$. For (A.50), we can proceed analogously to (A.41)-(A.42) and obtain

$$EQ_{a_k}^{up} = \sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1}|a_k)^{lo} h_{Q^*}^{up}(b_{k+1}) + Q^{\max} \Pr(L_o(b_k))^{up}, \quad (\text{A.53})$$

$$EQ_{a_k}^{lo} = \sum_{o_{k+1} \in E_o(b_k)} \Pr(o_{k+1}|a_k)^{lo} h_{Q^*}^{lo}(b_{k+1}) + Q^{\min} \Pr(L_o(b_k))^{up}, \quad (\text{A.54})$$

where $\Pr(o_{k+1}|a_k)^{lo}$ and $\Pr(L_o(b_k))^{up}$ are computed according to (A.32) and (A.43), respectively; $h_{Q^*}^{up}(b_{k+1})$ and $h_{Q^*}^{lo}(b_{k+1})$ are, respectively, available upper and lower bounds for $Q(b_{k+1})^*$; and Q^{\max} and Q^{\min} are, respectively, maximal and minimal values of Q^* for any b_{k+1} .

Bounds for (A.47) can be derived by combining the previous results in the obvious way as

$$Q(b_k, a_k)^{up} = q_{a_k}^{up} + \gamma EQ_{a_k}^{up}, \quad (\text{A.55})$$

$$Q(b_k, a_k)^{lo} = q_{a_k}^{lo} + \gamma EQ_{a_k}^{lo}. \quad (\text{A.56})$$

A.2 Trial-based bounds

Section A.1 focuses on partial enumeration of observations and belief state particles as a means to achieve tractability. During our analysis, we saw how the uncertainty introduced by these partial enumerations affected our ability to perform exact, recursive estimation of belief states, risk, and utility. In the context of Section A.1, we relied on the ability of some algorithm to be able to enumerate these particles “in a good way”, i.e., quickly and concentrating most of the probability mass (ideally, in best-first order of probability).

A similar approximate behavior can be achieved in a trial-based fashion [Barto et al., 1995, Bertsekas and Tsitsiklis, , Keller and Helmert, 2013]. Instead of relying on some deterministic particle enumeration procedure, there is also the option of performing stochastic simulations of the system, henceforth referred to as *trials*, and

estimate quantities of interest based on their different outcomes. If we were able to generate samples from the true² underlying probability distributions, we would expect outcomes of higher probability trials to have greater impact on estimates.

Our approach for generating sample-based estimates with confidence bounds is to use Hoeffding's inequality [Hoeffding, 1963], which is stated in the following.

Theorem A.2 (Hoeffding's inequality [Hoeffding, 1963]). *If X_1, X_2, \dots, X_n are independent samples of a random variable X and $a_i \leq X_i \leq b_i, i = 1, 2, \dots, n$, then for $t > 0$,*

$$\Pr(\bar{X} - \mu \geq t) \leq e^{\left(-\frac{2n^2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)}, \quad (\text{A.57})$$

where

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i, \quad \mu = E[X].$$

Different from the Central Limit Theorem, which only provides guarantees on the distribution of \bar{X} in the limit when $n \rightarrow \infty$, the result in (A.57) provides an exponentially-decaying bound for the probability of the sample mean \bar{X} overestimating the true mean μ by more than t for any value of n .

If we replace the random variable X by $-X$, we can use (A.57) to derive the bound

$$\Pr(-\bar{X} + \mu \geq t) \leq e^{\left(-\frac{2n^2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)}, \quad (\text{A.58})$$

which limits the probability of \bar{X} underestimating μ by more than t . Finally, if we combine (A.57) and (A.58), we can derive an interval bound

$$\Pr(-t_l \leq \bar{X} - \mu \leq t_u) \leq e^{\left(-\frac{2n^2t_u^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)} + e^{\left(-\frac{2n^2t_l^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)}, \quad (\text{A.59})$$

²As assumed by our models.

which exploits the fact that \bar{X} cannot overestimate and underestimate μ at the same time. Finally, if we are only interested in an interval of size t , we get

$$\Pr (|\bar{X} - \mu| \geq t) \leq 2e \left(-\frac{2n^2 t^2}{\sum_{i=1}^n (b_i - a_i)^2} \right). \quad (\text{A.60})$$

The purpose of using these inequalities is illustrated in Figure A-2. Let us assume that we want to estimate the expected value $E[X] = \mu$ of some quantity of interest X for some belief state b_k under a given policy π , e.g., utility or execution risk. If we did not have the transition and observations models associated with POMDP's, we could still try to estimate these quantities by running multiple simulations of π from b_k and using sample means \bar{X} as estimates of the true values. However, how many trials should we run in order to be confident that our upper and lower bounds, indeed, contain the true μ ?

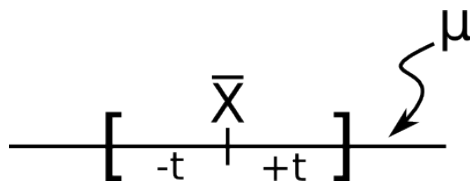


Figure A-2: Situation in which an interval of width $2t$ around the sample mean \bar{X} does not contain the true mean μ .

A sufficient condition for ensuring that $\Pr (|\bar{X} - \mu| \geq t) \leq \alpha$, given t and a confidence level α , is to choose n such that α equals the upper bound in (A.60). Hence, we get

$$n = \frac{(X^{\max} - X^{\min})^2 \ln \left(\frac{2}{\alpha} \right)}{2t^2}, \quad (\text{A.61})$$

where X^{\max} and X^{\min} are, respectively, the maximal and minimal values that X can assume during any given trial. In case the number of trials n and the confidence level α are fixed, a simple rearrangement of (A.61) allows us to compute the width t as

$$t = \sqrt{\frac{\ln \left(\frac{2}{\alpha} \right)}{2} \frac{(X^{\max} - X^{\min})}{\sqrt{n}}}. \quad (\text{A.62})$$

It might also be useful to determine the “sensitivity” of our bound estimates with respect to additional samples. This could be useful, for example, to determine which portion of a policy to simulate, given finite computational resources. One can easily derive an instantaneous sensitivity measure by taking the derivative of (A.62) with respect to n , yielding

$$\frac{dt}{dn} = -\sqrt{\frac{\ln\left(\frac{2}{\alpha}\right)}{2}} \frac{(X^{\max} - X^{\min})}{2n\sqrt{n}} = \frac{-t}{2n}. \quad (\text{A.63})$$

A plot of (A.63) for different values of n is shown in Figure A-3. It clearly confirms the intuitive notion that, as more and more samples are drawn, their individual impact on the quality of our bounds (namely, the magnitude of t) rapidly decreases. Therefore, when using simulations to evaluate the utility and risk of different actions in a policy, it makes sense that we try to distribute them evenly across different actions, as opposed to focusing a lot of effort on making the bounds for one particular action very precise.

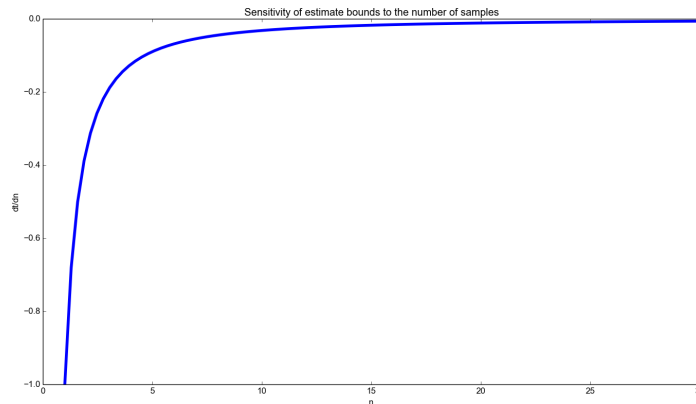


Figure A-3: Sensitivity of t with respect to n for a constant term multiplying $1/(n\sqrt{n})$ equals to -1 . The more negative the value, the better.

As mentioned before, the usefulness of (A.63) becomes apparent when we are trying to choose between actions whose utility and risk estimates overlap, as shown in Figure A-4. In these situations, there is no clear way to disambiguate between the two, so we must perform additional simulations in order to tighten the intervals. But if we have a limited simulation budget, how should we distribute it over the

different actions? According to (A.63), we should follow a greedy strategy and always simulate the action which has been simulated the least, since it will cause the biggest tightening of the overall intervals. This simple result was already expected, since the sample independence assumption in (A.57) causes current trials to provide no further information besides the results of the trials themselves.

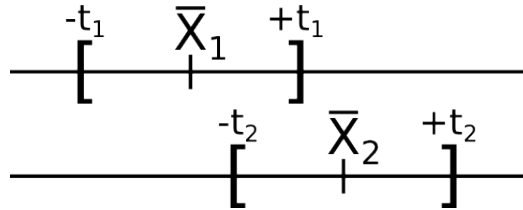


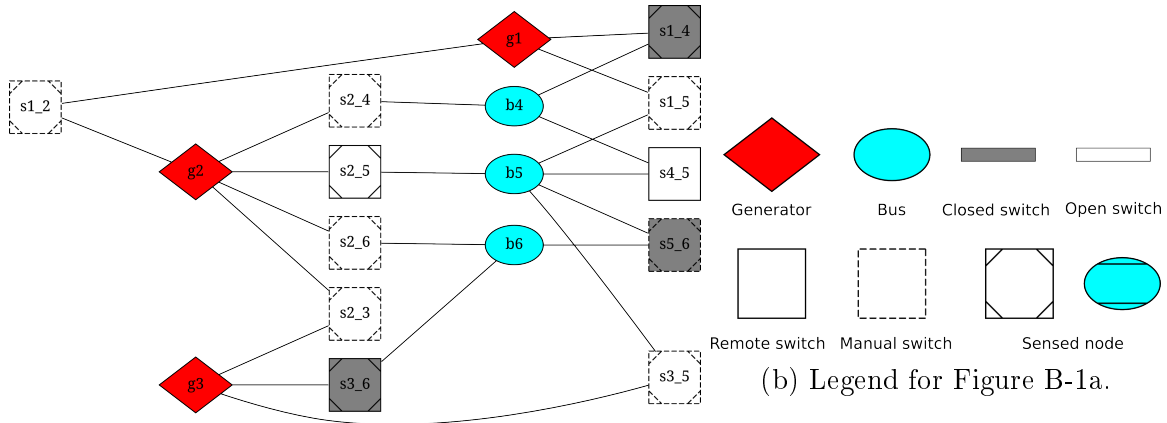
Figure A-4: Ambiguity for selecting between actions a_1 and a_2 , which generated, respectively, the sample means \bar{X}_1 and \bar{X}_2 .

Appendix B

A SAT model for power supply restoration

Power restoration in a partially observable setting requires a planner to reason over a belief space over the set of possible network states. In a simple instantiation with no probability information, one could consider a model-based inference strategy leveraging the power network's connection diagram, a model of its various sensors, and their readings in order to determine sets of possible or consistent states. Figure B-1 shows one such power network. These are generated automatically from the MATPOWER benchmarks [Zimmerman et al., 2011], where a switch node is added on each connection between buses and generators. Not only it allows connections between nodes to be turned on and off, but it also makes it straightforward to model line faults the same way as bus faults.

It is important to stress that the models in this section assume that no double-feeding exists in the network. In other words, a line or bus is either receiving no power or is connected to a single generator. Even though multiple-feeding is technically possible, it creates difficulties for simulating fault sensors that depend on a fault being *downstream* from a generator. For single-feeding, the direction of power flow is obvious: it always comes from the direction connected to the generator. However, when multiple generators feed a node, the direction of power flow becomes dependent on the generation and transmission parameters.



(a) Small instance of a power network with fault sensors.

Figure B-1: Small instance of a power network with fault sensors.

B.1 Modeling circuit-breakers

A circuit-breaker is a fault sensor that can be anchored to buses or line switches. It works very similarly to a relay, in the sense that current is used to trigger a switch that either closes or interrupts a circuit. A circuit-breaker is designed to prevent propagation of current surges by triggering its switch open whenever current goes beyond some predefined threshold. In the case of single-feeding, this restricts the circuit-breaker to detecting faults that are downstream from the generator, where downstream means that the generator and the fault should be on opposite sides of the circuit-breaker.

This can be more clearly understood by observing Figure B-2, which shows the electrical model of a feeder. Figure B-2b shows an electrically equivalent model of Figure B-2a that facilitates the analysis of the current flowing through the circuit-breaker. From Figure B-2b, we see that the current through the circuit-breaker only depends on the equivalent admittance Y_{down} from the downstream buses. Therefore, even if a fault creates a current spike in the generator by suddenly increasing Y_{up} , the current flowing through the circuit-breaker will remain unaltered. With that said, we are ready to develop our SAT model for circuit-breakers in a power network with single-feeding.

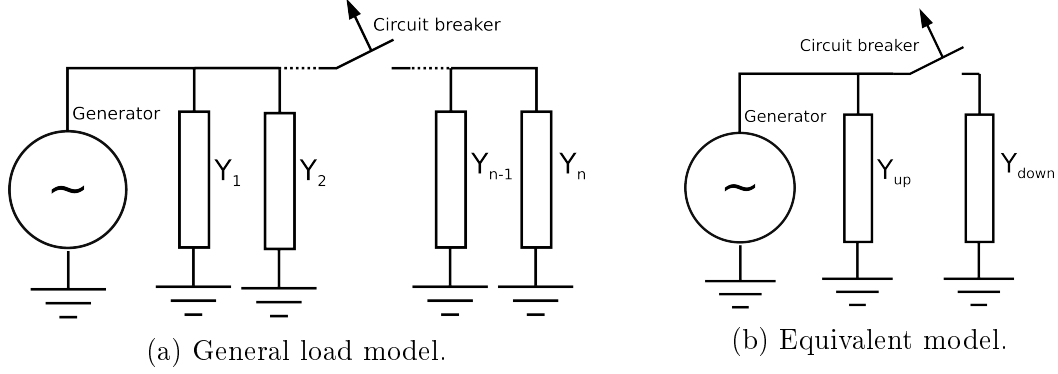


Figure B-2: Electrical model of a feeder.

We model a circuit-breaker as a discrete variable with domain $\{open, closed, off\}$. If a circuit-breaker is not receiving power, it remains *off* and provides no information about the circuit that it is connected to. Let cb and t stand for, respectively, a generic circuit-breaker and time step. We define

$$\begin{aligned} Powered(cb, t) &\Leftrightarrow Closed(cb, t) \wedge ConnectedGenerator(cb, t) \\ \neg Powered(cb, t) &\Leftrightarrow cb = off \end{aligned}$$

Let t^- and t^+ be, respectively, the time steps immediately before and after the occurrence of an electrical fault. A circuit-breaker signals a downstream fault according to the model

$$Powered(cb, t^-) \Rightarrow Open(cb, t^+) \Rightarrow FaultDownstream(cb, t^-), \quad (B.1)$$

where

$$FaultDownstream(cb, t^-) \equiv \bigvee_{n \in \mathbb{DS}(cb)} Fault(n). \quad (B.2)$$

The set $\mathbb{DS}(cb)$ in (B.2) is the set of downstream nodes from the circuit breaker in the direction of the power flow, as shown in Figure B-2b. For example, the set of downstream nodes for the circuit-breakers anchored to switches $s3_6$ and $s5_6$

in Figure B-1a are, respectively, $\{b5, b6\}$ and $\{b5\}$. The model in (B.1) assumes the possibility that a circuit-breaker may malfunction and not react to a downstream fault. However, if we are guaranteed that a circuit-breaker will always react to a fault, the SAT model of the circuit-breaker becomes

$$\begin{aligned}
 & \text{Powered}(cb, t^-) \Rightarrow (\text{Open}(cb, t^+) \Leftrightarrow \text{FaultDownstream}(cb, t^-)) \\
 & \quad \equiv \\
 & (\text{Open}(cb, t^+) \Rightarrow \text{FaultDownstream}(cb, t^-)) \wedge (\text{Closed}(cb, t^+) \Rightarrow \neg \text{FaultDownstream}(cb, t^-))
 \end{aligned}
 \tag{B.3}$$

B.2 Simulating the network

1. Given state of the switches, start determining which faults are receiving power.
2. Propagate fault outward from the location of the faults.
 - (a) If the fault hits an open switch, stop expanding that branch (adds nothing to the queue)
 - (b) If the fault hits a deterministic circuit-breaker, open it. Otherwise, flip a coin and decide whether the switch should be opened or not.
 - (c) It's necessary to determine if the fault is "downstream" from a generator or not.

Appendix C

PTPN XML schema

This appendix contains the XML schema used to translate cRMPL programs into PTPN's, the input type to Pike. Main contributors to these schema, listed in alphabetical order by first name, are Andreas Hofmann, Andrew Wang, Brian Williams, Cheng Fang, David Wang, Enrique Fernandez, Erez Karpas, Eric Timmons, James Paterson, Pedro Santana, Peng Yu, Steven Levine, and Tiago Vaquero.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema
  xmlns="http://mers.csail.mit.edu/tpn"
  targetNamespace="http://mers.csail.mit.edu/tpn"
  xmlns:t="http://mers.csail.mit.edu/tpn"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:include schemaLocation="tpn-defs.xsd"/>

  <xs:element name="tpns" type="tpns"/>
</xs:schema>
```

Listing C.1: Schema for a file containing multiple PTPN's.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://mers.csail.mit.edu/tpn"
  xmlns:t="http://mers.csail.mit.edu/tpn"
  xmlns="http://mers.csail.mit.edu/tpn"
  elementFormDefault="qualified">

  <xs:simpleType name="TPN-feature-type">
    <xs:restriction base="xs:string">
```

```

        <xs:enumeration value="simple-temporal-constraints"/>
        <xs:enumeration value="contingent-temporal-constraints"/>
        <xs:enumeration value="simple-contingent-temporal-constraints"/>
    </xs:restriction>
</xs:simpleType>

<xs:complexType name="decision-variable-equals">
    <xs:sequence>
        <xs:element name="variable" type="xs:IDREF"/>
        <xs:element name="value" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="state-variable-value-at">
    <xs:sequence>
        <xs:element name="variable" type="xs:IDREF"/>
        <xs:element name="event" type="xs:IDREF"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="state-variable-guard-value">
    <xs:choice>
        <xs:element name="constant" type="xs:string" />
        <xs:element name="state-variable-at" type="state-variable-value-at" />
    </xs:choice>
</xs:complexType>

<xs:complexType name="state-variable-guard-boolean-expr">
    <xs:sequence>
        <xs:element name="condition" type="xs:string"/>
        <xs:element name="value" type="state-variable-guard-value" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="guard-list">
    <xs:sequence>
        <xs:element name="guard" type="guard" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="guard">
    <xs:choice>
        <xs:element name="boolean-constant" type="xs:boolean"/>
    </xs:choice>

```

```

<xs:element name="decision-variable-equals" type="decision-variable-equals"/>
<xs:element name="state-variable-guard" type="state-variable-guard-boolean-expr"/
>
<xs:element name="and" type="guard-list" />
<xs:element name="or" type="guard-list" />
<xs:element name="not">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="guard" type="guard"×/xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>

<xs:complexType name="variable-domain">
  <xs:choice>
    <xs:element name="finite-domain" type="finite-variable-domain"×/xs:element>
    <xs:element name="continuous-domain" type="continuous-variable-domain"×/
xs:element>
  </xs:choice>
</xs:complexType>

<xs:complexType name="range">
  <xs:sequence>
    <xs:element name="lower-bound" type="xs:double" maxOccurs="1" minOccurs="0"×/
xs:element>
    <xs:element name="upper-bound" type="xs:double" maxOccurs="1" minOccurs="0"×/
xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="continuous-variable-domain">
  <xs:sequence>
    <xs:element name="range" type="range" maxOccurs="unbounded" minOccurs="1"×/
xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="finite-variable-domain">
  <xs:sequence>
    <xs:element name="value" type="xs:string" maxOccurs="unbounded" minOccurs="1"×
/xs:element>
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="guarded-tn-member">
  <xs:sequence>
    <xs:element name="id" type="xs:ID"/>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="guard" type="guard"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="dispatchable-guarded-tn-member">
  <xs:complexContent>
    <xs:extension base="guarded-tn-member">
      <xs:sequence>
        <xs:element name="dispatch" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:simpleType name="decision-variable-type">
  <xs:restriction base="xs:string">
    <xs:enumeration value="controllable"/>
    <xs:enumeration value="uncontrollable"/>
    <xs:enumeration value="probabilistic"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="decision-variable">
  <xs:complexContent>
    <xs:extension base="guarded-tn-member">
      <xs:sequence>
        <xs:element name="type" type="decision-variable-type"/>
        <xs:element name="at-event" type="xs:IDREF" minOccurs="0"/>
        <xs:element name="domain">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="domainval" maxOccurs="unbounded" minOccurs="1">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="value" type="xs:string"/>
                    <xs:element name="utility" type="xs:double" minOccurs="0"/>
                    <xs:element name="probability" type="xs:double" minOccurs="0"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

```

```

        </xs:complexType>
        </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="state-variable">
<xs:sequence>
    <xs:element name="id" type="xs:ID"></xs:element>
    <xs:element name="name" type="xs:string"></xs:element>
    <xs:element name="domain" type="variable-domain"></xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="distribution">
<xs:sequence>
    <xs:element name="distribution-type" type="xs:string"></xs:element>
    <xs:element name="parameters">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="parameter" type="xs:float" maxOccurs="unbounded"
minOccurs="0"></xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="duration">
<xs:choice>
    <xs:element name="bounded-duration">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="lower-bound" type="xs:double" maxOccurs="1" minOccurs="
0"></xs:element>
                <xs:element name="upper-bound" type="xs:double" maxOccurs="1" minOccurs="0"
></xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

```

```

<xs:element name="set-bounded-uncertain-duration">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="lower-bound" type="xs:double" maxOccurs="1" minOccurs="0"/>
      <xs:element name="upper-bound" type="xs:double" maxOccurs="1" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="probabilistic-uncertain-duration" type="distribution">
</xs:element>
</xs:choice>
</xs:complexType>

<xs:complexType name="wff-value">
  <xs:choice>
    <xs:element name="constant" type="xs:string" />
    <xs:element name="state-variable" type="xs:IDREF" />
  </xs:choice>
</xs:complexType>

<xs:simpleType name="timepoint-type">
  <xs:restriction base="xs:string">
    <xs:enumeration value="start"/>
    <xs:enumeration value="end"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="wff-boolean-expression-timepoint">
  <xs:sequence>
    <xs:element name="timepoint" type="timepoint-type" />
    <xs:element name="offset" type="xs:double" maxOccurs="1" minOccurs="0" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="wff-boolean-expression-when-at">
  <xs:sequence>
    <xs:element name="at" type="wff-boolean-expression-timepoint" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="wff-boolean-expression-when-between">
  <xs:sequence>

```



```

    <xs:element name="from" type="wff-boolean-expression-timepoint" />
    <xs:element name="to" type="wff-boolean-expression-timepoint" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="wff-boolean-expression-when">
  <xs:choice>
    <xs:element name="at" type="wff-boolean-expression-when-at" />
    <xs:element name="between" type="wff-boolean-expression-when-between" />
  </xs:choice>
</xs:complexType>

<xs:complexType name="wff-boolean-expression">
  <xs:sequence>
    <xs:element name="when" type="wff-boolean-expression-when" minOccurs="0"/>
    <xs:element name="condition" type="xs:string" />
    <xs:element name="value" type="wff-value" maxOccurs="unbounded" minOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="wff-list">
  <xs:sequence>
    <xs:element name="wff" type="wff" maxOccurs="unbounded" minOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="wff">
  <xs:choice>
    <xs:element name="boolean-constant" type="xs:boolean" />
    <xs:element name="boolean-expression" type="wff-boolean-expression" />
    <xs:element name="and" type="wff-list" />
    <xs:element name="or" type="wff-list" />
    <xs:element name="not">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="wff" type="wff"></xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>

<xs:complexType name="state-constraint">

```

```

    <xs:choice>
      <xs:element name="wff" type="wff" />
    </xs:choice>
  </xs:complexType>

<xs:complexType name="event">
  <xs:complexContent>
    <xs:extension base="dispatchable-guarded-tn-member">
      <xs:sequence>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

<xs:complexType name="temporal-constraint">
  <xs:complexContent>
    <xs:extension base="guarded-tn-member">
      <xs:sequence>
        <xs:element name="to-event" type="xs:IDREF"></xs:element>
        <xs:element name="from-event" type="xs:IDREF"></xs:element>
        <xs:element name="duration"
          type="duration">
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="episode">
  <xs:complexContent>
    <xs:extension base="temporal-constraint">
      <xs:sequence>
        <xs:element name="dispatch" type="xs:string" minOccurs="0"></xs:element>
        <xs:element name="macro-tpn-id" type="xs:string" minOccurs="0"></
xs:element>
        <xs:element name="state-constraint"
          type="state-constraint">
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="chance-constraint">

```

```

<xs:sequence>
  <xs:element name="id" type="xs:ID"></xs:element>
  <xs:element name="name" type="xs:string"></xs:element>
  <xs:element name="constraints" type="xs:IDREFS"></xs:element>
  <xs:element name="probability" type="xs:float"></xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="state-variable-assignment">
  <xs:sequence>
    <xs:element name="state-variable" type="xs:IDREF"></xs:element>
    <xs:element name="value" type="xs:string"></xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="state">
  <xs:sequence>
    <xs:element name="assignment" type="state-variable-assignment" minOccurs="0"
maxOccurs="unbounded"></xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="tpn">
<xs:sequence>
  <xs:element name="id" type="xs:ID"></xs:element>
  <xs:element name="name" type="xs:string"></xs:element>
  <xs:element name="features">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="feature" type="TPN-feature-type" minOccurs="0"
maxOccurs="unbounded"></xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="start-event" type="xs:IDREF"></xs:element>
  <xs:element name="end-event" type="xs:IDREF" minOccurs="0"></xs:element>

  <xs:element name="events">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="event" type="event" minOccurs="0" maxOccurs="unbounded"
></xs:element>
      </xs:sequence>
    </xs:complexType>

```

```

</xs:element>

<xs:element name="temporal-constraints">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="temporal-constraint" type="temporal-constraint"
minOccurs="0" maxOccurs="unbounded"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="episodes">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="episode" type="episode" minOccurs="0" maxOccurs="
unbounded"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="chance-constraints">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="chance-constraint" type="chance-constraint" minOccurs="
0" maxOccurs="unbounded"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="decision-variables">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="decision-variable" type="decision-variable" minOccurs="
0" maxOccurs="unbounded"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="state-variables">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="state-variable" type="state-variable" minOccurs="0"
maxOccurs="unbounded"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

    </xs:complexType>
  </xs:element>

  <xs:element name="initial-state" type="state" minOccurs="0" />

</xs:sequence>
</xs:complexType>

<xs:complexType name="tpns">
  <xs:sequence>
    <xs:element name="tpn" type="tpn" minOccurs="0" maxOccurs="unbounded">
      <!-- Make sure keys / IDs match within this TPN -->
      <!-- Make sure all temporal constraints and episodes have unique ID's -->
      <xs:key name="TEMPORAL-CONSTRAINT-ID">
        <xs:selector xpath="/t:temporal-constraints/t:temporal-constraint/t:id
| ./t:episodes/t:episode/t:id" />
        <xs:field xpath="." />
      </xs:key>

      <!-- Make sure all events have unique ID's -->
      <xs:key name="EVENT-ID">
        <xs:selector xpath="/t:events/t:event/t:id" />
        <xs:field xpath="." />
      </xs:key>

      <!-- Make sure all decision & state variables have unique ID's -->
      <xs:key name="VARIABLE-ID">
        <xs:selector xpath="//t:decision-variable/t:id | ../t:state-variable/
t:id" />
        <xs:field xpath="." />
      </xs:key>

      <!-- Make sure all temporal constraint from-event and to-event point to
          actual event ID's -->
      <xs:keyref name="TEMPORAL-CONSTRAINT-EVENT-IDREF" refer="EVENT-ID">
        <xs:selector xpath="/t:temporal-constraints/t:temporal-constraint/t:from
-event | ./t:temporal-constraints/t:temporal-constraint/t:to-event"/>
        <xs:field xpath="." />
      </xs:keyref>

      <!-- Make sure all episode from-event and to-event point to
          actual event ID's -->
      <xs:keyref name="EPISODE-EVENT-IDREF" refer="EVENT-ID">
        <xs:selector xpath="/t:episodes/t:episode/t:from-event | ./t:episodes/
t:episode/t:to-event"/>

```

```

    <xs:field xpath="."/>
  </xs:keyref>

  <!-- Make sure the TPN's start and end events also refer to
        actual event ID's -->
  <xs:keyref name="TPN-START-END-EVENT-IDREF" refer="EVENT-ID">
    <xs:selector xpath="./t:start-event | ./t:end-event"/>
    <xs:field xpath="."/>
  </xs:keyref>

  <!-- Make sure all references to variable ID's are actual ID's -->
  <xs:keyref name="VARIABLE-IDREF" refer="VARIABLE-ID">
    <xs:selector xpath="//t:variable"/>
    <xs:field xpath="."/>
  </xs:keyref>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:schema>

```

Listing C.2: Schema for PTPN definitions.

Appendix D

RSS model

This appendix contains the PDDL model used in the Resilient Space Systems (RSS) demonstration of Chapter 7.

```
(define (domain RSS_Project_Rover_Scenario)
  (:requirements :typing)
  (:types
    Rover - object
    Request - object
    RockSampleRequest - Request
    PictureRequest - Request
    Location - object)
  (:predicates
    (mastcam_ready ?rov - Rover)
    (hazcam_ready ?rov - Rover)
    (idle ?rov - Rover)
    (mastcam_on ?rov - Rover)
    (hazcam_on ?rov - Rover)
    (pending ?req - Request)
    (done ?req - Request)
    (transmitted ?req - Request)
    (has_rock ?loc - Location)
    (can_go_to ?loc - Location)
    (orbiter_communication_available ?loc - Location)
    (surveyed ?loc - Location)
    (at ?rov - Rover ?loc - Location)
    (location ?pic - PictureRequest ?loc - Location)
    (mastcam_issue ?rov - Rover))

  (:action move
    :parameters (? self - Rover ?loc1 - Location ?loc2 - Location)
```

```

:precondition
  (and
    (at ?self ?loc1)
    (can_go_to ?loc2)
    (idle ?self))
:effect
  (and
    (not (at ?self ?loc1))
    (at ?self ?loc2)
    (idle ?self)))

(:action turnon_mastcam
:parameters (?self - Rover ?loc - Location)
:precondition
  (and
    (at ?self ?loc)
    (idle ?self)
    (mastcam_ready ?self))
:effect
  (and
    (not (idle ?self))
    (mastcam_on ?self)))

(:action turnon_hazcam
:parameters (?self - Rover ?loc - Location)
:precondition
  (and
    (at ?self ?loc)
    (idle ?self)
    (hazcam_ready ?self)
    (mastcam_issue ?self))
:effect
  (and
    (not (idle ?self))
    (hazcam_on ?self)))

(:action take_pictures_mastcam
:parameters (?self - Rover ?loc - Location ?req - PictureRequest)
:precondition
  (and
    (at ?self ?loc)
    (mastcam_on ?self)
    (location ?req ?loc)
    (pending ?req))
:effect

```



```

    (and
      (done ?req)
      (idle ?self)
      (not (mastcam_on ?self))
      (idle ?self)))

(:action take_pictures_hazcam
 :parameters (?self - Rover ?loc - Location ?req - PictureRequest)
 :precondition
  (and
    (at ?self ?loc)
    (hazcam_on ?self)
    (location ?req ?loc)
    (pending ?req)
    (mastcam_issue ?self))
 :effect
  (and
    (not (pending ?req))
    (done ?req)
    (not (hazcam_on ?self))
    (idle ?self)))

(:action survey_location
 :parameters (?self - Rover ?loc - Location)
 :precondition
  (and
    (at ?self ?loc)
    (idle ?self))
 :effect
  (and
    (surveyed ?loc)
    (not (idle ?self))))

(:action collect_rock_sample
 :parameters (?self - Rover ?loc - Location ?req - RockSampleRequest)
 :precondition
  (and
    (at ?self ?loc)
    (pending ?req)
    (has_rock ?loc)
    (surveyed ?loc))
 :effect
  (and
    (not (pending ?req))
    (done ?req))

```

```

    (not (has_rock ?loc))
    (idle ?self)))

(:action transmit_data
 :parameters (?self - Rover ?loc - Location ?req - Request)
 :precondition
  (and
   (at ?self ?loc)
   (done ?req)
   (orbiter_communication_available ?loc)
   (idle ?self))
 :effect
  (and
   (transmitted ?req))))

```

Listing D.1: Domain file.

```

(define (problem Planning_Problem)
 (:domain RSS_Project_Rover_Scenario)
 (:objects
  rover1 - Rover
  l1 - Location
  l2 - Location
  l3 - Location
  l4 - Location
  l5 - Location
  unknown - Location
  pic_req1 - PictureRequest
  pic_req2 - PictureRequest
  pic_req3 - PictureRequest
  rock_req1 - RockSampleRequest
  rock_req2 - RockSampleRequest)

 (:init
  (at rover1 l1)
  (has_rock l2)
  (has_rock l4)
  (has_rock l5)
  (can_go_to l1)
  (can_go_to l2)
  (can_go_to l3)
  (can_go_to l4)
  (can_go_to l5)
  (idle rover1)
  (location pic_req1 l3)
  (location pic_req2 l2)

```

```
(location pic_req3 15)
(mastcam_ready rover1)
(hazcam_ready rover1)
(orbiter_communication_available 12)
(orbiter_communication_available 14)
(pending pic_req1)
(pending pic_req2)
(pending pic_req3)
(pending rock_req1)
(pending rock_req2))

(:goal
 (and
  (transmitted pic_req1)
  (transmitted pic_req2)
  (transmitted pic_req3)
  (transmitted rock_req1)
  (transmitted rock_req2))))
```

Listing D.2: Problem file.

Bibliography

- [Adept MobileRobots, 2005] Adept MobileRobots (2005). MobileSim. Available online at <http://robots.mobilerobots.com/wiki/MobileSim>.
- [Administration, 2009] Administration, F. A. (2009). *Risk Management Handbook*. FAA. FAA-H-8083-2.
- [Altman, 1999] Altman, E. (1999). *Constrained Markov Decision Processes*, volume 7. CRC Press.
- [Andre and Russell, 2002] Andre, D. and Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*, pages 119–125.
- [APPL, 2014] APPL (April 25, 2014). *PomdpX File Format (version 1.0)*. National University of Singapore. Available online at <http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php?n=Main.PomdpXDocumentation>.
- [Arantes et al., 2015] Arantes, J. d. S., Arantes, M. d. S., Toledo, C. F. M., and Williams, B. C. (2015). A multi-population genetic algorithm for uav path re-planning under critical situation. In *Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on*, pages 486–493. IEEE.
- [Arantes et al., 2016a] Arantes, M., Arantes, J., Toledo, C., and Williams, B. (2016a). A hybrid multi-population genetic algorithm for uav path planning. In *Genetic and Evolutionary Computation Conference*.
- [Arantes et al., 2016b] Arantes, M., Toledo, C., Williams, B., and Ono, M. (2016b). Collision-free encoding for chance-constrained, non-convex path planning. *IEEE Transactions on Robotics*. To appear.
- [Bajada et al., 2015] Bajada, J., Fox, M., and Long, D. (2015). Temporal planning with semantic attachment of non-linear monotonic continuous behaviours. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1523–1529. AAAI Press.
- [Barto et al., 1995] Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138.

- [Baum and Petrie, 1966] Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563.
- [Beaudry et al., 2010] Beaudry, E., Kabanza, F., and Michaud, F. (2010). Planning for concurrent action executions under action duration uncertainty using dynamically generated bayesian networks. In *ICAPS*, pages 10–17.
- [Behnel et al., 2011] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K. (2011). Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31–39.
- [Bellman, 1956] Bellman, R. (1956). Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10):767–769.
- [Benazera et al., 2005a] Benazera, E., Brafman, R., Meuleau, N., and Hansen, E. (2005a). An AO* Algorithm for Planning with Continuous Resources. In *In Workshop on Planning under Uncertainty for Autonomous Systems, associated with the International Conference on AI Planning and Scheduling (ICAPS)*.
- [Benazera et al., 2005b] Benazera, E., Brafman, R., Meuleau, N., Hansen, E. A., et al. (2005b). Planning with continuous resources in stochastic domains. In *International Joint Conference on Artificial Intelligence*, volume 19, page 1244.
- [Benton et al., 2012] Benton, J., Coles, A. J., and Coles, A. (2012). Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*, volume 77, page 78.
- [Berry and Gonthier, 1992] Berry, G. and Gonthier, G. (1992). The esterel synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87 – 152.
- [Bertsekas and Tsitsiklis,] Bertsekas, D. and Tsitsiklis, J. Neuro-dynamic programming, 1996. *Athena Scientific, Belmont, MA*.
- [Birge and Louveaux, 1997] Birge, J. R. and Louveaux, F. V. (1997). *Introduction to stochastic programming*. Springer.
- [Blackmore et al., 2011] Blackmore, L., Ono, M., and Williams, B. C. (2011). Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics*, 27(6):1080–1094.
- [Blackmore, 2007] Blackmore, L. J. C. (2007). *Robust execution for stochastic hybrid systems*. PhD thesis, Massachusetts Institute of Technology.
- [Block et al., 2006] Block, S. A., Wehowsky, A. F., and Williams, B. C. (2006). Robust execution on contingent, temporally flexible plans. In *AAAI*, volume 2006, pages 802–808.

- [Blum and Furst, 1997] Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial intelligence*, 90(1):281–300.
- [Bonet and Geffner, 2000] Bonet, B. and Geffner, H. (2000). Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 52–61.
- [Bonet and Geffner, 2003] Bonet, B. and Geffner, H. (2003). Labeled rtdp: Improving the convergence of real-time dynamic programming. In *ICAPS*, volume 3, pages 12–21.
- [Bonet and Geffner, 2009] Bonet, B. and Geffner, H. (2009). Solving POMDPs: RTDP-Bel vs. Point-based Algorithms. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1641–1646.
- [Bonet and Thiébaux, 2003] Bonet, B. and Thiébaux, S. (2003). Gpt meets psr. In *13th International Conference on Automated Planning and Scheduling*, pages 102–111.
- [Boutilier et al., 2000] Boutilier, C., Reiter, R., Soutchanski, M., Thrun, S., et al. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In *AAAI/IAAI*, pages 355–362.
- [Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.
- [Bryce et al., 2015] Bryce, D., Gao, S., Musliner, D. J., and Goldman, R. P. (2015). SMT-Based Nonlinear PDDL+ Planning. In *AAAI*, pages 3247–3253.
- [Burke et al., 2014] Burke, S., Fernandez, E., Figueredo, L., Hofmann, A., Hofmann, C., Karpas, E., Levine, S., Santana, P., Yu, P., and Williams, B. (2014). Intent Recognition and Temporal Relaxation in Human Robot Assembly. In *The ICAPS-14 System Demonstrations*.
- [Campbell et al., 2002] Campbell, M., Hoane, A. J., and Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*, 134(1):57–83.
- [Cassez et al., 2005] Cassez, F., David, A., Fleury, E., Larsen, K. G., and Lime, D. (2005). Efficient on-the-fly algorithms for the analysis of timed games. In *International Conference on Concurrency Theory*, pages 66–80. Springer.
- [Cimatti et al., 2014] Cimatti, A., Hunsberger, L., Micheli, A., Posenato, R., and Roveri, M. (2014). Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In *2014 21st International Symposium on Temporal Representation and Reasoning*, pages 27–36. IEEE.

- [Cimatti et al., 2016a] Cimatti, A., Hunsberger, L., Micheli, A., Posenato, R., and Roveri, M. (2016a). Dynamic controllability via timed game automata. *Acta Informatica*, pages 1–42.
- [Cimatti et al., 2015] Cimatti, A., Micheli, A., and Roveri, M. (2015). Strong temporal planning with uncontrollable durations: a state-space approach. *AAAI*.
- [Cimatti et al., 2016b] Cimatti, A., Micheli, A., and Roveri, M. (2016b). Dynamic Controllability of Disjunctive Temporal Networks: Validation and Synthesis of Executable Strategies. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*.
- [Coffrin and Van Hentenryck, 2014] Coffrin, C. and Van Hentenryck, P. (2014). A linear-programming approximation of AC power flows. *INFORMS Journal on Computing*, 26(4):718–734.
- [Coles et al., 2008] Coles, A., Fox, M., Long, D., and Smith, A. (2008). Planning with problems requiring temporal coordination. In *AAAI*, pages 892–897.
- [Coles et al., 2009] Coles, A. J., Coles, A., Fox, M., and Long, D. (2009). Temporal Planning in Domains with Linear Processes. In *IJCAI*, pages 1671–1676. Citeseer.
- [Coles et al., 2010] Coles, A. J., Coles, A. I., Fox, M., and Long, D. (2010). Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.
- [Coles et al., 2012] Coles, A. J., Coles, A. I., Fox, M., and Long, D. (2012). COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research*, pages 1–96.
- [Combi et al., 2013] Combi, C., Hunsberger, L., and Posenato, R. (2013). An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty-revisited. In *Agents and Artificial Intelligence*, pages 314–331. Springer.
- [Comtet, 1974] Comtet, L. (1974). *Advanced Combinatorics: The art of finite and infinite expansions*. Springer.
- [Conrad, 2010] Conrad, P. R. (2010). Flexible execution of plans with choice and uncertainty. Master’s thesis, Massachusetts Institute of Technology.
- [Conrad et al., 2009] Conrad, P. R., Shah, J. A., and Williams, B. C. (2009). Flexible execution of plans with choice. In *ICAPS*, volume 9, pages 74–81.
- [Conrad and Williams, 2011] Conrad, P. R. and Williams, B. C. (2011). Drake: An efficient executive for temporal plans with choice. *Journal of Artificial Intelligence Research*, 42(1):607–659.

- [Cooper, 1990] Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405.
- [Cushing et al., 2007] Cushing, W., Kambhampati, S., Weld, D. S., et al. (2007). When is temporal planning really temporal? In *Proceedings of the 20th international joint conference on Artificial intelligence*, pages 1852–1859. Morgan Kaufmann Publishers Inc.
- [Dai et al., 2011] Dai, P., Weld, D. S., and Goldsmith, J. (2011). Topological value iteration algorithms. *Journal of Artificial Intelligence Research*, 42:181–209.
- [De Kleer et al., 1992] De Kleer, J., Mackworth, A. K., and Reiter, R. (1992). Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2):197–222.
- [De Kleer and Williams, 1987] De Kleer, J. and Williams, B. C. (1987). Diagnosing multiple faults. *Artificial intelligence*, 32(1):97–130.
- [Dechter et al., 1991] Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artificial intelligence*, 49(1):61–95.
- [Della Penna et al., 2012] Della Penna, G., Magazzeni, D., and Mercorio, F. (2012). A universal planning system for hybrid domains. *Applied intelligence*, 36(4):932–959.
- [Do and Kambhampati, 2003] Do, M. B. and Kambhampati, S. (2003). Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res.(JAIR)*, 20:155–194.
- [Dolgov and Durfee, 2005] Dolgov, D. A. and Durfee, E. H. (2005). Stationary deterministic policies for constrained mdps with multiple rewards, costs, and discount factors. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1326–1331.
- [Dornhege et al., 2012] Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., and Nebel, B. (2012). Semantic attachments for domain-independent planning systems. In *Towards service robots for everyday environments*, pages 99–115. Springer.
- [Effinger et al., 2009] Effinger, R., Williams, B., Kelly, G., and Sheehy, M. (2009). Dynamic Controllability of Temporally-flexible Reactive Programs. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 09)*.
- [Effinger, 2006] Effinger, R. T. (2006). Optimal Temporal Planning at Reactive Time Scales via Dynamic Backtracking Branch and Bound. Master’s thesis, Massachusetts Institute of Technology.
- [Effinger, 2012] Effinger, R. T. (2012). *Risk-minimizing program execution in robotic domains*. PhD thesis, Massachusetts Institute of Technology.

- [Fang, 2014] Fang, C. (2014). Mission-level planning with constraints on risk. Master’s thesis, Massachusetts Institute of Technology.
- [Fang et al., 2014] Fang, C., Yu, P., and Williams, B. C. (2014). Chance-constrained probabilistic simple temporal problems. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.
- [Fargier et al., 1995] Fargier, H., Lang, J., Martin-Clouaire, R., and Schiex, T. (1995). A constraint satisfaction framework for decision under uncertainty. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 167–174. Morgan Kaufmann Publishers Inc.
- [Fargier et al., 1996] Fargier, H., Lang, J., and Schiex, T. (1996). Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proceedings of the National Conference on Artificial Intelligence*, pages 175–180.
- [Feinberg and Shwarz, 1995] Feinberg, E. and Shwarz, A. (1995). Constrained Discounted Dynamic Programming. *Math. of Operations Research*, 21:922–945.
- [Fernández-González et al., 2015] Fernández-González, E., Karpas, E., and Williams, B. C. (2015). Mixed discrete-continuous heuristic generative planning based on flow tubes. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1565–1572. AAAI Press.
- [Fiacco and McCormick, 1964] Fiacco, A. V. and McCormick, G. P. (1964). The sequential unconstrained minimization technique for nonlinear programming, a primal-dual method. *Management Science*, 10(2):360–366.
- [Firby, 1990] Firby, R. J. (1990). *Adaptive execution in complex dynamic worlds*. PhD thesis, Citeseer.
- [Fox and Long, 2003] Fox, M. and Long, D. (2003). PDDL2. 1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.(JAIR)*, 20:61–124.
- [Fox and Long, 2006] Fox, M. and Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res.(JAIR)*, 27:235–297.
- [Fritz and McIlraith, 2005] Fritz, C. and McIlraith, S. (2005). Compiling qualitative preferences into decision-theoretic golog programs. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06)*.
- [Gat, 1997] Gat, E. (1997). ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Aerospace Conference, 1997. Proceedings., IEEE*, volume 1, pages 319–324. IEEE.

- [Geffner and Bonet, 2013] Geffner, H. and Bonet, B. (2013). A concise introduction to models and methods for automated planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(1):1–141.
- [Gelle and Sabin, 2006] Gelle, E. and Sabin, M. (2006). Solver framework for conditional constraint satisfaction problems. In *Proceeding of European Conference on Artificial Intelligence (ECAI-06) Workshop on Configuration*, pages 14–19.
- [Gerevini and Long, 2005] Gerevini, A. and Long, D. (2005). Plan constraints and preferences in PDDL3. *The Language of the Fifth International Planning Competition. Tech. Rep. Technical Report, Department of Electronics for Automation, University of Brescia, Italy*, 75.
- [Gregory et al., 2012] Gregory, P., Long, D., Fox, M., and Beck, J. C. (2012). Planning Modulo Theories: Extending the Planning Paradigm. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*.
- [Gupta et al., 1996] Gupta, V., Jagadeesan, R., and Saraswat, V. (1996). Models for concurrent constraint programming. In *CONCUR'96: Concurrency Theory*, pages 66–83. Springer.
- [Halbwachs, 1998] Halbwachs, N. (1998). Synchronous programming of reactive systems. In *International Conference on Computer Aided Verification*, pages 1–16. Springer.
- [Halbwachs et al., 1991] Halbwachs, N., Caspi, P., Raymond, P., and Pilaud, D. (1991). The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320.
- [Hansen, 1998] Hansen, E. A. (1998). Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 211–219. Morgan Kaufmann Publishers Inc.
- [Hansen and Zilberstein, 2001] Hansen, E. A. and Zilberstein, S. (2001). LAO^{*}: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1):35–62.
- [Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274.
- [Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107.
- [Hart et al., 2012] Hart, W. E., Laird, C., Watson, J.-P., and Woodruff, D. L. (2012). *Pyomo—optimization modeling in Python*, volume 67. Springer Science & Business Media.

- [Hebrard et al., 2010] Hebrard, E., O’Mahony, E., and O’Sullivan, B. (2010). Constraint programming and combinatorial optimisation in numberjack. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 181–185. Springer.
- [Hoeffding, 1963] Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30.
- [Hoffmann and Edelkamp, 2005] Hoffmann, J. and Edelkamp, S. (2005). The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579.
- [Hou et al., 2014] Hou, P., Yeoh, W., and Varakantham, P. (2014). Revisiting risk-sensitive mdps: New algorithms and results. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*.
- [Hou et al., 2016] Hou, P., Yeoh, W., and Varakantham, P. (2016). Solving Risk-Sensitive POMDPs With and Without Cost Observations. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*.
- [Howey et al., 2004] Howey, R., Long, D., and Fox, M. (2004). VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, pages 294–301. IEEE.
- [Hunsberger, 2009] Hunsberger, L. (2009). Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *2009 16th International Symposium on Temporal Representation and Reasoning*, pages 155–162. IEEE.
- [Hunsberger, 2010] Hunsberger, L. (2010). A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In *2010 17th International Symposium on Temporal Representation and Reasoning*, pages 121–128. IEEE.
- [Hunsberger, 2013] Hunsberger, L. (2013). A faster execution algorithm for dynamically controllable stnus. In *2013 20th International Symposium on Temporal Representation and Reasoning*, pages 26–33. IEEE.
- [Hunsberger, 2014] Hunsberger, L. (2014). A faster algorithm for checking the dynamic controllability of simple temporal networks with uncertainty. In *ICAART (1)*, pages 63–73.
- [Hunsberger et al., 2012] Hunsberger, L., Posenato, R., and Combi, C. (2012). The dynamic controllability of conditional stns with uncertainty. In *Workshop on Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx)*, pages 1–8.

- [Ingham et al., 2001] Ingham, M., Ragno, R., and Williams, B. C. (2001). A reactive model-based programming language for robotic space explorers. *Proceedings of ISAIRAS-01*.
- [Ingham, 2003] Ingham, M. D. (2003). *Timed model-based programming: Executable specifications for robust mission-critical sequences*. PhD thesis, Massachusetts Institute of Technology.
- [Isom et al., 2008] Isom, J. D., Meyn, S. P., and Braatz, R. D. (2008). Piecewise Linear Dynamic Programming for Constrained POMDPs. In *Proceedings 23rd AAAI Conference on Artificial Intelligence*, pages 291–296.
- [Ivankovic et al., 2014] Ivankovic, F., Haslum, P., Thiébaux, S., Shivashankar, V., and Nau, D. S. (2014). Optimal planning with global numerical state constraints. In *ICAPS*.
- [Jones et al., 01 | Jones, E., Oliphant, T., Peterson, P., et al. (2001–). SciPy: Open source scientific tools for Python. [Online; accessed 2016-06-06].
- [Kaelbling et al., 1998] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134.
- [Kaipa et al., 2015] Kaipa, K. N., Thevendria-Karthic, S. S., Shriyam, S., Kabir, A. M., Langsfeld, J. D., and Gupta, S. K. (2015). Resolving automated perception system failures in bin-picking tasks using assistance from remote human operators. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1453–1458. IEEE.
- [Karaman et al., 2011] Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S. (2011). Anytime motion planning using the rrt. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483. IEEE.
- [Karmarkar, 1984] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM.
- [Kautz and Selman, 2006] Kautz, H. and Selman, B. (2006). Satplan04: Planning as satisfiability. *Working Notes on the Fifth International Planning Competition (IPC-2006)*, pages 45–46.
- [Keller and Helmert, 2013] Keller, T. and Helmert, M. (2013). Trial-based Heuristic Tree Search for Finite Horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 135–143.
- [Kim et al., 2011] Kim, D., Lee, J., Kim, K., and Poupart, P. (2011). Point-Based Value Iteration for Constrained POMDPs. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1968–1974.

- [Kim et al., 2001] Kim, P., Williams, B., and Abramson, M. (2001). Executing reactive, model-based programs through graph-based temporal planning. In *IJCAI*.
- [Kocsis and Szepesvári, 2006] Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer.
- [Koenig and Howard, 2004] Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2149–2154. IEEE.
- [Kurniawati et al., 2008] Kurniawati, H., Hsu, D., and Lee, W. S. (2008). SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Robotics: Science and Systems*, volume 2008.
- [Lane, 2016] Lane, S. (2016). Propositional and activity monitoring using qualitative spatial reasoning. Master’s thesis, Massachusetts Institute of Technology.
- [Le Guernic et al., 1986] Le Guernic, P., Benveniste, A., Bournai, P., and Gautier, T. (1986). Signal—a data flow-oriented language for signal processing. *IEEE transactions on acoustics, speech, and signal processing*, 34(2):362–374.
- [Levesque et al., 1997] Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. B. (1997). GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1):59–83.
- [Levine and Williams, 2014] Levine, S. J. and Williams, B. C. (2014). Concurrent plan recognition and execution for human-robot teams. In *ICAPS-14*.
- [Li, 2010] Li, H. (2010). *Kongming: A Generative Planner for Hybrid Systems with Temporally Extended Goals*. PhD thesis, Massachusetts Institute of Technology.
- [Li and Williams, 2011] Li, H. and Williams, B. C. (2011). Hybrid Planning with Temporally Extended Goals for Sustainable Ocean Observing. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*.
- [Li and Williams, 2008] Li, H. X. and Williams, B. C. (2008). Generative planning for hybrid systems based on flow tubes. In *ICAPS*, pages 206–213.
- [Lin et al., 2016] Lin, C., Hadfield-Menell, D., Chitnis, R., Russell, S., and Abbeel, P. (2016). Sequential Quadratic Programming for Task Plan Optimization. In *PlanRob Workshop, 26th International Conference on Automated Planning and Scheduling*.
- [Lozano-Pérez and Kaelbling, 2014] Lozano-Pérez, T. and Kaelbling, L. P. (2014). A constraint-based method for solving sequential manipulation planning problems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3684–3691. IEEE.

- [Marecki and Varakantham, 2010] Marecki, J. and Varakantham, P. (2010). Risk-sensitive planning in partially observable environments. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1357–1368. International Foundation for Autonomous Agents and Multiagent Systems.
- [Mausam and Weld, 2008] Mausam and Weld, D. S. (2008). Planning with durative actions in stochastic domains. *J. Artif. Intell. Res. (JAIR)*, 31:33–82.
- [McDermott et al., 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL-the planning domain definition language.
- [McGhan and Murray, 2015] McGhan, C. L. and Murray, R. M. (2015). Application of correct-by-construction principles for a resilient risk-aware architecture. In *AIAA SPACE 2015 Conference and Exposition*, page 4527.
- [McGhan et al., 2015] McGhan, C. L. R., Murray, R. M., Serra, R., Ingham, M. D., Ono, M., Estlin, T., and Williams, B. C. (2015). A risk-aware architecture for resilient spacecraft operations. In *2015 IEEE Aerospace Conference*, pages 1–15.
- [Meuleau et al., 2009] Meuleau, N., Benazera, E., Brafman, R. I., Hansen, E. A., and Mausam, M. (2009). A heuristic search approach to planning with continuous resources in stochastic domains. *Journal of Artificial Intelligence Research*, 34(1):27.
- [Micheli et al., 2015] Micheli, A., Do, M., and Smith, D. E. (2015). Compiling away uncertainty in strong temporal planning with uncontrollable durations. In *International Joint Conference on Artificial Intelligence*.
- [Millman and Aivazis, 2011] Millman, K. J. and Aivazis, M. (2011). Python for scientists and engineers. *Computing in Science & Engineering*, 13(2):9–12.
- [Morris, 2006] Morris, P. (2006). A structural characterization of temporal dynamic controllability. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP-2006)*, pages 375–389.
- [Morris, 2014] Morris, P. (2014). Dynamic controllability and dispatchability relationships. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 464–479. Springer.
- [Morris and Muscettola, 2005] Morris, P. and Muscettola, N. (2005). Temporal dynamic controllability revisited. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 1193–1198. AAAI Press / The MIT Press.
- [Morris et al., 2001] Morris, P., Muscettola, N., and Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *IJCAI*, volume 1, pages 494–502.

- [Nieuwenhuis et al., 2006] Nieuwenhuis, R., Oliveras, A., and Tinelli, C. (2006). Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll (t). *Journal of the ACM (JACM)*, 53(6):937–977.
- [Nilsson, 1982] Nilsson, N. J. (1982). *Principles of artificial intelligence*. Springer.
- [Ono et al., 2012a] Ono, M. et al. (2012a). *Robust, goal-directed plan execution with bounded risk*. PhD thesis, Massachusetts Institute of Technology.
- [Ono et al., 2012b] Ono, M., Kuwata, Y., and Balaram, J. (2012b). Joint chance-constrained dynamic programming. In *CDC*, pages 1915–1922.
- [Ono et al., 2013] Ono, M., Williams, B., and Blackmore, L. (2013). Probabilistic planning for continuous dynamic systems under bounded risk. *JAIR*, 46:511–577.
- [Ono and Williams, 2008] Ono, M. and Williams, B. C. (2008). Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 3427–3432. IEEE.
- [Papadimitriou and Tsitsiklis, 1987] Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450.
- [Pearl, 1984] Pearl, J. (1984). Heuristics: intelligent search strategies for computer problem solving.
- [Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.
- [Peintner et al., 2007] Peintner, B., Venable, K. B., and Yorke-Smith, N. (2007). Strong controllability of disjunctive temporal problems with uncertainty. In *Principles and Practice of Constraint Programming–CP 2007*, pages 856–863. Springer.
- [Penberthy and Weld, 1994] Penberthy, J. S. and Weld, D. S. (1994). Temporal planning with continuous change. In *AAAI*, volume 94, page 1010.
- [Pineda et al., 2015] Pineda, L., Wray, K. H., and Zilberstein, S. (2015). Revisiting Multi-Objective MDPs with Relaxed Lexicographic Preferences. In *2015 AAAI Fall Symposium Series*.
- [Piotrowski et al., 2016] Piotrowski, W., Fox, M., Long, D., Magazzeni, D., and Mercurio, F. (2016). Heuristic Planning for PDDL+ Domains. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*.

- [Poupart et al., 2015] Poupart, P., Malhotra, A., Pei, P., Kim, K.-E., Goh, B., and Bowling, M. (2015). Approximate linear programming for constrained partially observable markov decision processes. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*.
- [Quigley et al., 2009] Quigley, M., Faust, J., Foote, T., and Leibs, J. (2009). ROS: an open-source Robot Operating System. Available online at <http://www.ros.org/>.
- [Sanner, 2010] Sanner, S. (2010). *Relational dynamic influence diagram language (RDDL): Language description*. NICTA & the Australian National University. Available online at http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf.
- [Santana et al., 2015] Santana, P., Lane, S., Timmons, E., Williams, B., and Forster, C. (2015). Learning hybrid models with guarded transitions. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*.
- [Santana et al., 2014] Santana, P., Lopes, R., Amui, B., Borges, G., ao Ishihara, J., Williams, B., and Forster, C. (2014). A new filter for hybrid systems and its applications to robust attitude estimation. In *Proceedings of the 53rd IEEE Conference on Decision and Control*.
- [Santana et al., 2016a] Santana, P., Lopes, R., Borges, G., and Williams, B. (2016a). A hybrid data fusion approach for robust attitude estimation. *Recent Advances on Multisensor Attitude Estimation: Fundamental Concepts and Applications*, ?(?):?–?
- [Santana et al., 2016b] Santana, P., Thiébaux, S., and Williams, B. (2016b). RAO*: an Algorithm for Chance-Constrained POMDP’s. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*.
- [Santana et al., 2016c] Santana, P., Vaquero, T., Toledo, C., Wang, A., Fang, C., and Williams, B. (2016c). PARIS: a Polynomial-Time, Risk-Sensitive Scheduling Algorithm for Probabilistic Simple Temporal Networks with Uncertainty. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*.
- [Santana and Williams, 2014] Santana, P. and Williams, B. (2014). Chance-constrained consistency for probabilistic temporal plan networks. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling*.
- [Shah et al., 2009] Shah, J. A., Conrad, P. R., and Williams, B. C. (2009). Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Thessaloniki, Greece.
- [Shah et al., 2007] Shah, J. A., Stedl, J., Williams, B. C., and Robertson, P. (2007). A fast incremental algorithm for maintaining dispatchability of partially controllable plans. In *ICAPS*, pages 296–303.

- [Shah and Williams, 2008] Shah, J. A. and Williams, B. C. (2008). Fast dynamic scheduling of disjunctive temporal constraint networks through incremental compilation. In *ICAPS*, pages 322–329.
- [Shin and Davis, 2005] Shin, J.-A. and Davis, E. (2005). Processes and continuous change in a SAT-based planner. *Artificial Intelligence*, 166(1):194–253.
- [Shu et al., 2005] Shu, I.-h., Effinger, R. T., and Williams, B. C. (2005). Enabling fast flexible planning through incremental temporal reasoning with conflict extraction. In *ICAPS*, pages 252–261.
- [Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.
- [Silver and Veness, 2010] Silver, D. and Veness, J. (2010). Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, pages 2164–2172.
- [Simmons and Apfelbaum, 1998] Simmons, R. and Apfelbaum, D. (1998). A task description language for robot control. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 3, pages 1931–1937. IEEE.
- [Smallwood and Sondik, 1973] Smallwood, R. and Sondik, E. (1973). The optimal control of partially observable markov decision processes over a finite horizon. *Operations Research*, 21(5):1071–1088.
- [Smith et al., 2008] Smith, D. E., Frank, J., and Cushing, W. (2008). The ANML language. In *The ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.
- [Smith and Weld, 1998] Smith, D. E. and Weld, D. S. (1998). Conformant Graphplan. In *AAAI/IAAI*, pages 889–896.
- [Smith and Weld, 1999] Smith, D. E. and Weld, D. S. (1999). Temporal planning with mutual exclusion reasoning. In *IJCAI*, volume 99, pages 326–337.
- [Sprauel et al., 2014] Sprauel, J., Kolobov, A., and Teichteil-Königsbuch, F. (2014). Saturated path-constrained mdp: Planning under uncertainty and deterministic model-checking constraints. In *AAAI*, pages 2367–2373.
- [Srivastava et al., 2014] Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 639–646. IEEE.

- [Stallman and Sussman, 1977] Stallman, R. M. and Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial intelligence*, 9(2):135–196.
- [Stergiou and Koubarakis, 2000] Stergiou, K. and Koubarakis, M. (2000). Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117.
- [Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.
- [Szer et al., 2005] Szer, D., Charpillet, F., and Zilberstein, S. (2005). MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 576–583, Edinburgh, Scotland.
- [Tarim et al., 2006] Tarim, S. A., Manandhar, S., and Walsh, T. (2006). Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80.
- [Tarjan, 1972] Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160.
- [Teichteil-Königsbuch, 2012] Teichteil-Königsbuch, F. (2012). Path-Constrained Markov Decision Processes: bridging the gap between probabilistic model-checking and decision-theoretic planning. In *ECAI*, pages 744–749.
- [Thiébaux and Cordier, 2001] Thiébaux, S. and Cordier, M.-O. (2001). Supply restoration in power distribution systems — a benchmark for planning under uncertainty. In *Proc. 6th European Conference on Planning (ECP)*, pages 85–95.
- [Timmons, 2013] Timmons, E. (2013). Fast, approximate state estimation of concurrent probabilistic hybrid automata. Master’s thesis, Massachusetts Institute of Technology.
- [Timmons et al., 2015] Timmons, E., Fang, C., Fernandez, E., Karpas, E., Levine, S., Santana, P., Wang, A., Wang, D., Yu, P., and Williams, B. (2015). Reactive Model-based Programming of Micro-UAVs. In *The ICAPS-15 ISF Research Workshops*.
- [Toussaint, 2015] Toussaint, M. (2015). Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 1930–1936. AAAI Press.
- [Trevizan et al., 2016] Trevizan, F., Thiébaux, S., Santana, P., and Williams, B. (2016). Heuristic Search in Dual Space for Constrained Stochastic Shortest Path Problems. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*.

- [Tsamardinos, 2002] Tsamardinos, I. (2002). A probabilistic approach to robust execution of temporal plans with uncertainty. In *Methods and Applications of Artificial Intelligence*, pages 97–108. Springer.
- [Tsamardinos et al., 2001] Tsamardinos, I., Pollack, M. E., and Ganchev, P. (2001). Flexible dispatch of disjunctive plans. In *Sixth European Conference on Planning*.
- [Tsamardinos et al., 2003] Tsamardinos, I., Vidal, T., and Pollack, M. (2003). CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388.
- [Undurti, 2011] Undurti, A. (2011). *Planning under uncertainty and constraints for teams of autonomous agents*. PhD thesis, Massachusetts Institute of Technology.
- [Undurti and How, 2010] Undurti, A. and How, J. P. (2010). An online algorithm for constrained POMDPs. In *IEEE International Conference on Robotics and Automation*, pages 3966–3973.
- [van der Walt et al., 2011] van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30.
- [Venable et al., 2010] Venable, K. B., Volpato, M., Peintner, B., and Yorke-Smith, N. (2010). Weak and dynamic controllability of temporal problems with disjunctions and uncertainty. In *Workshop on Constraint Satisfaction Techniques for Planning & Scheduling*, pages 50–59.
- [Venable and Yorke-Smith, 2005] Venable, K. B. and Yorke-Smith, N. (2005). Disjunctive temporal planning with uncertainty. In *IJCAI*, pages 1721–1722.
- [Vidal, 1999] Vidal, T. (1999). Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45.
- [Vidal and Ghallab, 1996] Vidal, T. and Ghallab, M. (1996). Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *ECAI*, pages 48–54.
- [Vlasic and Boudette, 2016] Vlasic, B. and Boudette, N. (2016). As U.S. Investigates Fatal Tesla Crash, Company Defends Autopilot System. *The New York Times*. Available online at <http://www.nytimes.com/2016/07/13/business/tesla-autopilot-fatal-crash-investigation.html>. Accessed on July 14th, 2016.
- [Wächter and Biegler, 2006] Wächter, A. and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57.

- [Walcott, 2004] Walcott, A. (2004). Unifying Model-Based Programming and Path Planning Through Optimal Search. Master’s thesis, Massachusetts Institute of Technology.
- [Wang, 2013] Wang, A. J. (2013). Risk allocation for temporal risk assessment. Master’s thesis, Massachusetts Institute of Technology.
- [Wang and Williams, 2015a] Wang, A. J. and Williams, B. C. (2015a). Chance-constrained scheduling via conflict-directed risk allocation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*.
- [Wang, 2015] Wang, D. (2015). *A Factored Planner for the Temporal Coordination of Autonomous Systems*. PhD thesis, Massachusetts Institute of Technology.
- [Wang and Williams, 2015b] Wang, D. and Williams, B. (2015b). tBurton: A Divide and Conquer Temporal Planner. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*.
- [Washington, 1996] Washington, R. (1996). Incremental markov-model planning. In *Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on*, pages 41–47. IEEE.
- [Washington, 1997] Washington, R. (1997). BI-POMDP: Bounded, incremental partially-observable Markov-model planning. In *European Conference on Planning*, pages 440–451. Springer.
- [Williams et al., 2001] Williams, B. C., Chung, S., and Gupta, V. (2001). Mode estimation of model-based programs: monitoring systems with complex behavior. In *IJCAI*, pages 579–590.
- [Williams and Ingham, 2002] Williams, B. C. and Ingham, M. D. (2002). Model-based programming: Controlling embedded systems by reasoning about hidden state. In *Principles and Practice of Constraint Programming-CP 2002*, pages 508–524. Springer.
- [Williams et al., 2003] Williams, B. C., Ingham, M. D., Chung, S. H., and Elliott, P. H. (2003). Model-based programming of intelligent embedded systems and robotic space explorers. *Proceedings of the IEEE*, 91(1):212–237.
- [Williams and Ragno, 2007] Williams, B. C. and Ragno, R. J. (2007). Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathematics*, 155(12):1562–1595.
- [Wray and Zilberstein, 2015] Wray, K. H. and Zilberstein, S. (2015). Multi-objective POMDPs with lexicographic reward preferences. In *Proceedings of the 24th International Joint Conference of Artificial Intelligence (IJCAI)*, pages 1719–1725.

- [Wray et al., 2015] Wray, K. H., Zilberstein, S., and Mouaddib, A.-I. (2015). Multi-Objective MDPs with Conditional Lexicographic Reward Preferences. In *AAAI*, pages 3418–3424.
- [Younes and Littman, 2004] Younes, H. L. and Littman, M. L. (2004). PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*.
- [Yu et al., 2015] Yu, P., Fang, C., and Williams, B. C. (2015). Resolving over-constrained probabilistic temporal problems through chance constraint relaxation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, Austin, TX.
- [Yu et al., 1998] Yu, S., Lin, Y., and Yan, P. (1998). Optimization models for the first arrival target distribution function in discrete time. *Journal of mathematical analysis and applications*, 225(1):193–223.
- [Ziafati et al., 2012] Ziafati, P., Dastani, M., Meyer, J.-J., and van der Torre, L. (2012). Agent programming languages requirements for programming autonomous robots. In *Programming Multi-Agent Systems*, pages 35–53. Springer.
- [Zimmerman et al., 2011] Zimmerman, R. D., Murillo-Sánchez, C. E., and Thomas, R. J. (2011). Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *Power Systems, IEEE Transactions on*, 26(1):12–19.