HEALTH-AWARE DECISION MAKING UNDER UNCERTAINTY FOR COMPLEX SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
AND THE COMMITTEE ON GRADUATE STUDIES OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Edward Balaban
June 2020
This dissertation is online at: http://purl.stanford.edu/mz480yb6491

© 2020 by Edward Balaban. All Rights Reserved.
Re-distributed by Stanford University under license with the author.

This work is licensed under a Creative Commons Attribution-Noncommercial 3.0 United States License.
http://creativecommons.org/licenses/by-nc/3.0/us/

This dissertation is online at: http://purl.stanford.edu/mz480yb6491
I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mykel Kochenderfer, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Juan Alonso

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mac Schwager

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.
Abstract

Aerospace vehicles, particularly spacecraft, often operate in harsh and uncertain environments, where decisions critical to mission success may need to be made quickly and with incomplete information. This is especially true when the vehicle experiences component faults or failures. The field of system health management has evolved over the last several decades from simple automated alarms to sophisticated artificial intelligence algorithms designed to analyze such off-nominal conditions and generate appropriate responses. This evolution took place largely apart from the development of automated system control, planning, and scheduling (referred to collectively as decision making). While there have been efforts to establish information exchange between system health management and decision making, successful practical implementations of integrated architectures have remained rare.

This thesis consists of three major parts. In the first part, the limitations of the currently prevalent system health management methodology are described and illustrated through numerical examples. In particular, prognostics (a relatively recent addition to the field of system health management) is shown to be meaningful only in a narrow subset of applications and, even then, challenging to implement in an effective manner. Instead, an approach is proposed that unifies system health management and operational decision making in their formulations in order to overcome these shortcomings. The thesis discusses implementation details of the new approach—referred to as Health Aware Decision Making, HADM— and provides an analysis of its computational complexity.

One of the key ingredients for successfully implementing HADM for realistic systems operating in harsh and uncertain environments is availability of decision making
algorithms that can reason over large, continuously valued action and observation domains. The second part of the thesis describes an algorithm developed for this category of problems, Large Problem Decision Making (LPDM). The algorithm is based on Determinized Sparse Partially Observable Trees (DESPOT), a state-of-the-art solver for problems formulated as partially observable Markov decision processes (POMDPs). LPDM incorporates novel methods for handling complex model spaces and is shown to outperform both the original DESPOT and a version of DESPOT augmented with the Blind Value algorithm (a recent method of handling large, continuously valued action spaces) on benchmarking problems.

The third major part of the thesis applies the methodology and the algorithms developed in the first two parts to create an advanced decision support system for space missions: System Health Enabled Realtime Planning Advisor (SHERPA). SHERPA is designed to be model-based, modular, and adaptable to different use cases throughout the lifetime of a mission. The system is targeted for first use on a NASA robotic rover mission to the Moon, scheduled for launch in 2023. The mission, Volatiles Investigating Polar Exploration Rover (VIPER), intends to land the solar-powered rover in a lunar polar region and use it to characterize the distribution of water ice and other volatiles in preparation for establishing a permanent human base. The thesis describes in detail one SHERPA use case developed for the VIPER mission. In the use case, focused on rover traverse evaluation and refinement, a traverse template is provided to SHERPA specifying the science activities to be performed at an ordered set of waypoints. SHERPA uses mission simulations with optimized action selection to evaluate the robustness of the proposed template to uncertainties that are likely to be a factor during the mission, then recommends a schedule of battery recharge periods that maximizes the chances of a successful traverse. Another use case, currently under development, generates a full traverse for the VIPER rover taking only the high-level mission objectives and constraints as inputs. The latter use case will also form the foundation for SHERPA’s landing site selection and vehicle parameter optimization capabilities.
Acknowledgments

Many people have, in various ways, helped me throughout this Ph.D. journey. I am glad to have an opportunity to express my sincere gratitude to them here.

First of all, I would like to thank my advisor, Mykel Kochenderfer. Your productivity, passion for your work, and technical erudition are, by now, legendary. Yet what stood out even more for me is how much you cared about everyone in your lab being successful and how you always made yourself available for a brainstorming session, a paper review or just a chat, no matter how many other things you had on your plate. Thank you for your unfailing support throughout this process and for your friendship. I hope to continue our collaboration for years to come.

I owe a special debt of gratitude to Juan Alonso, my first advisor and a member of the reading committee. When I was academically “homeless”, Juan welcomed me into his lab, even though he was already advising dozens of students and even though the research topic that I had in mind did not fall neatly within the existing areas of research in the lab. Thank you for your kindness, your generosity with time and advice, and your ever-present optimism. I hope we will have other opportunities to work together in the future.

I would also like to thank the other members of my reading and defense committees — Mac Schwagger, Clark Barrett, and Kyle Wray — for your insightful comments and attention to detail. You have undoubtedly helped make this dissertation better.

Stanford Intelligent Systems Laboratory is one of the friendliest, most creative and collaborative research environments I have ever encountered. I am already looking back with some nostalgia on our early debates about POMDPs.jl in the “smoke-filled rooms” and over long email exchanges. Zach Sunberg, Eric Mueller, Tim Wheeler,
Louis Dressel, Rachael Tompa, Youngjun Kim, Zouhair Mahboubi, Jon Cox, Max Egorov—I enjoyed interacting with all of you immensely and hope our professional paths will keep intersecting.

NASA has been a professional home for me for a number of years now. I feel privileged to be affiliated with an organization with such a storied history and with some of the best and brightest employees on the planet. I am also very grateful for all the opportunities I have gotten along the way, including the opportunity to pursue this degree. I would like to thank NASA Ames Center Director Eugene Tu for his support and for putting in place policies that made it easier for people like me to get their advanced degrees while working full time. I am also deeply thankful to the Director of Exploration Systems Rupak Biswas, Ames Chief Scientist Jacob Cohen and (now retired) Associate Center Director Steven Zornetzer for their support, encouragement, friendship, and help in my periodic searches for funding.

Many thanks to my friends and colleagues in the NASA Ames Intelligent Systems Division and beyond, particularly to Howard Cannon, Tony Colaprete, Greg Dorais, George Gorospe, Sonie Lau, Ritchie Lee, Molly O’Connor, Greg Ortzeg, Mark Shirley, and Ryan Vaughan. I am grateful for your help with proofreading, SHERPA, rover hardware, dealing with paperwork, obtaining funding, and numerous other things. Dave Iverson, Chetan Kulkarni, Rodney Martin, Indranil Roychoudhury, Lilly Spirkovska—thank you not just for the professional collaborations we have had over the years and for your assistance with this work, but also for being there through the various highs and lows of life in general.

Thank you to the SHERPA interns (Tomer Arnon, Saylor Brisson, Jennifer Chang, Sarah Feng, Alex Gao, Miguel Garcia, Leyton Ho, Logan Kilpatrick, Chandler Kilpatrick, and Nobel Truong) who have contributed to many aspects of the framework, including to the various incarnations of our visualization tools, the testing infrastructure, documentation, some of the rule-based policies and use cases, as well as to some of our more experimental, “skunkworks” initiatives. Because of your enthusiasm and efforts, SHERPA is a much more polished system than it would have been otherwise.

One of the things I am most grateful to Stanford for are the friendships that essentially transcended into family ties. Mumtaz Vauhkonen, Quaizar Vohra, and
Saurabh Madaan—I am as thankful for your camaraderie during our derivations of pages upon pages of machine learning math as I am for your wisdom on how to pacify a newborn baby. Josh Schoolcraft, thank you for making the drives from Los Angeles for our study sessions. I am fairly certain that you have set some sort of a record for the longest study session commute. I am also fairly certain that your vintage Subaru left quite a few parts behind on Interstate 5. Morgan Cable, thank you for always cheering us on even though you had your own papers and reports to worry about. Moran Bercovici, Betty Livshitz, William Wolf, and Betania Carvalho—toda raba and muito obrigado for everything. Even though we ended up on different continents, this does not change a thing. Looking forward to many more decades of family get-togethers, professional collaborations, and vigorous debates on Frank Zappa’s impact on human civilization.

Kathleen and Pablo, Kathy, Ralph, and Mira, and the seafaring Bozina family—thank you for making California feel like a true home. Many thanks to my parents-in-law not just for all your help, but, even more importantly, for your warmth.

I hold my parents and grandparents responsible for setting me on this path by always encouraging my curiosity, filling our home with books, and giving me the freedom to pursue my interests. While there is no adequate way to do it in a couple of sentences, thank you for all your efforts, for all your sacrifices over the years, and for somehow always managing to provide a happy, supportive environment to grow up in despite the often challenging external circumstances. To my brother Leo and his family, thank you for always having my back. Suzy and Bob, thank you for being in our lives and for everything you have done.

To my daughter Lucia and son Ari, thank you for always insisting on helping with my typing (in retrospect, maybe I would have finished a little sooner had I taken you up on your offers) and for never failing to brighten my day with your joyful faces and wild antics. It is now impossible to imagine my life without you two.

Most of all, however, I owe a huge debt of gratitude to my wife Leticia. Without your love, patience, support through all the ups and downs, and shouldering a disproportionate amount of child-rearing, none of this would have been possible. This thesis is as much yours as it is mine. Te amo mucho.
# List of Algorithms

5.1 Blind Value algorithm .................................................. 79
5.2 AAS algorithm ........................................................... 84
5.3 TCC algorithm ........................................................... 86
5.4 TCC cluster assignment procedure .................................. 87
5.5 TCC belief creation procedure ...................................... 88
5.6 LPDM algorithm .......................................................... 90
5.7 LPDM tree exploration procedure ................................... 91
5.8 LPDM tree expansion procedure ................................... 92
5.9 LPDM tree backup procedure ....................................... 92
5.10 AAS sampling procedure for Light Dark 1D ................. 95
5.11 AAS sampling algorithm for Light Dark 2D ................... 96
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAS</td>
<td>Adaptive Action Selection</td>
<td>83</td>
</tr>
<tr>
<td>AIM</td>
<td>area of interest mapping</td>
<td>121</td>
</tr>
<tr>
<td>AR-DESPOT</td>
<td>Anytime Regularized DESPOT</td>
<td>88</td>
</tr>
<tr>
<td>BV</td>
<td>Blind Value</td>
<td>78</td>
</tr>
<tr>
<td>CC-POMDP</td>
<td>Chance-Constrained Partially Observable Markov Decision Processes</td>
<td>111</td>
</tr>
<tr>
<td>CIVA</td>
<td>Conditionally Irrelevant Variable Abstraction</td>
<td>111</td>
</tr>
<tr>
<td>CLARK</td>
<td>Conditional Planning for Autonomy with Risk</td>
<td>111</td>
</tr>
<tr>
<td>CSP</td>
<td>constraint satisfaction problem</td>
<td>60</td>
</tr>
<tr>
<td>Dec-POMDP</td>
<td>decentralized partially observable Markov decision process</td>
<td>33</td>
</tr>
<tr>
<td>DEM</td>
<td>digital elevation map</td>
<td>118</td>
</tr>
<tr>
<td>DESPOT</td>
<td>Determinized Sparse Partially Observable Trees</td>
<td>30</td>
</tr>
<tr>
<td>DM</td>
<td>decision making</td>
<td>3</td>
</tr>
<tr>
<td>DMU</td>
<td>decision making under uncertainty</td>
<td>15</td>
</tr>
<tr>
<td>DPW</td>
<td>Double Progressive Widening</td>
<td>78</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>DSN</td>
<td>Deep Space Network</td>
<td>106</td>
</tr>
<tr>
<td>EoL</td>
<td>end of [useful] life</td>
<td>37</td>
</tr>
<tr>
<td>FRTDP</td>
<td>Focused Real-Time Dynamic Programming</td>
<td>111</td>
</tr>
<tr>
<td>FSA</td>
<td>Fast Simulated Annealing</td>
<td>9</td>
</tr>
<tr>
<td>FSLD</td>
<td>Fully Stochastic Light-Dark</td>
<td>93</td>
</tr>
<tr>
<td>HADM</td>
<td>Health-Aware Decision Making</td>
<td>8</td>
</tr>
<tr>
<td>HBSS</td>
<td>Heuristic-Biased Stochastic Sampling</td>
<td>110</td>
</tr>
<tr>
<td>ISHM</td>
<td>Integrated System Health Management</td>
<td>4</td>
</tr>
<tr>
<td>LD</td>
<td>Light-Dark</td>
<td>97</td>
</tr>
<tr>
<td>LPDM</td>
<td>Large Problem Decision Making</td>
<td>9</td>
</tr>
<tr>
<td>MCTS</td>
<td>Monte Carlo Tree Search</td>
<td>25</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov decision process</td>
<td>23</td>
</tr>
<tr>
<td>MOMDP</td>
<td>mixed observability Markov decision process</td>
<td>27</td>
</tr>
<tr>
<td>MPOMDP</td>
<td>multi-agent partially observable Markov decision process</td>
<td>33</td>
</tr>
<tr>
<td>MSolo</td>
<td>Mass Spectrometer Observing Lunar Operations</td>
<td>119</td>
</tr>
<tr>
<td>MUSE</td>
<td>Multi-User Scheduling Environment</td>
<td>108</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
<td>5</td>
</tr>
<tr>
<td>NIRVSS</td>
<td>Near-Infrared Volatile Spectrometer System</td>
<td>119</td>
</tr>
<tr>
<td>NSS</td>
<td>Neutron Spectrometer System</td>
<td>119</td>
</tr>
<tr>
<td>OpenSPIFe</td>
<td>Open Scheduling and Planning Interface for Exploration</td>
<td>108</td>
</tr>
<tr>
<td>PHM</td>
<td>Prognostics and Health Management</td>
<td>37</td>
</tr>
<tr>
<td>POMCP</td>
<td>Partially Observable Monte Carlo Planning</td>
<td>29</td>
</tr>
<tr>
<td>POMCPOW</td>
<td>Partially Observable Monte Carlo Planning with Observation Widening</td>
<td>82</td>
</tr>
<tr>
<td>POMDP</td>
<td>partially observable Markov decision process</td>
<td>9</td>
</tr>
<tr>
<td>PSR</td>
<td>permanently shadowed region</td>
<td>117</td>
</tr>
<tr>
<td>PV</td>
<td>photo-voltaic</td>
<td>118</td>
</tr>
<tr>
<td>RP</td>
<td>Resource Prospector</td>
<td>105</td>
</tr>
<tr>
<td>RUL</td>
<td>remaining useful life</td>
<td>37</td>
</tr>
<tr>
<td>RWDU</td>
<td>Regularized Weighted Discounted Utility</td>
<td>88</td>
</tr>
<tr>
<td>SER</td>
<td>System Emergency Response</td>
<td>61</td>
</tr>
<tr>
<td>SEXTANT</td>
<td>Surface Exploration Traverse Analysis and Navigation Tool</td>
<td>110</td>
</tr>
<tr>
<td>SHERPA</td>
<td>System Health Enabled Realtime Planning Adviser</td>
<td>10</td>
</tr>
<tr>
<td>SHM</td>
<td>system health management</td>
<td>3</td>
</tr>
<tr>
<td>SoC</td>
<td>state of charge</td>
<td>131</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>TCC</td>
<td>Transition Correlation Clustering</td>
<td>85</td>
</tr>
<tr>
<td>TRIDENT</td>
<td>The Regolith and Ice Drill for Exploring New Terrain</td>
<td>119</td>
</tr>
<tr>
<td>TS</td>
<td>Traverse Synthesis</td>
<td>154</td>
</tr>
<tr>
<td>UCB</td>
<td>Upper Confidence Bound</td>
<td>28</td>
</tr>
<tr>
<td>UCT</td>
<td>Upper Confidence Trees</td>
<td>29</td>
</tr>
<tr>
<td>VIPER</td>
<td>Volatiles Investigating Polar Exploration Rover</td>
<td>10</td>
</tr>
<tr>
<td>xGDS</td>
<td>Exploration Ground Data Systems</td>
<td>109</td>
</tr>
</tbody>
</table>
## Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>action</td>
<td>17</td>
</tr>
<tr>
<td>$A$</td>
<td>action space</td>
<td>17</td>
</tr>
<tr>
<td>$b$</td>
<td>belief state</td>
<td>20</td>
</tr>
<tr>
<td>$b_0$</td>
<td>initial belief</td>
<td>60</td>
</tr>
<tr>
<td>$b'$</td>
<td>next belief</td>
<td>26</td>
</tr>
<tr>
<td>$D$</td>
<td>policy tree depth (equivalent to the planning</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>horizon $H$ in most cases)</td>
<td></td>
</tr>
<tr>
<td>$\Phi$</td>
<td>set of DESPOT/LPDM scenarios</td>
<td>31</td>
</tr>
<tr>
<td>$\phi$</td>
<td>DESPOT/LPDM scenario</td>
<td>31</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>MDP/POMDP reward discount factor</td>
<td>23</td>
</tr>
<tr>
<td>$G$</td>
<td>generative model (simulator)</td>
<td>28</td>
</tr>
<tr>
<td>$\mathcal{H}$</td>
<td>system health metric</td>
<td>39</td>
</tr>
<tr>
<td>$h$</td>
<td>history</td>
<td>18</td>
</tr>
<tr>
<td>$H$</td>
<td>decision (planning) horizon</td>
<td>22</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>$K$</td>
<td>number of pre-sampled DESPOT/LPDM scenarios (also number of root node belief state particles)</td>
<td>31</td>
</tr>
<tr>
<td>$M$</td>
<td>system model</td>
<td>54</td>
</tr>
<tr>
<td>$o$</td>
<td>observation</td>
<td>18</td>
</tr>
<tr>
<td>$O$</td>
<td>observation space</td>
<td>18</td>
</tr>
<tr>
<td>$\pi$</td>
<td>policy</td>
<td>22</td>
</tr>
<tr>
<td>$\pi^*$</td>
<td>optimal policy</td>
<td>22</td>
</tr>
<tr>
<td>$\Pi$</td>
<td>space/set of policies</td>
<td>24</td>
</tr>
<tr>
<td>$Q$</td>
<td>state-action (or belief-action) utility function</td>
<td>22</td>
</tr>
<tr>
<td>$Q^*$</td>
<td>optimal state-action (or belief-action) utility function</td>
<td>22</td>
</tr>
<tr>
<td>$r$</td>
<td>reward</td>
<td>19</td>
</tr>
<tr>
<td>$R$</td>
<td>reward model</td>
<td>19</td>
</tr>
<tr>
<td>$s$</td>
<td>state</td>
<td>17</td>
</tr>
<tr>
<td>$s_0$</td>
<td>initial state</td>
<td>39</td>
</tr>
<tr>
<td>$s'$</td>
<td>next state</td>
<td>17</td>
</tr>
<tr>
<td>$S$</td>
<td>state space</td>
<td>17</td>
</tr>
<tr>
<td>$S_T$</td>
<td>terminal states</td>
<td>23</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>$\tau$</td>
<td>temperature parameter in the Adaptive Action Selection algorithm</td>
<td>84</td>
</tr>
<tr>
<td>$T$</td>
<td>state transition model</td>
<td>19</td>
</tr>
<tr>
<td>$U$</td>
<td>utility function</td>
<td>22</td>
</tr>
<tr>
<td>$U^*$</td>
<td>optimal utility function</td>
<td>22</td>
</tr>
<tr>
<td>$V$</td>
<td>value function</td>
<td>22</td>
</tr>
<tr>
<td>$V^*$</td>
<td>optimal value function</td>
<td>22</td>
</tr>
<tr>
<td>$\xi$</td>
<td>model uncertainty coefficient</td>
<td>94</td>
</tr>
<tr>
<td>$Z$</td>
<td>observation model</td>
<td>19</td>
</tr>
</tbody>
</table>
Contents

Abstract iv

Acknowledgments vi

Acronyms x

Notation xiv

1 Introduction 1

1.1 Scope ............................................. 2
1.2 Definitions ........................................ 3
1.3 Background ........................................ 3
1.4 Motivation ......................................... 5
1.5 Challenges ........................................ 7
1.6 Limitations of Current Methods ..................... 7
1.7 Approach .......................................... 8
1.8 Contributions ...................................... 10
  1.8.1 Theory ......................................... 11
  1.8.2 Algorithms .................................... 11
  1.8.3 Application .................................... 12
1.9 Organization ...................................... 13

2 Decision Making under Uncertainty 15

2.1 Introduction ....................................... 16
2.1.1 States ................................................................. 17
2.1.2 Actions .............................................................. 17
2.1.3 Observations ....................................................... 18
2.1.4 History .............................................................. 18
2.1.5 Models .............................................................. 19
2.1.6 Types of Uncertainty .......................................... 20
2.1.7 Utility ............................................................... 22
2.1.8 Sequential Decision Making ............................... 22
2.2 Markov Decision Processes ........................................ 23
2.3 Partially Observable Markov Decision Processes ............. 25
2.4 Mixed Observability Markov Decision Processes ............. 27
2.5 Modeling for MDP and POMDP Problems ..................... 27
2.6 Monte Carlo Tree Search .......................................... 28
2.7 Partially Observable Monte Carlo Planning .................... 29
2.8 Determinized Sparse Partially Observable Trees ................ 30
2.9 Belief Updating ..................................................... 32
2.10 Discussion .......................................................... 32

3 System Health Management ................................. 34
  3.1 Introduction ........................................................ 34
  3.2 Contemporary SHM methodology ............................ 36
  3.3 General Issues .................................................... 38
    3.3.1 Incompatible DM and SHM objectives .................... 39
    3.3.2 Separated System Models ................................ 41
  3.4 Prognostics ....................................................... 43
    3.4.1 Definition ................................................. 43
    3.4.2 Systems with Uncontrolled Degradation Processes .... 43
    3.4.3 Systems with Controllable Degradation Processes .... 50
  3.5 Discussion ........................................................ 53

4 Health-Aware Decision Making ............................. 55
  4.1 Introduction ........................................................ 55
6.3 Architecture ................................................. 111
   6.3.1 Solvers/Policies ........................................ 112
   6.3.2 Support Infrastructure .................................. 113
   6.3.3 Missions ................................................ 113
   6.3.4 Use Cases ................................................ 114
   6.3.5 Fault Handling .......................................... 114
6.4 VIPER Mission ............................................. 116
   6.4.1 Candidate Operating Areas ............................... 116
   6.4.2 Scientific Objectives ................................... 117
   6.4.3 Vehicle .................................................. 118
   6.4.4 Science Payload ......................................... 118
   6.4.5 Mission Rules .......................................... 120
   6.4.6 Information Available to SHERPA ....................... 120
6.5 VIPER Mission Modeling ................................. 124
   6.5.1 State Vector ........................................... 124
   6.5.2 Observation Vector ..................................... 126
   6.5.3 Generative Model ...................................... 126
6.6 Traverse Evaluation and Refinement Use Case .... 128
   6.6.1 Modeling ............................................... 129
   6.6.2 Rule-Based Policies .................................... 131
   6.6.3 Prognostic Policy ...................................... 132
   6.6.4 Value Bound Algorithms ................................. 133
   6.6.5 Metrics ................................................ 133
6.7 Experiments ................................................ 133
   6.7.1 Setup .................................................. 134
   6.7.2 Results ................................................ 139
6.8 Discussion ................................................ 143

7 Conclusions ................................................. 147
   7.1 Summary and Contributions ............................... 147
   7.1.1 Theory ................................................ 148
List of Tables

4.1 HADM computational complexity increase summary 72
5.1 Common parameters for Light-Dark experiments 97
5.2 AR-DESPOT action spaces for Light-Dark experiments 98
6.2 An example activity dictionary 121
6.3 Lunar ice stability depth zone types 122
6.6 SHERPA TER performance metrics 134
6.7 Common parameters for the SHERPA TER experiments 135
A.1 Solver benchmarking results, partially stochastic Light-Dark 1D 162
A.2 Solver benchmarking results, partially stochastic Light-Dark 2D 163
A.3 Solver benchmarking results, fully stochastic Light-Dark 1D 164
A.4 Solver benchmarking results, fully stochastic Light-Dark 2D 165
A.5 Model uncertainty experiment results, Light-Dark 1D 166
A.6 Model uncertainty experiment results, Light-Dark 2D 167
B.1 Rover parameters used in the SHERPA models 169
B.2 Uncertainties modeled for the VIPER mission 170
B.3 SHERPA fault accommodation experiment parameters 170
C.1 Recoverable faults and delays experiment results, TER use case 172
C.2 Recoverable faults and delays experiment results, TER use case (100 particle beliefs) 173
C.3 Fault accommodation experiment results, TER use case 174
C.4 Model uncertainty experiment results, TER use case,
    DESPOT (EA, DO) . . . . . . . . . . . . . . . . . . . . . . . . . . . 175
C.5 Model uncertainty experiment results, TER use case,
    LPDM (AAS, TCC) . . . . . . . . . . . . . . . . . . . . . . . . . . . 176
List of Figures

3.1 A typical system architecture with SHM .......................... 37
3.2 A typical contemporary SHM architecture ........................ 38
3.3 Balancing SHM and DM objectives (Example 3.1) ............ 39
3.4 Execution scenarios in Example 3.1 ................................. 40
3.5 Separated versus unified SHM/DM models (Example 3.2) ..... 42
3.6 An EoL prediction illustration, with an overlay showing the $p(\text{EoL})$ distribution ................................................. 44
3.7 Prognostics of an uncontrolled degradation process with deterministic and stochastic models (Example 3.3) ................. 44
3.8 PHM vs. DM for a controlled system (Example 3.4) .......... 51
3.9 Execution scenarios in Example 3.4 ................................. 52
4.1 The main loop of health-aware decision making ................. 60
4.2 Examples of computational complexity increase for different decision horizons in fully deterministic problems and problems with action outcome uncertainty .............................. 70
5.1 Light-Dark 2D domain .................................................. 94
6.1 SHERPA architecture ................................................... 112
6.2 The VIPER rover ....................................................... 116
6.3 Hermite A crater: the area used for the SHERPA experiments .... 117
6.4 Hermite A crater topography ......................................... 123
6.5 Ice stability depth zones in the Hermite A crater region ....... 123
6.6 Dynamic maps for the Hermite A crater region for 09/23/2021 at 00:00:00 GMT .......................................................... 124

6.7 Stochasticity of the VIPER rover battery SoC across 30 instantiations of the same traverse (recharge durations are selected by LPDM) .... 129
Chapter 1

Introduction

Operating a complex system typically involves two high-level reasoning tasks: decision making, that is, determining which actions to execute in order to achieve the desired objectives, and system health management, focused on ensuring that the system remains in a condition allowing it to successfully execute the actions. Systems of a particular interest to this thesis are those in aerospace applications, where both decision making and system health management can be particularly challenging due to the harshness of operating environments, complexity of the systems, and the uncertainties inherent in their missions. This thesis focuses on the problem of controlling systems in a manner that takes into account both operational and system health management needs. This chapter provides the background on the problem, describes the previous efforts on coordinating the two tasks, and outlines the limitations of the currently prevalent methodology. The chapter then introduces the approach adopted in this work to unifying operational decision making with system health management and the application of this approach to decision support for robotic space missions. Primary contributions of the thesis are listed next, followed by an overview of its structure.
1.1 Scope

The discussion and methods provided in this thesis are expected to be applicable to systems in a variety of fields, such as transportation, manufacturing, and medicine. However, most of the examples used throughout the thesis and the application presented in Chapter 6 come from the field of aerospace. This is because (a) the main ideas of the thesis were developed as a result of prior work in the field and (b) the need for some of the key decision-making attributes integral to the thesis is particularly apparent in aerospace applications. These attributes include:

- ability to generate decisions in real-time;

- ability to make decisions under uncertainty;

- ability to reason about sequences of system actions rather than single actions only; and

- ability to scale to systems of realistic complexity, with large state spaces, a variety of control actions available, and substantial amounts of information supplied by the system’s sensors.

Finally, the overall automated reasoning approach adopted in this work is model-based reasoning (Russell and Norvig, 2009), with a particular focus on using probabilistic state-space models (Koller and Friedman, 2009). In model-based reasoning, a problem is formulated via abstracted representations of the system and its operating environment, i.e., models. Inference is then done by a problem-independent algorithm that uses the domain knowledge obtained from the models in combination with observations from the system’s sensors to estimate system state or provide action recommendations (state-space modeling terminology and definitions used throughout the thesis are detailed in Chapters 2 and 3). Except for supplementing model-based reasoning by rule-based methods in certain tasks (Chapters 2, 5, and 6), use of other automated reasoning approaches — e.g., case-based reasoning or procedural reasoning — is not considered in this work.
1.2 Definitions

The field of system health management (SHM) progressed from simple event-triggered alarms and human-initiated responses to a discipline that often includes sophisticated reasoning algorithms and automated fault recovery recommendations (Aaseng, 2001). The main goal of modern SHM can be formally defined as the preservation of a system’s ability to function as intended (Johnson and Day, 2011; Rasmussen, 2008). While in this thesis the term SHM is applied to the operational phase of a system’s lifespan, the term may also encompass design-time considerations (Johnson and Day, 2011).

The actual achievement of a system’s operational objectives, on the other hand, is under the purview of the fields of control, planning, and scheduling. In this thesis all of the processes aimed at accomplishing operational objectives are generally referred to as decision making (DM), with more specialized terms used where necessary.

The term complex system is used throughout the thesis as a shorthand for systems possessing one or more of the following characteristics: (1) the system can assume a large number of states that may be represented as continuously valued vector quantities; (2) the system’s performance is monitored via a large set of sensor values (observations) that may also be represented as continuously valued vector quantities; and (3) the system is controlled via a large, possibly continuously valued, set of commands (actions). Similarly, the terms complex state space, complex observation space, and complex action space are used as a shorthand for referring to the individual characteristics. State, observation, and action spaces may be collectively referred to as model spaces. The context for the usage of large in reference to model spaces is provided in Chapter 5.

1.3 Background

Historically, the field of DM has developed separately from SHM. Typical SHM functions (monitoring, fault detection, diagnosis, mitigation, and recovery) were originally handled by human operators, similar to DM (Aaseng, 2001; Ogata, 2010).
As simple automated control was being introduced for DM, automated fault monitors and alarms reduced operator workload for SHM. Gradually, more sophisticated control techniques were developed for DM (Ogata, 2010), while automated emergency responses started handling some off-nominal system health conditions (Rasmussen, 2008). Capable computerized planning and scheduling tools eventually became available for performing higher-level, more strategic decision making (Ghallab, Nau, and Traverso, 2016). On the SHM side, advanced methods were developed to improve robustness of anomalous event detection and alerting (Chryssanthacopoulos and Kochenderfer, 2012; Iverson, 2004). Computerized fault diagnosis was also added, eventually growing into a subfield of artificial intelligence in its own right (Feldman, Provan, and Gemund, 2010; Korbicz, Koscielny, Kowalczyk, and Cholewa, 2012; Metodi, Stern, Kalech, and Codish, 2014). In some cases diagnosis was coupled with failure prediction (prognostic) algorithms (Lu and Saeks, 1979), as well as with recovery procedures (Avizienis, 1976). Still, the two sides, DM and SHM, remained largely separated.

When the concept of Integrated System Health Management (ISHM) became formalized, most interpretations of integrated encompassed some interactions between DM and SHM, although practical implementations of such interactions have been limited (Figueroa and Walker, 2018). A notable early example of ISHM is the Deep Space 1 Remote Agent Experiment, or RAX (Bernard et al., 1999). RAX integrated an automated planner/scheduler, a command execution system, and a mode identification and reconfiguration module. The latter performed state estimation and fault detection/diagnosis. If a fault was detected and identified, a single recovery action was recommended to the execution module. The mode identification and reconfiguration module was based on Livingstone (Williams and Pandurang, 1996), a reactive configuration management engine that reasons over declarative models. Livingstone later evolved into Livingstone 2 (Kurien and Nayak, 2000), which served as the diagnostic component of several other ISHM prototypes (Meyer et al., 2003; Nicewarner and Dorais, 2006; Schwabacher, Samuels, and Brownston, 2002). In Livingstone 2 problems were represented as partially observable Markov decision processes (Kaelbling, Littman, and Cassandra, 1998), although that representation was only used
for tracking the state of a system, rather than for generating action recommendations. Benedettini, Baines, Lightfoot, and Greenough (2009) and Prajapati, Roy, and Prasad (2018) list additional ISHM implementation examples, while also noting that they have been relatively rare.

1.4 Motivation

The need for further automation of DM and SHM in aerospace applications, as well as for effective coordination between them is only going to become more prominent, as the scale of aerospace operations continues to increase, coupled with the drive to decrease their costs. On the aviation side, in addition to the continuing growth of traditional airline operations, the emerging sectors of Urban Air Mobility and Unmanned Aircraft Systems may increase the current number of daily flight operations by at least an order of magnitude within the next two decades (Federal Aviation Administration, 2019). If these projections hold true, present-day approaches to flight operations, air traffic management, pre-flight inspections, and aircraft maintenance that rely heavily on highly trained personnel will not be sustainable (Vascik, 2019).

On the space side, efforts are underway to develop airline-like access to low Earth orbit: safe, frequent, and affordable (Pelton, 2019). In order to achieve that goal, however, major advances in reusability, reliability, and operational efficiency of orbital vehicles would need to be achieved. Robust, automated DM and SHM capabilities could play pivotal roles in all three of these categories (Fox and Glass, 2000; Madry, 2020; Troya, Mailloux, Miller, Fitzharris, and Mueller, 2018).

Beyond low Earth orbit, first steps towards a cis-lunar and lunar economy are being taken. National Aeronautics and Space Administration (NASA) has been given a mandate for “extending human exploration missions into cis-lunar space” and re-establishing a presence on the Moon (United States Congress, 2017). A number of other national space agencies, private companies, and academic institutions are also working on creating the foundational infrastructure of a space-based economy, including manufacturing equipment (Patané, Joyce, Snyder, and Shestople, 2017; Prater et al., 2019), in-situ resource utilization facilities (Lehner et al., 2019; Sivolella, 2019),
large crewed habitats (Borowski, Ryan, McCurdy, and Sauls, 2019), and other complex systems (Barr, 2015; Kinsman et al., 2019; Zuniga, Modi, Kaluthantrige, and Vertadier, 2019). Even routine operations within this new ecosystem may require sophisticated, strategic decision making; when malfunctions do occur, a timely and effective response may be crucial to preventing loss of life or loss of essential functionality. When an onboard explosion endangered the Apollo 13 crew on the way to the Moon in 1970, multidisciplinary teams of experts worked around the clock to exhaustively analyze potential scenarios and improvise a solution (NASA, 1970). With some studies projecting at least a thousand people living and working in space at any given time by the 2040s (Kutter, 2016), the same approach to handling off-nominal and emergency situations is unlikely to scale sufficiently well.

Finally, as discussed in more detail in Chapter 6, the traditional approach to operating robotic exploration missions—built on the foundation of control sequences designed, verified, and uploaded by human experts on Earth—is also reaching its limits as increasingly ambitious robotic missions to deep space destinations are being planned. NASA’s Europa Lander mission (NASA, 2016), currently in early development stages, can serve as a representative example. Operating in the harsh surface environment of Jupiter’s fourth largest moon, with high levels of radiation and low ambient temperatures, the risk of malfunctions for the lander is expected to be significant. Long communication delays (as long as 106 minutes round-trip time) will make timely interventions by ground controllers in case of problems impractical. Given the operating conditions, the lander is only anticipated to survive for roughly twenty Earth days, meaning that any unplanned delays could have a substantial impact on the overall scientific return. For these reasons, the current concept of operations envisions the lander functioning fully autonomously for long stretches of the mission, including identifying root causes of potential malfunctions and executing recovery procedures.
1.5 Challenges

A number of fundamental challenges exist for implementing automated DM and SHM for realistic complex systems. The curse of dimensionality—i.e., the exponential growth in complexity of many computational tasks with the number of input data dimensions—often becomes a concern (Bellman, 1961). When making decisions under uncertainty, the lesser known curse of history (Pineau, Gordon, and Thrun, 2006), where the number of contingent plans increases exponentially with the planning horizon, may also create significant difficulties. Both of these issues are discussed in Chapters 2, 4, and 5 in the context of automated decision making methods.

Successfully diagnosing system health issues typically requires detailed models and sufficient sensor coverage (Henry, Simani, and Patton, 2010), which may not be available for every off-nominal condition the system may encounter. Just narrowing down the set of possible root causes of a malfunction on the basis of noisy, often ambiguous sensor inputs may become a hard computational task (Verma, 2004). Lastly, accurately capturing even nominal system behavior in a model can be challenging, particularly if stochastic traits are involved (Maybeck, 1982). If reasoning about degradation of a system component is desired, developing models for the task comes with its own set of difficulties (discussed in Chapter 3).

1.6 Limitations of Current Methods

This work originally started with the intent of improving the prevalent ISHM approach, rather than abandoning it. In particular, the main objective was to identify or develop methods to translate prognostic output into better system health management decisions. Over the course of the work it became evident, however, that the traditional approach of employing separate—even if connected—subsystems for decision making and system health management has fundamental limitations (discussed in detail in Chapter 3). Among them is the possibility that divergent objectives of DM and SHM will result in undesirable or poorly timed system actions. Another limitation is that the two subsystems, managing their own sources of information
about the state of the system and having separate sets of control actions to chose from, may not come up with the best decisions to achieve their objectives, even if incompatibility of actions is avoided.

Finally, it was determined that the process of prognostics adds no value in making decisions for systems where control actions influence system degradation processes (which, for example, is the case for many aerospace systems of interest). For certain categories of systems where health degradation is decoupled from control actions or where control over system behavior is limited in general, prognostic analysis may theoretically be applied, although in practice could be challenging to implement effectively.

1.7 Approach

What instead appeared to be a more promising direction was to incorporate elements of health management into the overall system decision making process. The basic idea is not new; certain system health elements have been included into decision making formulations before, both in own work and in work by others (Section 4.1). This thesis takes it a step further, however, demonstrating not only that the two tasks can be performed within the same reasoning framework, but also why they should be. A general, comprehensive approach to unification of operational decision making and system health management is then proposed.

The unified approach, Health-Aware Decision Making (HADM), relies on two key concepts: (1) state-space system modeling and (2) formulation of a utility/value function over the state space (Chapter 4). As part of the unification process, state, observation, and action spaces are combined. The proposed approach is suitable for systems of various levels of complexity, systems where deterministic assumptions can be made and where uncertainties need to be taken into account (Balaban, Johnson, and Kochenderfer, 2018, 2019). Even though the approach is most beneficial in cases where optimality of operations is desired, it also has advantages in cases where optimality is not expected — as long as the system is operating within certain constraints.
While it is proposed for DM and SHM to be unified in their formulations, the rationale for implementing system emergency response as a standalone function is also presented.

The proposed unified approach, while improving the effectiveness of action selection, may, however, result in computational complexity growth in certain situations — relative to separately implemented DM and SHM (Chapter 4). This may be the case when the size, number of dimensions, or both increase for any of the model spaces (state, action, or observation) as a result of unification. While extensive research efforts have led to the development of a number of decision making methods for complex state spaces, the same cannot yet be said for complex action or observation spaces. Being able to efficiently handle all three types of complex spaces may be needed, however, for solving realistic HADM problems, where a continuous (or large discrete) representation of state and observation spaces may be required for timely detection and identification of incipient malfunctions. A combined action space, instrumental to producing decisions that balance DM and SHM needs, may also create a significant computational burden for existing decision making algorithms.

With that need in mind, the Large Problem Decision Making (LPDM) solver — a new algorithm for automated decision making under uncertainty — was created. LPDM was built on the foundation of an existing, state-of-the-art partially observable Markov decision process (POMDP) solver already capable of handling complex state spaces (POMDPs are introduced in Chapter 2). The approach to handling complex action spaces is based on the recognition that methods for global black-box optimization may be applicable to the problem of action selection from a large action domain (Chapter 5). One of the global optimization methods, Fast Simulated Annealing (FSA), was chosen as the template for providing complex action spaces support in LPDM.

None of the existing methods for handling complex observation spaces appeared to be suitable for HADM (please see the discussion in Chapter 5), therefore a novel method was developed and implemented in LPDM. The method is based on correlation clustering of system state transitions and, among its features, allows for the possibility that seemingly unrelated observations may be providing clues about the
same underlying system state (a common occurrence in realistic systems that can be used in disambiguation of a fault diagnosis).

While the development of LPDM was motivated by HADM applications, it is a general-purpose POMDP solver and is expected to be useful for a variety of decision making problems involving complex systems operating under uncertainty. LPDM was benchmarked against DESPOT on different variants of Light-Dark Domain, a commonly used abstract POMDP problem featuring continuously valued state, action and observation spaces. In the benchmarking experiments, the advantages of LPDM’s methods for handling complex action and observation spaces became particularly evident on the more challenging, two-dimensional variants of Light-Dark.

LPDM serves as a foundational piece of System Health Enabled Realtime Planning Adviser (SHERPA), a decision support system for robotic exploration missions built on HADM principles. SHERPA is designed to be configurable for different types of missions and for different use cases within a mission (e.g., assisting with selecting the landing site for a robotic rover, then also helping to plan its traverses). SHERPA is built using Julia, a modern high-performance programming language for scientific computing (Bezanson, Edelman, Karpinski, and Shah, 2017) and is intended to be compatible with the POMDPs.jl framework (Egorov, Sunberg, et al., 2017). This allows for the possibility of using alternative POMDP solvers and system state estimators, should they prove to be more appropriate for a particular application. SHERPA and its use cases for the upcoming Volatiles Investigating Polar Exploration Rover (VIPER) mission are discussed in detail in Chapter 6.

1.8 Contributions

This section briefly overviews the primary contributions of this thesis by grouping them in three categories: Theory, Algorithms, and Application. The contributions of this work are also highlighted within the more detailed descriptions in the subsequent chapters and then summarized in Chapter 7.
1.8.1 Theory

The first contribution of this work is re-examining the currently prevalent approach to system health management—where SHM is implemented as a separate function from the system’s operational DM—and identifying its limitations. In particular, two key limitations are identified: (1) that separated DM/SHM formulations can encounter performance issues due to conflicting objectives and (2) that separated formulations can be ineffective due to mutually inaccessible model spaces. Additionally, the thesis shows that prognostics (a common component in modern SHM architectures) is not meaningful in most cases and is challenging to implement effectively in the rest.

The second contribution of the thesis is proposing a unified approach for performing DM and SHM, Health-Aware Decision Making (HADM), that overcomes the limitations of conventional, separated DM/SHM formulations. While the HADM approach is expected to result in more effective and efficient operations, in some applications it may result in additional computational costs. The computational complexity of HADM is analyzed and recommendations are made for overcoming complexity increases.

1.8.2 Algorithms

The most complex HADM problems are those where uncertainty is present in action outcomes and system state estimation (partial observability). This type of problems becomes even more challenging to solve when high-dimensional, continuously valued (or simply large) state, action, and observation spaces are involved—which is often the case in realistic aerospace applications. While there exists a substantial body of work aimed at dealing with high-dimensional state spaces, progress on handling action and observation spaces with the same properties has been more limited.

The third contribution of the thesis is developing methods for reasoning under state and outcome uncertainty for problems where high-dimensional, continuous (or large discrete) action and observation spaces are necessary. Specifically, it is done for problems formulated as partially observable Markov decision processes. The method
for handling complex action spaces takes inspiration from *Simulated Annealing*, a
global, black-box optimization algorithm. The method for handling complex obser-
vation spaces is novel and, as best as can be determined, without analogs in relevant
literature. It is based on correlation clustering of state transitions.

The two methods are combined with DESPOT, a state-of-the-art POMDP solver
capable of handling complex state spaces (overviewed in Chapter 2) to result in the
Large Problem Decision Making (LPDM) solver. Performance of the new solver is
evaluated on a set of benchmarking problems.

1.8.3 Application

The overall HADM approach and the new LPDM solver form the foundation for the
fourth contribution: a new decision support system for robotic planetary missions,
System Health Enabled Realtime Planning Advisor (SHERPA). SHERPA is likely the
first decision support system capable of automated reasoning under uncertainty to
be used on a space mission — the upcoming Volatiles Investigating Polar Exploration
Rover (VIPER) mission being developed at NASA Ames Research Center. VIPER,
currently scheduled to launch in November of 2023, will investigate South Pole regions
of the Moon, identifying and characterizing deposits of water ice and other volatiles.
It is envisioned that SHERPA will support a number of VIPER use cases based on
intelligent execution of mission scenarios under uncertainty, spanning all phases of
the mission.

A VIPER use case that has already been developed and tested, Traverse Evalua-
tion and Refinement, implements mixed-initiative traverse planning. In it SHERPA
takes an ordered set of waypoints and activities selected by human experts, evaluates
the robustness of the resulting family of traverses to possible delays and malfunctions,
and computes durations of recharge periods throughout the route that maximize the
likelihood of traverse completion.

In another use case, Traverse Synthesis, SHERPA only takes the high-level mis-
sion objectives and constraints as inputs. It then automatically generates multi-day
traverses for the rover that are, again, designed to be robust to delays and malfunctions. Traverse Synthesis is currently nearing completion and is overviewed as part of ongoing work in Chapter 7.

1.9 Organization

This chapter introduced the general areas of operational decision making and system health management and provided some historical perspective on the trajectories of their development. The motivation for the work in this thesis and its main contributions were outlined as well. The rest of the thesis is structured as follows.

Chapter 2 provides a brief review of the relevant automated decision making concepts, particularly those dealing with decision making under uncertainty. Where relevant, connections to system health management are highlighted and discussed. Key algorithms, referenced through the remainder of the thesis, are also introduced.

Chapter 3 describes the prevalent approach of implementing system health management as an independent component, identifies its main downsides, and illustrates these downsides with numerical examples. In addition to conveying the issues with the overall approach, the chapter includes a detailed discussion of the problems with relying on prognostics in making SHM decisions. The chapter also contains a section on prior work conducted in the SHM area that has led to some of the main ideas and conclusions described in the thesis.

Chapter 4 consists of three major parts. First, it describes the approach proposed in this thesis that overcomes the limitations of contemporary SHM — Health-Aware Decision Making. That is followed by a discussion on why it would be advantageous to implement system emergency response independently from HADM. The chapter then presents a computational complexity analysis of HADM (based, in part, on the algorithms described in Chapter 2). The analysis is organized into three parts of its own, first discussing problems with deterministic states and action outcomes, followed by problems with uncertainty present in action outcomes, and, finally, problems with uncertainty present in both action outcomes and state estimation.
Chapter 5 discusses the challenges automated decision-making algorithms face when reasoning over complex action and observation spaces. The chapter then presents two algorithms, Adaptive Action Selection and Transition Correlation Clustering, designed to overcome some of these challenges. The two algorithms are combined with DESPOT to form a new POMDP solver, LPDM. In the final part of the chapter, performance of LPDM is evaluated relative to DESPOT on several variants of an abstract POMDP with complex model spaces.

Chapter 6 focuses on SHERPA, the decision support system for robotic space missions based on LPDM and the overall principles of HADM. Background information on the VIPER mission to the South Pole of the Moon — where SHERPA is currently being used for traverse planning and evaluation — is also provided. The chapter then covers the experiments conducted to validate SHERPA’s performance, including those with simulated system health problems.

Chapter 7 first summarizes the thesis and reviews its key contributions. It then outlines ongoing, as well as future work on LPDM and SHERPA and concludes the thesis with brief final remarks.
Chapter 2

Decision Making under Uncertainty

“In preparing for battle I have always found that plans are useless, but planning is indispensable.” This quote, attributed to Dwight D. Eisenhower, may be just as applicable to operating complex systems (such as aircraft or spacecraft) in challenging environments as it is to wartime battles. Rarely does a space mission proceed entirely according to its pre-launch plan, with at least some replanning or adjustments typically required. Even in commercial aviation, with its strong emphasis on consistency in execution, events that may substantially affect the initial flight plan occur daily for a significant number of flights (e.g., adverse weather, equipment malfunctions, or air traffic management issues). The field of decision making under uncertainty (DMU) has been developed to formalize and optimize action selection when a deterministic unfolding of events cannot be guaranteed. As DMU is a central theme of the thesis, this chapter overviews the fundamental concepts and introduces some of the relevant algorithms. Connections to the other central theme, system health management, are highlighted as well.
2.1 Introduction

In September 2005, the Japan Aerospace Exploration Agency (JAXA) Hayabusa spacecraft — launched in May 2003 — arrived in the vicinity of asteroid 25143 Itokawa to study it and collect samples for return to Earth (Yoshikawa, Kawaguchi, Fujiwara, and Tsuchiyama, 2015). Hayabusa was, however, originally supposed to launch in July 2002 to a different asteroid, 4660 Nereus, with asteroid 1989 ML as a backup target. Problems with the launch vehicle and the spacecraft itself delayed the launch by nearly a year, putting both original targets out of reach.

While en route to Itokawa, the largest solar flare in recorded history damaged the spacecraft’s solar arrays. Resulting reduction in electrical power diminished the efficiency of the ion engines, delaying the arrival to Itokawa by several months. This significantly limited the amount of time Hayabusa was able to spend at the asteroid and decreased the planned number of sampling sites from three to two. Additionally, two attitude control reaction wheels failed, necessitating use of maneuvering thrusters for attitude changes and requiring further changes to the mission plan. Only one of the two sampling sites ended up being visited, as the other was found to be too rocky for a safe touchdown. During one of the two landing attempts at the sole sampling site, release of the mini-probe MINERVA failed and the exact position and status of Hayabusa could not be ascertained due to communication dropouts.

Despite these and numerous other problems and malfunctions, nearly five years later, the spacecraft was able to complete its journey back to Earth. On June 14, 2010, its reentry capsule made a soft landing, delivering valuable samples from Itokawa for scientific analysis.

The case of the Hayabusa mission, just as the earlier case of the Apollo 13 mission, is a vivid illustration of successful decision making under uncertainty. It contains several examples of all three uncertainty types that are discussed throughout this thesis: state uncertainty, action outcome uncertainty, and model uncertainty. For the Hayabusa spacecraft, decision making was accomplished effectively through the ingenuity of its ground support teams, who managed to restore and maintain communications with the spacecraft during critical phases of the mission and came up with
timely solutions to its numerous issues. When, however, communication links fail or when a communication signal takes too long to travel to and from the spacecraft, having the ability to perform autonomous decision making under uncertainty may mean the difference between mission success (even if partial) and its failure.

The rest of this section discusses the general approach to DMU adopted in this thesis, defines key concepts (highlighted in bold), introduces notation, and overviews several relevant algorithms. Notation generally follows standard DMU conventions (Kaelbling, Littman, and Cassandra, 1998; Kochenderfer, 2015). The discussion begins with the most fundamental concept— that of system states.

2.1.1 States

A system and its operating environment (the world) may assume a number of distinct configurations (for realistic systems this number is typically infinite). A state $s$ is a descriptor of such a configuration. Typically a state contains only just enough information to fully capture a unique configuration. As a convenient shorthand, whenever a reference to a system state is made in the rest of the thesis, it is implied that the state describes the relevant information about the system, as well as about its operating environment (unless a distinction between the two needs to be made for clarity).

In this thesis, states are assumed to have the Markov property (Kemeny and Snell, 1983), i.e., that a state $s$ of a system is defined in such a way that the next state $s'$ depends only on $s$ and not on any previous states. In this work, a state can be a scalar or a vector quantity, discrete or continuous. All possible states of a given system form its state space $S$.

2.1.2 Actions

In a controlled system, an action $a$ initiates transition from the current state $s$ to the next state $s'$ (in an uncontrolled system, transition from $s$ to $s'$ does not require an initiating action). All available control actions for a system form its action space $A$ ($A$ may be state-dependent).
An action can be represented by a scalar or vector. The latter may be used when, for instance, multiple components of a system are controlled simultaneously. Like states, actions may be discrete or continuous.

2.1.3 Observations

It is not always possible to determine the state of a system and its operating environment exactly. An observation $o$ is information about the state, even if indirect, that is possible to obtain. Observations can be obtained through system sensors, for example, and in many cases may be incomplete, noisy, or even contradictory.

Similarly to states and actions, observations in this work may be represented by a scalar or a vector (discrete or continuous). Vectors may be used when, e.g., readings come from multiple sensors. All observations possible for a system form its observation space $O$.

2.1.4 History

For an uncontrolled system, history $h$ is a sequence of observations collected over time. For a discrete sequence of time steps (up to the current time $t$) at which observations are collected, the history is then $h_t = \{o_1, o_2, \ldots, o_t\}$. For a controlled system, first a decision is made about the action to take in the current state, then the action is executed, the resulting observations are collected, and the cycle starts anew. For the controlled case, history is a sequence of actions and observations, e.g., $h_t = \{a_1, o_1, a_2, o_2, \ldots, a_t, o_t\}$.

In SHERPA (Chapter 6), the decide-act-observe cycle is decoupled from discrete time indices; in fact, time becomes a component of the state vector. Instead, the notion of decision making periods (or, equivalently, decision making steps) is used, where each period can last an arbitrary amount of real-world time. A history sequence would then be indexed as $h_t = \{a_1, o_1, a_2, o_2, \ldots, a_n, o_n\}$, where $n$ is the current decision making period.
CHAPTER 2. DECISION MAKING UNDER UNCERTAINTY

2.1.5 Models

A model is an abstracted representation of a system and its operating environment (or a representation of a particular aspect of them). Models may be represented as functions that take variable inputs and generate an output. A model may be deterministic or stochastic; in the former case the model always produces the same output for the same input, in the latter case the output is stochastic. All models mentioned in the thesis should be considered stochastic, unless noted otherwise, with the fundamental types described below.

State transition models

For uncontrolled systems, a state transition model $T(s' | s)$ describes the probability of transitioning to a particular state $s'$ from state $s$. A state transition model for controlled systems takes the form $T(s' | s, a)$, describing the probability of transitioning to a particular state $s'$ as a result of taking action $a$ from state $s$.

Observation models

For uncontrolled systems, an observation model $Z(s' | s, o)$ describes the probability of getting an observation $o$ upon transition to state $s'$ from state $s$. For controlled systems, an observation model $Z(o | s, a, s')$ does the same, but for a transition resulting from action $a$. Note that the most general definition is used here; in some cases an observation has a dependency on $a$ and $s'$ only or even just on $s'$.

Reward models

For uncontrolled systems, a reward model $R(s)$ describes the reward $r$ obtained as a result of transition from state $s$ to state $s'$. Similarly, for controlled systems $R(s, a)$ describes a reward obtained as a result of reaching state $s'$ by taking action $a$ in state $s$. As for $Z$, these are the most general definitions. In many cases $R$ is dependent on the current state only, for example. $R$ is the only fundamental model type used in the thesis that is deterministic, as stochastic reward models have been shown to provide no benefits for the decision making methods chosen.
2.1.6 Types of Uncertainty

For the purposes of this thesis, all relevant sources of uncertainty are classified into three categories: outcome uncertainty, state uncertainty, and model uncertainty. Their definitions and illustrative examples are provided below.

**Outcome Uncertainty**

If executing action $a$ in state $s$ for some system is not guaranteed to transition the system to a unique next state $s'$, the system is defined to have **outcome uncertainty** (or, equivalently, **outcome uncertainty** for brevity). If modeling outcome uncertainty is important, a stochastic state transition model may be used. It is assumed that outcome uncertainty cannot be reduced by, for instance, taking information-gathering actions and analyzing resulting observations.

An example of outcome uncertainty from the Hayabusa mission is the failed deployment of MINERVA probe. A command to release the probe was issued with the expectation that the next system state will describe MINERVA separated from Hayabusa. Instead, the system transitioned to a state where the probe was still attached to Hayabusa.

**State Uncertainty**

When the current state $s$ of a system cannot be determined exactly, the system is defined to have **state uncertainty**. The uncertainty about the Hayabusa spacecraft position and configuration due to communication issues during one of the landing attempts on Itokawa is a good example of state uncertainty.

If observations are available that contain at least some information about the true system state (and thus may help reduce state uncertainty), the states are defined to have **partial observability**. Systems with partial state observability are the main focus of this thesis. In modeling systems with partial state observability for decision-making purposes, belief states may be used. A **belief state** $b$ (or **belief**, for brevity) captures the information available about a partially observable true system state. A belief $b$ may be modeled as a probability distribution over $S$, as is done in this thesis.
$B$ denotes the space of all beliefs.

**Model Uncertainty**

For some systems a model needed for decision making may not be available or may be under-specified, with some of its parameters unknown or known only approximately. This is referred to as **model uncertainty**. Model uncertainty can exacerbate outcome uncertainty; there is, however, an important distinction between the two. Unlike outcome uncertainty, model uncertainty may be reduced through taking actions (including those selected specifically to gather information) and analyzing the resulting observations. While methods for automated model learning are not dealt with extensively in this thesis, model uncertainty is discussed at several points. For instance, in Section 3.4 (Prognostics), several challenges in developing degradation models can be classified as model uncertainty problems.

Where appropriate, model uncertainty may be represented as state uncertainty if unknown parameters are included in the state description. An example of that in the thesis is the VIPER fault scenario described in Section 6.7.1, where friction coefficients of rover’s motors are considered unknown. They are included in the state vector and are estimated as part of the belief update process (which includes observation analysis). This methodology is generally applicable to the SHM problem of **fault identification**, where, in addition to determining the **fault mode**, the magnitude of the fault needs to be estimated.\(^1\)

In the Hayabusa case study, the issue of model uncertainty can be illustrated on the following example. The mission plan, revised after the delayed arrival of the spacecraft to the vicinity of asteroid Itokawa, included sampling from two sites—without realizing that the second planned site was too rocky for a safe touchdown. It became evident that the site is unsuitable only when close-up photos of it were taken by the spacecraft. Had a formal model of the asteroid’s surface been used, it could have then been updated with this new information.

\(^1\text{Fault identification is also often considered to be part of fault diagnosis, rather than a separate task.}\)
2.1.7 Utility

Along with state-space models, utility theory (Fishburn, 1970) is fundamental to the work described in this thesis—particularly to the proposed unified DM/SHM approach. Utility is a numerical measure of preference over the space of possible outcomes. A related concept is that of a utility function $U$ (or, equivalently, value function $V$), that defines such a numerical measure for a particular set of input variables. For instance, if a utility function is defined for lunar rover states, a higher utility value may be assigned to a state where a desired scientific location has been reached and a lower value to a state where a malfunction has occurred.

Utility functions commonly used in the rest of the thesis are defined for the following inputs: states, beliefs, state-action pairs, and belief-action pairs. State-action and belief-action utility functions are also often denoted as $Q(s,a)$ and $Q(b,a)$, respectively. Optimal versions of the functions are denoted as $V^*$ (value), $U^*$ (utility), and $Q^*$ (state/belief-action utility).

2.1.8 Sequential Decision Making

The general function of decision making is to select actions. While in some systems the focus of decision making is only on choosing a single, immediate action at a given time $t$, real world decision making problems often involve considering a sequence of actions. The length of the sequence, i.e., how many time intervals or decision making steps are being looked ahead, is defined as the decision horizon $H$ (or, equivalently, planning horizon). In a strictly deterministic system operating in a deterministic environment, an entire sequence of actions—a plan—can be selected ahead of time. In systems with action outcome uncertainty, however, a fixed plan can quickly become obsolete. Instead, a policy $\pi(s) : S \rightarrow A$ needs to be selected that prescribes which action is to be taken in any state. In a partially observable setting, a policy maps beliefs to actions, i.e., $\pi(b) : B \rightarrow A$. Note that only deterministic policies, i.e., those where a state/belief is only mapped to one action, are considered in this thesis.

A policy can be either offline (precomputed for all states or beliefs of interest) or online (computed for the current state or belief only). An optimal policy $\pi^*$ is a
policy that, in expectation, optimizes a desired metric.

In this thesis the metric optimized is state (or belief state) utility. Problems with outcome uncertainty are modeled as Markov decision processes and problems with both outcome and state uncertainty are modeled as partially observable Markov decision processes. These modeling frameworks are introduced in the next two sections (Section 2.2 and Section 2.3, respectively). These sections will also introduce some of the standard policy derivation algorithms for the two frameworks. Sections 2.6–2.8 go into detail on some of the state-of-the-art Markov decision process (MDP) and POMDP algorithms directly relevant to the material in Chapters 5 and 6.

### 2.2 Markov Decision Processes

Markov decision process (MDP) is a general mathematical framework for optimal sequential decision making under outcome uncertainty (Bellman, 1957). An MDP is formally defined by a tuple \((S, A, T, R, \gamma)\), where \(S\) is the state space, \(A\) is the action space, \(T : S \times A \times S \to \mathbb{R}\) is the state transition function, \(R : S \times A \times S \to \mathbb{R}\) is the reward function, and \(\gamma \in (0, 1]\) is the discount factor that is sometimes used to bias towards earlier rewards, particularly in infinite horizon problems \((H \to \infty)\). A special subset \(S_T \subset S\) of **terminal (goal) states** is often defined—although not strictly necessary for solving an MDP. Transitions from a terminal state are only allowed back to itself.

The objective in solving an MDP is to find an optimal policy \(\pi^*\) that, for \(\forall s \in S\), in expectation maximizes the utility function \(U(s)\) (often called the value function \(V(s)\) in MDP literature). The expected utility associated with executing a policy \(\pi\) for \(n\) steps from state \(s\) is defined in terms of the reward function and can be computed recursively as

\[
U^\pi_n(s) = R(s, \pi(s), s') + \gamma \sum_{s'} T(s, \pi(s), s') U^\pi_{n-1}(s'),
\]

(2.1)
An estimate of optimal utility for a state can then also be computed recursively using
\[
\hat{U}_n(s) = \max_{a \in A} \left( R(s, a) + \gamma \sum_{s'} T(s, a, s') \hat{U}_{n-1}(s') \right),
\] (2.2)

which for \( n \to \infty \) becomes the Bellman equation (Bellman, 1957) computing the optimal utility:
\[
U^*(s) = \max_{a \in A} \left( R(s, a) + \gamma \sum_{s'} T(s, a, s') U^*(s') \right).
\] (2.3)

Knowing the optimal utility function, an optimal policy can be derived:
\[
\pi^*(s) = \arg \max_{a \in A} \left( R(s, a) + \gamma \sum_{s'} T(s, a, s') U^*(s') \right).
\] (2.4)

Equations 2.1–2.4 form the basis for two standard, exact algorithms for generating optimal MDP policies: value iteration (Bellman, 1957) and policy iteration (Howard, 1960).

As the name implies, value iteration works by gradually improving estimates of value (utility) for all states of an MDP using Equation 2.2. Iterations are terminated when \( \hat{U}_k(s) \) stop changing or, as more commonly done in practice, when changes in \( \hat{U}_k(s) \) estimates become small enough (\( k \) is the current iteration). The Bellman residual is one example of a metric used to terminate value iteration. It is defined as the max norm \( \| U_k - U_{k-1} \| \), where \( \| U \| = \max_s |U(s)| \) (Kochenderfer, 2015). Once the algorithm converges, an optimal policy is obtained using Equation 2.4.

Policy iteration uses the fact that with a finite \( |S| \) and \( |A| \), there is a finite space of possible policies \( \Pi \). It searches for an optimal policy \( \pi^* \in \Pi \) by starting with an arbitrary policy \( \pi_0 \) and repeating the following two steps:

1. **Policy evaluation:** using Equation 2.1 and the current policy \( \pi_k \), compute \( U^{\pi_k} \).

2. **Policy improvement:** for all \( s \in S \), compute \( \pi_{k+1} \) using \( U^{\pi_k} \) and the following
equation (similar to Equation 2.4):

\[
\pi_{k+1}(s) = \arg\max_{a \in A} \left( R(s, a) + \gamma \sum_{s'} T(s, a, s') U_{\pi_k}(s') \right).
\] (2.5)

The algorithm terminates when \( \pi_{k+1} = \pi_k \).

Value iteration has a computational cost \( O(|A||S|^2) \) per iteration; however, the number of iterations may grow exponentially in the MDP discount factor (Condon, 1992; Kaelbling, Littman, and Moore, 1996). Policy iteration has a significantly higher per-iteration cost, \( O(|A||S|^2 + |S|^3) \), although in practice may converge in fewer iterations than value iteration (Kaelbling, Littman, and Moore, 1996; Littman, Dean, and Kaelbling, 1995).

Unfortunately, value iteration and policy iteration are not suitable for problems with large or continuously valued \( S \) and \( A \). Instead, algorithms for computing approximately optimal policies have been developed. Notable examples of such algorithms include sparse sampling (Kearns, Mansour, and Ng, 2002), LAO* (Hansen and Zilberstein, 2001), and Monte Carlo Tree Search (Browne and Powley, 2012). The latter has become widely used in recent years for MDP applications due to its scalability to large state spaces. It has also served as the foundation for a number of other MDP and POMDP solvers, including POMCP (Section 2.7) and DESPOT (Section 2.8). Monte Carlo Tree Search (MCTS) is described in Section 2.6.

### 2.3 Partially Observable Markov Decision Processes

Partially observable Markov decision process (POMDP) is a generalization of MDP for acting under both outcome and state uncertainty (Kaelbling, Littman, and Cassandra, 1998). While alternative mathematical frameworks for the same purpose exist — such as Predictive State Representations (Littman, Sutton, and Singh, 2002; Singh, James, and Rudary, 2004) — in this thesis partially observable problems are formulated as POMDPs only.
A POMDP is defined by a tuple \( \langle S, A, T, R, O, Z, \gamma \rangle \), i.e., compared to an MDP, an observation space \( O \) is added, along with an observation model \( Z : S \times A \times S \to \mathbb{R} \). The objective in solving a POMDP is similar to that of an MDP: find an optimal policy \( \pi^* \) that, for \( \forall b \in B \), in expectation maximizes the utility function \( U(b) \) (also known as the value function \( V(b) \)).

Once a POMDP policy \( \pi^* \) is found, the decide-act-observe cycle unfolds as follows: the system in a belief state \( b \) obtains the recommended action \( a = \pi^*(b) \), executes it, obtains observation \( o \), then uses \( o \) to perform a belief update to determine the next belief \( b' \). The system itself has, in fact, transitioned to some latent state \( s' \) as a result of executing action \( a \), but since it cannot be observed directly, \( b' \) needs to be computed on the basis of \( b, a, \) and \( o \). Performing a belief update may be thought of as an equivalent to executing state transition in a fully observable setting using a state transition model \( T(s' | s, a) \). Methods for belief updating are discussed in Section 2.9.

For small discrete POMDPs, exact optimal policies may be computed (Kochenderfer, 2015). For instance, since a POMDP is equivalent to an MDP with its state space defined as the space of all beliefs \( B \) (Kaelbling, Littman, and Cassandra, 1998), value or policy iteration can be used to solve it. Beliefs would simply take the place of states in equations 2.1–2.5. Generally, however, computing optimal finite-horizon POMDP policies exactly is considered a PSPACE-complete problem (Papadimitriou and Tsitsiklis, 1987), making it unlikely that a time- and memory-efficient algorithm to do so can be found. Therefore, in most cases, approximately optimal policies are derived using either offline or online algorithms (Kurniawati, Hsu, and Lee, 2008; Pineau, Gordon, and Thrun, 2006; Ross, Chaib-draa, and Pineau, 2008).

Two online POMDP solvers directly relevant to this thesis (POMCP and DESPOT) are described in Sections 2.7 and 2.8, respectively. Monte Carlo Tree Search, an online MDP solver which served as the template for both POMCP and DESPOT, is covered in Section 2.6.
2.4 Mixed Observability Markov Decision Processes

Mixed observability Markov decision processes (MOMDPs) are a special class of POMDPs that plays a role in SHERPA, the decision support framework described in Chapter 6. In a MOMDP, although the system state is not fully observable, some of its components are (Ong, Png, Hsu, and Lee, 2010). A MOMDP state vector can be represented as a factored vector \((s_{fo}, s_{po})\), with \(s_{fo}\) representing the fully observable state components and \(s_{po}\) representing the partially observable components. The joint factored MOMDP state space is \(S = S_{fo} \times S_{po}\), where \(S_{fo}\) is the space of all values for \(s_{fo}\) and \(S_{po}\) is the space of all values for \(s_{po}\). The state transition function is also factored, with \(T_{fo}\) and \(T_{po}\) defined for the fully and partially observable state vector components, respectively. A MOMDP is formally specified as a tuple \(\langle S_{fo}, S_{po}, A, T_{fo}, T_{po}, R, O, Z, \gamma \rangle\), which only differs from the POMDP tuple in its representations of the state space and the state transition function. General POMDP solution methods have been shown to be adaptable to MOMDP formulations (Bandyopadhyay et al., 2013; Egorov, Kochenderfer, and Uudmae, 2016; Ong, Png, Hsu, and Lee, 2010).

In cases where a MOMDP model is sufficient, potentially significant improvements in computational complexity can be achieved relative to using an equivalent POMDP model (Ong, Png, Hsu, and Lee, 2010)—due to the reduced dimensionality of the belief space. This is an important consideration for the proposed unified approach to operational decision making and system health management, as discussed in Section 4.4. Further information on the role of MOMDPs in this work can be found in Sections 6.5.1, 6.3.4, and 7.2.1.

2.5 Modeling for MDP and POMDP Problems

Representing and constructing explicit probability distributions for state transitions models \(T\) and observation models \(Z\)—as required by value iteration, policy iteration, and exact POMDP solution methods—may be difficult for realistic complex
systems. A number of MDP and POMDP solvers have been developed, however, that only require state transition or observation samples (some are described later in this chapter). Such samples can be produced by a generative model (i.e., a system simulator), with such models more readily available for systems of interest.

In the fully observable (MDP) case, a generative model $s', r = G(s, a)$ stochastically produces the next state $s'$ and reward $r$ given the current state $s$ and action $a$. In the partially observable (POMDP) case, a generative model $s', o, r = G(s, a)$ also outputs an observation $o$. The physics-based simulator of the VIPER rover, described in Chapter 6, is an example of the latter generative model type.

2.6 Monte Carlo Tree Search

MCTS is an online MDP solver that builds a partial policy (search) tree by using Monte Carlo simulations. An MCTS search tree contains alternating levels of state and action nodes, representing a subset of possible future scenarios. Each state node has branches to child action nodes. Action nodes, in turn, have branches to subsequent states.

The algorithm does not require an explicit state transition model $T(s, a)$, but can instead use a generative model $G(s, a)$ (an explicit model can be “converted” to a generative model by coupling it with a random number generator, but the opposite is not true). Use of a generative model is assumed in the rest of the description.

MCTS is an anytime algorithm—an important property that means that the algorithm iteratively improves its solution and, if interrupted, will return the best solution found up to that point. The standard MCTS algorithm starts by assigning the current state $s_t$ to be the root node of the search tree and then builds the rest of the tree by repeating the following four steps:

1. Selection. This part of the process takes place within the tree already built. At each state node an action branch is selected using a $Q(s, a)$-based preference criterion, often computed via one of the Upper Confidence Bound (UCB) algorithm variants (Auer, Cesa-Bianchi, and Fischer, 2002). The tree is traversed through the action node to a next-level state node, as determined by $G(s, a)$. 
2. **Expansion.** Once an action node with no children is reached, expansion of the tree is performed. \( G(s, a) \) is used to generate a new state and create a corresponding state node, with children nodes underneath it for each action in the MDP action space.

3. **Simulation.** After the tree is expanded, a simulation is performed using \( G(s, a) \) and following a predetermined policy (frequently a random policy, but could also be a domain-specific heuristic). The simulation is terminated after a certain number of steps or when a terminal state is encountered. If a reward discount factor \( \gamma \in (0, 1) \) is used, the simulation can also be stopped once future rewards become negligible due to discount factor compounding.

4. **Update.** The cumulative reward value obtained during the simulation is then back-propagated to update \( Q(s, a) \) values for action nodes visited during the selection step.

Repetition is stopped when a termination criterion is reached, such as the maximum execution time or the maximum number of iterations. The action at the root node with the highest \( Q(s, a) \) value is then returned as \( \pi(s_t) \).

### 2.7 Partially Observable Monte Carlo Planning

The Partially Observable Monte Carlo Planning (POMCP) algorithm is a straightforward extension of MCTS for partially observable domains (Silver and Veness, 2010). Instead of a partial search tree of states, POMCP builds a partial search tree of histories. A particular variant of UCB, Upper Confidence Trees (UCT), is adopted for in-tree action selection. The tree contains a node \( T(h) = (N(h), V(h), B(h)) \) for each represented history \( h \), with \( N(h) \) being the number of times that node has been visited, \( V(h) \) — the value of history \( h \), and \( B(h) \) — a set of unweighted state samples, i.e., *particles* (Del Moral, 1996), forming an approximate belief \( \hat{b}_h \). \( V(h) \) is estimated as the average return of all simulations starting at \( T(h) \). Values and visit counts for history-action pairs are also maintained.
The algorithm generally follows the same pattern as MCTS in constructing the search tree. Each simulation starts with a state sampled from the current belief in the tree root node. In the first, in-tree phase of the simulation, when child (action) nodes exist for all history nodes, actions are selected by using the UCB1 procedure (Auer, Cesa-Bianchi, and Fischer, 2002). In UCB1, the value of an action is augmented by an exploration bonus that is highest for rarely tried actions. For a POMCP history-action pair, this augmented value is computed as

\[ V \oplus (ha) = V(ha) + c \sqrt{\frac{\log N(h)}{N(ha)}}. \]  
(2.6)

Actions are selected to maximize the augmented value, i.e., \( \arg \max_a V \oplus (ha) \). The scalar constant \( c \) determines the relative ratio of exploration to exploitation. When \( c = 0 \), UCB1 selects actions greedily, i.e., picks the one with the highest current value. For each history \( h \) traversed during simulation, the corresponding belief state \( B(h) \) is updated to include the simulation state. In the second phase of a simulation, once a fringe node is reached, actions are selected via a predefined history-based policy (e.g., a random policy). After a simulation concludes (when, for example, the maximum number of steps is reached), exactly one new node is added to the tree, corresponding to the first new history generated during that simulation.

Once a termination criterion is reached for tree construction (e.g., the maximum number of simulations is exceeded), the action at the root node with the highest value is returned as \( \pi(b_t) \).

## 2.8 Determinized Sparse Partially Observable Trees

Due to the properties of UCB-style action selection, the worst-case running time of POMCP is rather poor: \( \Omega(\exp(\exp(\ldots \exp(1) \ldots))) \), nested \( D - 1 \) times, where \( D \) is the search/policy tree depth (Coquelin and Munos, 2007). Determinized Sparse Partially Observable Trees (DESPOT)—a recent POMDP solver also based on MCTS
principles — avoids this issue by relying on a set of $K$ scenarios sampled a priori to construct policies for the current belief state (Bai, Cai, Ye, Hsu, and Lee, 2015; Ye, Somani, Hsu, and Lee, 2017). A DESPOT scenario for a belief $b$ is an infinite abstract random sequence:

$$\phi = (s_0, \phi_1, \phi_2, \ldots),$$  \hspace{1cm} (2.7)

where $s_0$ is a scenario starting state sampled according to $b$ and $\phi_i$ is a real number sampled independently and uniformly from $[0, 1]$. The $K$ start states of the scenarios form the approximate belief $\hat{b}_0$ (in the Anytime Regularized implementation of DESPOT, vectors of weighted particles are used to form the belief nodes of the tree). Numbers $\phi_i$ are used in a generative deterministic model $G(s, a, \phi)$ to produce next state-observation pairs $(s', o')$. When the model is simulated for an action sequence $(a_1, a_2, \ldots)$ under a scenario $(s_0, \phi_1, \phi_2, \ldots)$, it generates a simulation trajectory $(s_0, a_1, s_1, o_1, a_2, s_2, o_2, \ldots)$.

The simulation trajectory traces out a path $(a_1, o_1, a_2, o_2, \ldots)$ from the root of the tree. All of the nodes and edges on this path are added to the tree. Each belief node $b$ in the tree contains a set $\Phi_b$ of all scenarios it encounters. Repeating this process for every action sequence under every sampled scenario completes the construction of the tree. A standard belief tree of depth $D$ would have $O(|A|^D |O|^D)$ nodes, while a corresponding DESPOT has $O(|A|^D K)$ nodes for $|A| > 2$ because of the reduced observation branching.

Formally, a DESPOT policy $\pi$ is a policy tree derived from a DESPOT $\mathcal{D}$. A policy tree has the same root as $\mathcal{D}$, but only retains one action branch at each internal belief node (all of the observation branches are retained, however). The set $\Pi_{b_0, D, K}$ consists of all policies derived from DESPOTs of depth $D$, constructed with all possible $K$ sampled scenarios for a belief $b_0$ (this definition will be useful in the discussion on computational complexity in Section 4.4).
2.9 Belief Updating

Once the system’s next action is chosen and executed, its current belief state needs to be updated, i.e., the next belief state needs to be calculated. This can be done exactly using Bayes’ rule (Koller and Friedman, 2009):

\[
\begin{align*}
    b'(s) &= \frac{\int_{s' \in S} T(s' | s, a) Z(o | s, a, s') b(s) ds}{\int_{s' \in S} \int_{s \in S} T(s' | s, a) Z(o | s, a, s') b(s) ds ds'}. 
\end{align*}
\]

(2.8)

For discrete formulations, integrals can be replaced by sums. The integrals in Equation 2.8 may be difficult (if not impossible) to solve analytically in most realistic cases and expensive to compute numerically. Equation 2.8 also requires availability of explicit $T$ and $Z$ models.

For systems with linear dynamics and sources of uncertainty that can be described with Gaussian distributions, belief updating can be done using the Kalman filter (Kalman, 1960). Variants of the Kalman filter for nonlinear systems have also been developed, such as the extended Kalman filter (Smith, Schmidt, and McGee, 1962) or the unscented Kalman filter (Julier and Uhlmann, 1997).

In other cases, numerical techniques may need to be used. Particle filtering (Gordon, Salmond, and Smith, 1993) is a popular general approach for numerical belief updating, with many variants developed. In this thesis an adaptive sequential importance resampling particle filter (Del Moral, Doucet, and Jasra, 2012) is used both for the benchmarking problems in Chapter 5 and within SHERPA in Chapter 6.

2.10 Discussion

This chapter overviewed select concepts and algorithms from the field of decision making under uncertainty relevant to the discussion in the remainder of the thesis. Where appropriate, connections to system health management concepts are highlighted in preparation for the discussion in Chapter 3. For readers interested in further reading on the topic, Kochenderfer (2015) provides an extensive treatment of DMU methods.
Probabilistic models are covered in detail by Koller and Friedman (2009). For problems that involve model uncertainty, Sutton and Barto (2018) provide a thorough introduction to reinforcement learning, an area of artificial intelligence that combines decision making under uncertainty with model learning.

While sequential DMU methods have undergone significant development since value iteration and policy iteration were first introduced, their application to real-world problems may still require skill and a certain amount of experimentation. For all of the approximate MDP and POMDP algorithms presented in this chapter, there is a trade-off between solution quality and computational complexity, a trade-off that may need to be considered carefully. The choice of algorithm parameters, such as the exploration constant $c$ in the UCB1 algorithm (Section 2.7), can also have a material effect on the solution quality and require some experimentation. The same is true for some of the domain-specific elements, such as rollout (simulation) policies (MCTS, POMCP, and DESPOT) or belief state value bound estimation algorithms for some versions of DESPOT.

Overall, decision making under uncertainty remains a very active research field. Development of more capable approximate methods for MDPs and POMDPs is continuing apace, including those taking advantage of progress in massively parallel computing (Cai, Luo, Hsu, and Lee, 2018; Lee and Kim, 2016). Work is also being done on deep learning methods for DMU (e.g., Karkus, Hsu, and Lee, 2017). Finally, while this thesis focuses on DMU for a single agent, active research is being conducted on multi-agent formulations. Multi-agent partially observable Markov decision process (MPOMDP) is one such formulation, where all agents are assumed to share their actions and observations — i.e., the joint history — at each decision step (Messias, Spaan, and Lima, 2011). Control in a MPOMDP is assumed to be centralized. Decentralized partially observable Markov decision process (Dec-POMDP) is a more general formulation, where the assumption that the joint history is always shared among the agents is not made (Oliehoek and Amato, 2016).
Chapter 3

System Health Management

Health management of complex systems has evolved from simple automated alarms into a subfield of artificial intelligence with techniques for analyzing off-nominal conditions and automatically generating responses to them. This chapter provides an overview of the prevalent system health management methodology, while also pointing out its general limitations. The chapter then focuses on a relatively recent addition to modern SHM — prognostics — and examines why prognostic analysis is not meaningful for most real-world systems and why it is challenging to implement effectively for the rest.

3.1 Introduction

System health management has traditionally included fault detection, isolation (diagnosis), and recovery components, with fault prognostics being a more recent addition (these concepts are defined more formally in Section 3.2). Fielded, non-experimental examples of system health management in aerospace applications have so far been limited to fault detection and diagnostics (Ezhilarasu, Skaf, and Jennions, 2019). One of the first production examples of SHM capability is the Central Maintenance Computer in the Boeing 777 airliner (Felke, 1995; Ramohalli, 1992), designed to diagnose faults in the safety critical components of the aircraft, e.g., landing gear actuators and electrical power systems. Later aircraft designs, such as the Boeing 787 and
the Airbus 380, also include automated onboard fault detection and diagnosis (Scan-
dura, 2017). In spacecraft operations, automated fault detection is fairly common
(Iverson et al., 2012; Meß, Dannemann, and Greif, 2019), although automated diag-
nosis capabilities have remained in the development/experimental phase for the most
part (Ezhilarasu, Skaf, and Jennions, 2019; Tipaldi and Bruenjes, 2015). In own
prior work, a fault diagnosis and recovery system was prototyped for the propulsion
components of the X-34 experimental space plane, handling faults such as stuck fuel
valves and malfunctioning sensors (Balaban, Maul, Sweet, and Fulton, 2004; Meyer
et al., 2003). Another SHM system was developed for a robotic Mars lander sample
acquisition drill (Balaban, Cannon, Narasimhan, and Brownston, 2007). Examples
of faults handled by that system included drill bit jamming and drill auger binding
to the surrounding material.

It is implicitly assumed that a system of interest to this work is subject to degra-
dation processes affecting performance within its expected useful life span. For the
SHM discussion to follow it is useful to classify systems into two general categories:
those with uncontrolled degradation processes and those with controllable degrada-
tion processes.

The uncontrolled degradation category is defined to include not only those
system types for which control over their degradation processes is not available or
required, but also those operating on predefined (open-loop) control sequences, such
as industrial robots performing the same sets of operations over extended periods of
time. Also included are system types where degradation is considered uncontrolled
within some time interval of interest (decision horizon). The rate of degradation
is influenced by internal (e.g., chemical decomposition) and external factors (e.g.,
temperature of the operating environment). In addition to industrial robots, examples
of system types with uncontrolled degradation processes include bridges, buildings,
electronic components, and certain types of rotating machinery, such as electrical
power generators.

In systems within the controllable degradation category, degradation processes
are influenced not only by the same types of internal and external factors as for the
first category, but also by control actions, either directly or indirectly. Most of the
discussion in this chapter is applicable to both categories, although systems with controllable degradation processes would, naturally, benefit more from active health management.

Throughout the chapter, a robotic exploration rover operating on the surface of the Moon (a simplified representation of the VIPER rover) is used as the main running example of a complex system with controllable degradation processes. The rover is solar-powered and stores electrical energy in a rechargeable battery. It is operating in a polar region, where solar illumination of a particular location can change quickly due to low sun angles above the horizon and resulting shadows from terrain features. Where it benefits the discussion, other examples of systems and system components from the aerospace domain are also introduced.

The next section (3.2) reviews the prevailing contemporary approach to SHM, outlines the general issues hampering its effectiveness, and illustrates them through numerical examples. Section 3.4 then focuses on prognostics, detailing why prognostics is not meaningful for systems with controllable degradation processes and why it may be challenging to implement effectively for systems with uncontrolled degradation processes. Numerical examples are also provided for illustration purposes. More than just identifying the issues with the current way of implementing SHM, however, the intent of this chapter is to provide the motivation and lay the groundwork for the proposed way forward—a unified DM/SHM approach presented in Chapter 4.

### 3.2 Contemporary SHM methodology

A typical contemporary SHM integration approach is shown in Figure 3.1. A DM subsystem generates an action $a_{t, DM}$, aimed at achieving operational objectives. The plant executes $a_{t, DM}$ and an observation $o_t$ is generated and relayed to both DM and SHM. DM computes $a_{t+1, DM}$ on the basis of $o_t$, while SHM analyzes $o_t$ for indications of faults (defined here as system states considered to be off-nominal) and, if any are detected, issues a recommendation on mitigation or recovery $a_{t+1, SHM}$ to the DM subsystem or, in some cases, as a command directly to the plant (Valasek, 2012). SHM may select $a_{t+1, SHM}$ from the general system action space $A_{DM}$ or, if defined,
from an additional $A_{SHM}$ space containing only actions specific to system health.

![Diagram](image.png)

**Figure 3.1**: A typical system architecture with SHM

Figure 3.2 details the SHM subsystem. The fault detection module corresponds to the traditional red-line monitors detecting threshold-crossing events of sensor values, represented on the diagram by a Boolean **fault detection** function $F$. If a fault is detected ($F(o_t) = true$), **fault isolation** and **diagnosis** (or **identification**) are performed, generating a vector of fault descriptors $f_t$ (Daigle and Roychoudhury, 2010). Each fault descriptor typically identifies a component, its fault mode, and its fault parameters (Daigle and Roychoudhury, 2010). There are diagnostic systems that also include an estimated fault probability in the descriptor (Narasimhan and Brownston, 2007). If the uncertainty of the diagnostic results is deemed too high (e.g., $f_t$ consists of only low-probability elements), **uncertainty management** is sometimes performed in order to obtain a better estimate of the current system condition (Lopez and Sarigul-Klijn, 2010).

Some recent SHM implementations then pass $f_t$ to a **prognostic** module (Roychoudhury and Daigle, 2011). In the SHM context, the intended goal of the prognostic module is to predict, at time $t_p$ (here $t_p = t$), whether and when faults will lead to system **failure** (defined as inability to perform its assigned function) within the window $[t_p, t_p + H]$ of a prediction horizon $H$ (the terms **prediction horizon** and **decision horizon** are equivalent for our purposes). In prognostics literature, time of failure is commonly synonymous with the term **end of [useful] life** (EoL). Equivalently, the goal of prognostics can be defined as predicting the system’s **remaining useful life** (RUL). In Figure 3.2, the prognostic prediction is defined as a probability density of EoL given the set of system faults at time $t$, $p(\text{EoL} | f_t)$. Uncertainty management is sometimes also prescribed following prognostic analysis, meant to improve the prediction if the confidence in it is low (Wang, Youn, and Hu, 2012). If a prognostic module is part of an SHM sequence, the term **Prognostics and Health Management**...
Management (PHM) is used by some instead of SHM in order to emphasize the role prognostics is playing in managing the system’s lifecycle (e.g., Ferrell, 2000).

Finally, $p(\text{EoL} \mid f_t)$ and $f_t$ are passed to the fault mitigation and recovery component to select an action $a_{t+1,\text{SHM}}$ from the action set $A_{\text{SHM}}$ in order to mitigate or recover from faults in $f_t$. As part of this process, operational constraints may be set for those faulty components that cannot be restored to nominal health. If functional redundancy exists for such components, their further use may be avoided.

### 3.3 General Issues

There are two general issues inherent in the current SHM approach:

1. DM and SHM objectives may at times be incompatible;

2. Separated system models may result in inferior solutions both from DM and SHM points of view.

In the following subsections the issues are illustrated via numerical examples where current SHM is contrasted with a unified DM/SHM approach. In the latter case it is
assumed that there is only one set of system objectives and that system models are combined. Such a unified approach will then be explored more fully in Chapter 4.

3.3.1 Incompatible DM and SHM objectives

As defined earlier in the thesis (Chapter 2), the purpose of decision making is to maximize a utility function $U$ that numerically represents achievement of operational goals. The objective of system health management, on the other hand, is to preserve a system’s ability to function as intended (Chapter 1), i.e., if an abstract composite metric $H$ of system health is defined, the goal of SHM is to maximize that metric. While it may appear at first that the two objectives are complimentary — after all, it would seem that a healthy system is more likely to achieve its operational goals than one with diminished capabilities — there may be situations where pursuing both objectives may result in subpar decisions, as the next example illustrates.

Example 3.1. The rover starts at $wp_0$ in Zone 1 (Figure 3.3) with 500 Wh (out of the overall 1500 Wh capacity). The solar panels can charge the battery at a rate of 250 W and system health $H$ is defined simply as the battery state of charge. Sunlight in Zone 1 will last for another twelve hours.

<table>
<thead>
<tr>
<th>Zone 1</th>
<th>Zone 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$wp_0$</td>
<td>$wp_1$</td>
</tr>
<tr>
<td>drive (4 h)</td>
<td>drive (4 h)</td>
</tr>
</tbody>
</table>

Figure 3.3: Balancing SHM and DM objectives (Example 3.1)

Two actions are available in the initial state $s_0$ at $wp_0$ (Figure 3.4): skip charging ($a_1 = +0$ Wh) and charge to full capacity ($a_2 = +1000$ Wh). In Figure 3.4, the unitless numbers are Watt-hours of energy going in or out of the battery. Time (in hours) at each state is denoted as ‘[t] h’.

The rover needs to perform a two-hour stationary science activity at $wp_1$ and be able to arrive at $wp_2$, the next recharging point. The prior probability of the activity at $wp_1$ needing to be redone (a two-hour delay) is 0.5. The science payload power consumption is 200 W, resulting in a net-positive power
flow \((250 \text{ W} - 200 \text{ W} = +50 \text{ W})\).

Driving times from \(wp_0\) to \(wp_1\) and from \(wp_1\) to \(wp_2\) are four hours, with the average of 300 W consumed by the rover’s mobility components, resulting in a net-negative power flow \((250 \text{ W} - 300 \text{ W} = -50 \text{ W})\). If operating without sunlight, a 150 W heater needs to be used to keep the batteries and electronics warm, thus resulting in a net-negative power flow of \(-300 \text{ W} - 150 \text{ W} = -450 \text{ W}\) for driving and \(-200 \text{ W} - 150 \text{ W} = -350 \text{ W}\) for stationary science activities.

According to the general SHM objective of maximizing a system health metric \(\mathcal{H}\) (battery charge, in this case), the action chosen at \(wp_0\) is \(\pi_{\text{SHM}}(s_0) = a_2\) and so the battery is recharged to its full capacity, 1500 Wh. After the science activity at \(w_1\) is completed, an assessment is made that the activity needs to be repeated. The two-hour delay means that the entirety of the \(wp_1 \rightarrow wp_2\) segment needs to be done without sunlight, resulting in complete battery depletion before \(wp_2\) is reached (a deficit of 300 Wh).

In computing a unified policy, however, where SHM actions are considered in the context of the overall mission, all four scenarios depicted in Figure 3.4 would play a role. If \(a_1\) is chosen, the expected amount of battery charge remaining would be \(Q_1 = \ldots\)
0.5 \cdot 200 \text{ Wh} + 0.5 \cdot 300 \text{ Wh} = 250 \text{ Wh}. \text{ For } a_2: Q_2 = 0.5 \cdot 400 \text{ Wh} + 0.5 \cdot (-300) \text{ Wh} = 50 \text{ Wh}. \text{ Action } a_1 \text{ (no recharge) would be selected and, with the two-hour delay at } wp_1, \text{ the rover would arrive at } wp_2 \text{ with } 300 \text{ Wh still remaining.}

For the upcoming discussion on DM/SHM unification in Chapter 4, this example is intended to highlight the first benefit of unification: the ability to naturally take operational objectives and constraints into account when making a system health recovery decision. The next example, on the other hand, illustrates how DM may benefit from a unified action space and access to health-related models.

### 3.3.2 Separated System Models

As illustrated on the diagram in Figure 3.2, conventionally DM and SHM are implemented as separate components, with their own system models and state, action, and observation spaces. For instance, in the prototype propulsion SHM component for the X-34 space plane (Balaban, Maul, Sweet, and Fulton, 2004; Meyer et al., 2003) the same sensors readings were monitored by DM and SHM components. However, the actual generation of observations from sensor signals (and thus their interpretation) was not coordinated and the two components maintained their own state spaces. SHM action space $A_{\text{SHM}}$ was a limited subset of $A_{\text{DM}}$ (e.g., containing commands for opening and closing valves) and while SHM had visibility into DM commands already issued, it had no insight into future vehicle actions. Similarly, there were no provisions for DM to have access to information internal to SHM. The next numerical example uses the simplified lunar rover case to illustrate how such separation of the two functions can result in suboptimal outcomes.

**Example 3.2.** The rover is traveling from $wp_0$ to $wp_1$ (flat terrain) when a decision is made at point A to make a detour and attempt data collection at a scientifically valuable $wp_2$, requiring a six-hour climb up a steep hill (Figure 3.5). The one-hour data collection activity at $wp_2$ must be completed before the loss of illumination in ten hours.

After completing the two-hour climb up to point $B$ (1/3 of the way up), it is observed that the internal temperature of one of the drive motors has risen
to $T_m = 60^\circ C$ from the nominal $20^\circ C$. At $T_m = 80^\circ C$, there is a significant risk of permanent damage and failure of the motor.

![Figure 3.5: Separated versus unified SHM/DM models (Example 3.2)](image)

If SHM on the rover consists of traditional fault detection, diagnosis, and mitigation/recovery components only, it may diagnose the fault to be *increased friction*, mark the component (motor) as faulty, and set a constraint on the acceptable terrain types to either *flat* or *downhill*. It would then invoke the recommended action for this fault from $A_{SHM}$: *stop and cool down* (until the motor temperature reaches $20^\circ C$).

If a prognostic component is present, it may predict that, at the current temperature increase rate, the RUL of the motor is one hour (with four hours of climb remaining to reach $wp_2$). The same mitigation action (*stop and cool down*) would be initiated and the same constraint on the current to the motor (and, thus, on the incline angle) may be set. After $T_m$ returns to nominal (which happens to take one hour), control is returned to DM. Given the new terrain type constraints and that the motor is classified as faulty, DM would command the rover to abort the detour, return to point $A$, and resume the drive to $wp_1$.

If, however, DM had *stop and cool down* as part of its action space and updated the state variables for the affected motor with the newly computed heat-up and cool down rates, an operational policy could be computed that optimizes the duration of driving and cool down intervals and allows the rover to reach $wp_2$ in time. For instance, if the rover drives for two hours, then stops for an hour to cool down the motor, it can still reach $wp_2$ in $2 + 1 + 2 + 1 + 2 = 8$ hours. With the science activity taking one hour, there would still be an hour in reserve before the loss of sunlight at $wp_2$.

This example illustrates the second benefit of DM/SHM unification, that having combined models and model spaces may allow for computation of operational policies
infeasible in a separated DM/SHM architecture. In some cases of degraded system performance, this could mean the difference between abandoning the remaining operational objectives and accomplishing at least some subset of them.

3.4 Prognostics

The addition of prognostics to SHM architectures was intended to provide an element of foresight into future degradation of system performance and, thus, enable proactive actions that reverse, mitigate, or accommodate it. While not objecting to this high-level goal itself, this section discusses both fundamental and practical reasons for why the commonly accepted definition of the prognostic problem is not meaningful for making decisions in most realistic system degradation cases. Chapter 4 will also show that the proposed unified DM/SHM architecture obviates the need for defining a distinct “prognostic” component altogether.

3.4.1 Definition

In general, prognostics can be defined as a procedure that predicts the time of occurrence of an event \( E \) (Daigle, Sankararaman, and Kulkarni, 2015). If \( \psi_E : S \rightarrow \mathbb{B} \) (where \( \mathbb{B} \triangleq \{0, 1\} \)) is an event threshold function, then \( t_E \triangleq \inf\{t \in [t_p, t_p + H] : \psi_E(s_t) = 1\} \) is the nearest predicted time of \( E \) (\( s_t \) is the state at time \( t \)). If the state evolution trajectory is non-deterministic, then \( p(t_E | s_{0:t_p}) \) is computed instead. Figure 3.6 illustrates prognostics on an example of a system’s EoL prediction, where the \( p(\text{EoL}) \) probability distribution is estimated based on a set of stochastic state evolution trajectories. If states cannot be directly observed, \( p(t_E | o_{0:t_p}) \) is computed. As defined, prognostics is only meaningful in a specific set of circumstances and two examples are used next to illustrate why this is so.

3.4.2 Systems with Uncontrolled Degradation Processes

There are two main desirable, interrelated attributes for a prognostic module: (1) low uncertainty in EoL estimation, so that a decision on mitigation or recovery actions
can be made with confidence, and (2) the ability to make a prediction far enough in advance that the actions can be successfully executed. In the case of uncontrolled degradation processes, this means that prognostic analysis is primarily useful for those systems that have long expected lifetimes, low degradation process uncertainty, or both. The first example illustrates why this is the case using a simple uncontrolled degradation process, shown in Figure 3.7.

**Example 3.3.** At time $t = 0$, the health state of a system is $s_0 = 1$ (states are scalar). According to a deterministic model, the nominal system health degradation rate is constant at $\dot{s}_n = 0.05/\Delta t$, where $\Delta t$ is the prediction time step, selected as the minimum time interval within which a change in the system’s health is expected to be detectable. A stochastic model predicts the
probability of the nominal degradation rate \( \dot{s}_n \) within any time step as \( p_n = 0.8 \) and the probability of a higher degradation rate (\( \dot{s}_h = \dot{s}_n + \epsilon/\Delta t \)) as \( p_h = 0.2 \). Assume \( \epsilon = 0.05 \).

The objective for both models is to predict EoL, i.e., the smallest \( t \) for which \( s \leq 0 \). For this example, the prediction uncertainty is defined as \( \sigma(t_p) = |E[E_{\text{LoL}}(t_p)] - E[E_{\text{LoL}}(t_p)]| \), i.e., the absolute difference between the expected EoL values computed by the two models at a prediction time \( t_p \). A requirement is set on the maximum EoL prediction uncertainty as \( \sigma_{\text{max}} = 1\Delta t \).

For this example, the health state is assumed to be fully observable and the fraction of full health remaining at \( t_p \) is defined as \( \rho_p = s_p/s_0 \). In Figure 3.7, a prediction is shown to be made at \( t_p = t_0 \), with \( s_p = s_0 \), \( \rho_p = 1 \), and the prediction horizon \( H = 20\Delta t \). Since

\[
E[E_{\text{LoL}}(t_p)] = \frac{\rho_p}{\dot{s}_n} \tag{3.1}
\]

and

\[
E[E_{\text{LoL}}(t_p)] = \frac{\rho_p}{p_h \dot{s}_n + p_h \dot{s}_h} = \frac{\rho_p}{(1 - p_h) \dot{s}_n + p_h (\dot{s}_n + \epsilon/\Delta t)} = \frac{\rho_p}{(\dot{s}_n + p_h \epsilon/\Delta t)}, \tag{3.2}
\]

then

\[
\sigma = \left| \frac{\rho_p}{\dot{s}_n} - \frac{\rho_p}{(\dot{s}_n + p_h \epsilon/\Delta t)} \right| = \rho_p \left| 20 - \frac{1}{(0.05 + p_h \epsilon)} \right| \Delta t. \tag{3.3}
\]

In the last equation, the value for the nominal degradation rate \( \dot{s}_n \) is substituted in order to focus on the effects of the degradation rate uncertainty. As can be seen in Figure 3.7, EoL is reached by both models within the prediction horizon. However, from Equation 3.3, \( \sigma = 3.33\Delta t > \sigma_{\text{max}} \).

For the requirement on \( \sigma \) to be satisfied, either \( \rho_p \) (health fraction remaining), \( p_h \) (the probability of deviations from the nominal degradation rate), \( \epsilon \) (the magnitude of deviations), or some combination of them needs to be reduced. If \( p_h \) and \( \epsilon \) are kept the same, with \( \rho_p \) set to 0.25 the value of \( \sigma \) is 0.83\( \Delta t \). However, \( t_p \) now needs to be \( \approx 15\Delta t \), with only 5\( \Delta t \) left until failure (i.e., RUL, the remaining useful life). RUL corresponds to the time available to either replace/repair the uncontrolled system or initiate an emergency response. For a quickly degrading system with \( \Delta t = 1 \text{ s} \), RUL
would only be 5 s, which is likely enough time for an emergency response, but not for repair or replacement. In practice, in systems where responding to a fast-developing uncontrolled degradation process is important, estimating $p(\text{EoL})$ is unlikely to bring tangible benefits. For instance, if pressure starts building quickly in a fuel tank of an ascending crewed rocket, the launch abort system (emergency response) is likely to be activated by the exceedance of a predefined pressure limit or pressure increase rate (i.e., functions of fault detection). Computing whether the tank breach will occur in 10 or 12 seconds will not materially influence that response. It may also be difficult to have high confidence in a prognostic prediction from a limited number of observations of a fast-developing degradation process.

There are some processes, mostly chemical in nature, with mid-range degradation rates (minutes, hours, days) that could be considered uncontrolled. One example of such a process is the decomposition of hydrogen peroxide, which is used as propellant in the attitude control thrusters of a Soyuz spacecraft descent module (Hall and Shayler, 2003). Over time, some of the hydrogen peroxide breaks down into water and oxygen, decreasing the amount of usable propellant. This limits the on-orbit life of the vehicle to about 200 days. For processes like this, extrapolation of the degradation function could be part of a caution and warning mechanism.

What follows from Example 3.3 is that if the degradation process uncertainty is relatively high or varies significantly over time, a short prediction horizon (compared to the overall system lifetime) may be necessary to limit the uncertainty propagation and result in a usable $\sigma$. In this case, systems with longer lifetimes are more suitable for applying prognostics. For example, if a bridge failure can be predicted three years in advance with an accuracy of $\pm 1$ year, that can still be a useful prediction.

However, while many systems with uncontrolled degradation processes could be classified as having long expected lifetimes, there exists a number of fundamental practical difficulties in performing effective prognostic analysis for them. One of the primary issues stems directly from the typically long lifetimes (often decades). In order to establish trust in the degradation models, they need to be adequately tested using long-term observations from “normal use” or observations from properly formulated accelerated testing. Useful and valid (from the statistical point of view)
“normal use” degradation data sets are rare for most long-duration degradation processes (Eker, Camci, and Jennions, 2012; Heng, Zhang, Tan, and Mathew, 2009). Run-to-failure data sets collected in “normal use” conditions are particularly rare as, in practice, operating a system to complete failure (or even close to it) may have cost or safety implications (Eker, Camci, and Jennions, 2012). Accelerated degradation efforts in controlled settings are, therefore, quite common. If an accelerated degradation regime is proposed, however, what needs to be clearly demonstrated is that:

1. **The regime can be used as a substitute for real-life degradation processes.** For instance, while Rigamonti, Baraldi, Zio, Astigarraga, and Galarza (2016) and Celaya, Kulkarni, Biswas, and Saha (2011) use thermal and electrical overstress to quickly degrade electrical capacitors and predict their time of failure (by using empirical equations), their work does not make a connection to real-life degradation, which takes place at lower temperatures and voltage/current levels. Similar issues are highlighted by Dao, Hodgkin, Krstina, Mardel, and Tian (2006a,b) for composite materials, where mechanical, thermal, and chemical processes result in complex interactions during aging.

2. **There is a known mapping from the accelerated timeline to the unaccelerated timeline.** To illustrate this requirement, electronic components are again used as an example. In an overview of condition monitoring and prognostics of insulated gate bipolar transistors, Oh, Han, McCluskey, Han, and Youn (2015) note that numerous specialized fatigue models have been constructed that aim to predict *cycles-to-failure* under repetitive cycle loading in accelerated aging experiments. The application of various general fatigue analysis models, e.g., Coffin (1954), Miner (1945), or Matsuishi and Endo (1968), have also been proposed for predicting failure on the basis of usage cycles. As Oh, Han, McCluskey, Han, and Youn (2015) describe, however, there are significant practical challenges in using any of these methods to estimate a component’s lifetime under realistic conditions and, as importantly, on realistic time scales. The first challenge is with accurately converting environmental and operational
loading conditions into thermomechanical stresses. The second is with defining and extracting the number, amplitude, and duration of stress cycles when a system is subjected to a potentially complex usage profile. The third challenge is with properly accounting for damage accumulation and distribution in the various components of the system. In regard to the second challenge, Ciappa, Carbognani, Cova, and Fichtner (2003), for instance, proposed a method to decompose a complex usage profile into elementary cycles and compared it to the Coffin-Manson model (Coffin, 1954). The comparison was done on an accelerated aging data set only. While the proposed model performed better on the data set than the simpler Coffin-Manson model, the authors concluded that neither would be able to fully represent all of the mechanisms governing degradation under realistic use, including stress relaxation, anisotropic effects, and microstructural changes. The authors also note that even though more complicated models may be able to account for some of these additional mechanisms, the number of free parameters associated with such models would make their calibration virtually impossible.

There are subfields of prognostics where accelerated aging regimes may be viable, such as for metallic structures of aircraft or rotating machinery (where mechanical degradation factors could be assumed dominant). However, the issue of high uncertainty of degradation trajectories still arises, even for test articles made out of isotropic materials and aged under uniform conditions (Meng, 1994; Virkler, Hillberry, and Goel, 1979). Finite element modeling may help alleviate degradation trajectory uncertainty in specific cases, albeit at a significant computational cost (Heng, Zhang, Tan, and Mathew, 2009). Some of the other challenges preventing effective prognostics for systems with uncontrolled degradation include the accuracy of estimating the actual state of health, the effects of fault interactions, and the effects of system maintenance actions (Heng, Zhang, Tan, and Mathew, 2009).

If these challenges are successfully overcome and the failure mechanisms of a component are understood well enough to develop useful degradation models, a different question then arises: should the design or usage of the component be changed to mitigate these mechanisms? While in some cases this may not be feasible, in others it
may be the simplest and most reliable way of improving safety and reducing system maintenance requirements (Bathias and Pineau, 2013). A redesign or change in usage would, on the other hand, make the degradation models obsolete. The next tier of degradation modes would then need to be analyzed and modeled, possibly followed by another redesign. Thus, analysis intended for the development of degradation (prognostics) models instead becomes part of the design improvement cycle.

In some instances, in addition to maintenance optimization, prognostics of long-lifespan components have been proposed as a means of ensuring safety of the overall system, even when the system’s periods of active operational usage are substantially shorter than the expected lifetime of such components (e.g., Tang, Roemer, Bharadwaj, and Belcastro, 2008). To see why relying on prognostics for this function may not be advisable, consider the following example. A prognostic algorithm monitors the condition of an aircraft turbofan engine fan disk using a degradation model $M$. A fan disk degradation $\Delta t$ (the minimum time interval within which a change in component health is expected to be detectable) is measured in months, while a typical flight lasts several hours. If an in-flight failure of the fan disk is predicted to occur in, for example, $1 \pm 0.25$ hours, can $M$ be trusted for determining a course of action, given that the impending failure was not forecast (and addressed) well before the current flight?

An argument can be made that even a healthy long-lifespan component can be damaged due to an unexpected (paroxysmal) event and projected to fail before the expected flight completion. This case is similar to the one described earlier (crewed rocket launcher), where a catastrophic short-duration degradation process takes place. Even assuming that an appropriate process model is available and that its parameters can be determined quickly and accurately, the optimal response will still likely be an emergency procedure (e.g., an emergency landing), prompted by an off-nominal state estimate rather than the prognostic prediction. If a coupling exists between emergency actions and the degradation process, then the case becomes that of a controllable degradation process, discussed in Section 3.4.3.

For systems with uncontrolled degradation processes that are, in fact, suitable for health management based on prognostics, the action space $A$ is often limited to: (a) no
action, (b) replacement, or (c) repair to achieve a nominal operating condition. Even so, rather than following the sequence in Figure 3.2—i.e., computing $p(\text{EoL})$, then deriving decisions using that information—it would be advisable for any predictive analysis to be driven instead by the requirements of a decision making procedure. This would allow for the formulation of an appropriate problem, development of suitable models, and determination of the maximum prediction (decision) horizon needed. For instance, if domain knowledge informs that variability in the system behavior past some health index $h_{\text{min}}$ is too great for choosing actions with sufficiently high confidence (e.g., beyond $h = 0.8$ for the degradation process depicted in Figure 3.6), then EoL can be redefined as $h_{\text{min}}$ and system dynamics beyond $h_{\text{min}}$ need to be neither modeled nor computed, potentially freeing computing resources for estimating system behavior up to $h_{\text{min}}$ with more accuracy. In another scenario, assume that for some slow-degrading system the maximum execution time of any action in $A$ (e.g., arranging for and performing system replacement) is $2\Delta t$. Then the question that needs to be answered at every decision step is not “When will the system fail?”, but rather, “Will the system fail within the next $2\Delta t$?”. The frequency of decision steps can then also be set to $2\Delta t$.

3.4.3 Systems with Controllable Degradation Processes

In a realistic controlled system, uncertainty is often present in state transitions. When the system’s control actions can affect its degradation processes, prognostics, as defined at the beginning of Section 3.4, and the PHM version of the sequence depicted in Figure 3.2 are no longer meaningful—for two key reasons. First, not having the knowledge, at $t_p$, of the future system actions, a PHM algorithm will have to either (a) rely on some precomputed plan to obtain $a_{t_p+1:H}$ for its predictive analysis (a plan that can quickly become obsolete due to action outcome uncertainty) or (b) use a random policy (which can, for instance, result in actions inappropriate for some system state being selected with the same probability as the more appropriate ones). A random policy is also likely to result in greater state uncertainty throughout the $[t_p, t_p + H]$ interval. Second, rerunning prognostic analysis after each action-initiated
state transition to account for new information (e.g., Tang, Hettler, Zhang, and De-castro, 2011) may not always be helpful. Once a suboptimal execution branch has been committed to, it may remain suboptimal regardless of future decisions. The following example illustrates these issues:

**Example 3.4.** The lunar rover needs to traverse an area with no sunlight, going around a large crater from waypoint \( wp_0 \) to the closest suitable recharge location at \( wp_4 \) (Figure 3.8). For this example, the rover’s battery state of charge is considered to be its health indicator. At \( wp_0 \) the battery state of charge is 1100 Wh.

There are three possible levels of terrain difficulty: *difficult* (requiring 600 Wh per drive segment), *moderate* (300 Wh per segment), and *easy* (200 Wh per segment). All drive segments are the same length. Probabilities of terrain types in different regions are shown in Figure 3.8.

The rover can go to the left, \( wp_0 \rightarrow wp_1 \rightarrow wp_4 \), or to the right, \( wp_0 \rightarrow wp_2 \rightarrow wp_4 \) (left and right are relative to the diagram). If going to the right, there is an option to detour around a smaller crater \( wp_2 \rightarrow wp_3 \rightarrow wp_4 \) (*easy* terrain with \( p = 1.0 \)) instead of going directly \( wp_2 \rightarrow wp_4 \).

A PHM algorithm used for decision support — running a sufficiently large number of simulations — would consider two possible execution scenarios along the left route: (L1) \( e_{\text{total}} = 1200 \text{ Wh}, \ p = 0.4 \) and (L2) \( e_{\text{total}} = 600 \text{ Wh}, \ p = 0.6 \) (\( e_{\text{total}} \) is the total energy consumed in a scenario). The expected energy consumption along the left route is, therefore, \( E[e_{\text{total}}, L] = 1200 \text{ Wh} \cdot 0.4 + 600 \text{ Wh} \cdot 0.6 = 840 \text{ Wh} \).
The algorithm would then consider four possible execution scenarios along the right route (assuming uniform random choice of action at $wp_2$): (R1) $e_{total} = 1200$ Wh, $p = 0.25$; (R2) $e_{total} = 600$ Wh, $p = 0.25$; (R3) $e_{total} = 1000$ Wh, $p = 0.25$; and (R4) $e_{total} = 700$ Wh, $p = 0.25$. Then, $E[e_{total, R}] = (1200 + 600 + 1000 + 700) \cdot 0.25$ Wh = 875 Wh. With $E[e_{total, L}] < E[e_{total, R}]$, the PHM algorithm commits to the left path. Note that prognostics algorithms typically generate less information to support action selection than what was presented here, computing an aggregate $p(EoL)$ only and not retaining potentially valuable performance data on individual execution trajectories (e.g., Daigle, Sankararaman, and Kulkarni, 2015).

A DM algorithm capable of sequential reasoning under uncertainty (Kochenderfer, 2015) would compute $E[e_{total, L}] = 840$ Wh in the same manner as the PHM algorithm, as there are no actions needing to be selected along the left route after $wp_0$. On the right side, however, the DM algorithm can make an informed choice at $wp_2$, based on observations made along $wp_0 \rightarrow wp_2$. This means having only two possible execution scenarios: (R1) if the terrain is observed as difficult, the detour through $wp_3$ is taken, and (R2) if the terrain is observed as moderate, the rover goes directly to $wp_4$. For R1, $e_{total} = 1000$ Wh, $p = 0.5$. For R2, $e_{total} = 600$ Wh, $p = 0.5$. The expected energy use is thus $E[e_{total, R}] = (1000 + 600) \cdot 0.5$ Wh = 800 Wh. With $E[e_{total, L}] > E[e_{total, R}]$, the algorithm chooses the right path.

Now assume that the true terrain condition both on the left and right sides of the crater is difficult. The left path ($wp_0 \rightarrow wp_1 \rightarrow wp_4$) will require 1200 Wh to traverse,
therefore a rover relying on the PHM algorithm will fall 100 Wh short and will not reach \( wp_4 \). A rover relying on the DM algorithm will expend only 1000 Wh (scenario R1), arriving at \( wp_4 \) with 100 Wh in reserve.

It may be suggested that the listed issues with the PHM approach could be eliminated if access to a precomputed operational policy \( \pi_{DM} \) is provided (rather than relying on some random policy). However, even if such a policy was accessible to PHM, that would still be insufficient. If, at time \( t \), \( p(EoL) \) is computed using \( \pi_{DM} \), then some \( a_{t+1,SHM} \) is executed on the basis of \( p(EoL) \), \( p(EoL) \) would immediately become invalid unless \( T(s_t, a_{t+1,SHM}, s_{t+1}) = T(s_t, a_{t+1,DM}, s_{t+1}) \). In order to properly take \( p(EoL) \) into account, a new policy \( \pi_{DM} \) would need to be computed, which PHM is not capable of doing on its own.

### 3.5 Discussion

This chapter described the currently prevalent approach to system health management and illustrated its limitations through a series of numerical examples. Among the limitations is the possibility of conflicting objectives with DM and difficulties with producing optimized solutions due to separated DM/SHM model spaces.

Techniques developed for fault detection and diagnosis, two of the traditional SHM components (Figure 3.2), can certainly be useful in estimating the current system state and perhaps even initiating automated response or emergency procedures in off-nominal conditions. Prognostics, on the other hand, has to make assumptions about future system actions in order to produce an estimate of the system’s remaining useful life. If no control capability exists for the system or if executing control actions has no effect on system health, computing EoL may be worthwhile (provided that appropriate degradation models exist). The same is also true if a fixed, open-loop command sequence can be assumed for the system within the prediction horizon. Most complex systems of interest are controlled in closed-loop manner, however, where future actions depend on future system states. In this case the prognostic procedure has to rely on random policies to run its simulations and produce an RUL probability distribution. As explained in the chapter, such a procedure would provide little, if
any, decision-making benefit.

It is important to note, however, that while the *prognostic procedure* may not be useful in most decision-making situations, that does not mean that a *model* $\mathcal{M}$ of some relevant degradation process cannot be of value. Quite the contrary, if such a model is successfully developed — given the challenges described in the chapter — it can be a helpful part of the overall system state transition model. Even so, rather than computing $p(\text{EoL})$ or the probability distribution for time of occurrence of some other event $E$, $\mathcal{M}$ would need to be of the form $T(s, a, s')$, describing the probability of transitioning to a particular state $s'$ as a result of action $a$. Further, $\mathcal{M}$ would need to be defined for $S \times A \times S$, i.e., for all valid combinations of system states (including environmental conditions, if so specified), actions, and follow-on states.
Chapter 4

Health-Aware Decision Making

Chapter 3 outlined the general issues with the contemporary approach to system health management, while also delving into the more specific ones with its prognostic component. This chapter proposes a way forward via a comprehensive unification of decision making and system health management and shows how the traditional system health management concepts map into the unified framework. The computational complexity of implementing the new approach versus a conventional DM/SHM architecture is also analyzed.

4.1 Introduction

A number of options for overcoming the limitations of the current SHM approach (Chapter 3) were examined over the course of this work, including improving the prognostic procedure through building sets of partial scenario trees and developing a coordination protocol between DM and SHM. It eventually became apparent, however, that the most promising way forward was to incorporate elements of system health management into the overall DM decision making process. The idea itself is not necessarily new. If system energy reserves, for example, are to be considered a metric of system health (as they sometimes are), then reasoning about system health has been part of automated decision making in a number of previous efforts—e.g., Spaan, Gonçalves, and Sequeira (2010), Eisner, Funke, and Storandt (2011), or

In own prior work, combining DM and SHM was also studied in application to UAVs (Balaban and Alonso, 2013). The problem of health-aware mission planning was formulated as a POMDP and an online Monte Carlo sampling algorithm was designed to provide vehicle trajectory recommendations. The algorithm was tested on scenarios where failure of one of the serially connected electrical motors led to excessive current draw and overheating of the second motor. Physics-based aerodynamic, thermal, and electrical models of the vehicle were used. An updated version of the algorithm was then deployed on a planetary rover prototype and tested in experiments where increased wheel motor friction, parasitic electrical load, and sensor faults were injected (Balaban, Narasimhan, Daigle, Celaya, et al., 2011; Balaban, Narasimhan, Daigle, Roychoudhury, et al., 2013). Once again, physics-based models were used by the algorithm to generate action recommendations.

This thesis takes a step beyond these prior efforts and proposes to treat all relevant degrading components in the system as, essentially, consumable resources and comprehensively unify operational decision making and system health management (with the exception of emergency response, as discussed later in the chapter). The terms unified and unification are chosen specifically to emphasize the idea of DM and SHM being performed within the same computational framework, rather than the two being integrated as separate components exchanging information.

In the past, limitations in computing hardware and algorithms would have made the proposed approach rather challenging to implement. Recent advances in both areas make it feasible, however. The POMDP solver presented in Chapter 5 is intended
to make the unified approach possible even for complex systems operating under state or action outcome uncertainty.

The rest of the chapter introduces a systematic view on DM/SHM unification, discusses its benefits, and illustrates how current SHM concepts map into the proposed approach without loss of functionality or generality. The unification framework is described in Section 4.2, followed by a discussion on emergency response functionality and its potential implementation methods in Section 4.3. Section 4.4 presents a computational complexity analysis of the unified approach and Section 4.5 concludes the chapter.

### 4.2 Unification Approach

Utility theory (Chapter 2) plays a central role in the proposed unified SHM/DM approach. The following two ingredients are key for a successful unification: (1) a state-based system modeling framework and (2) a utility (value) function describing operational preferences for the system. States can be vector quantities. For real-world problems, relevant elements of the operating environment are sometimes included in the state vector, either explicitly or implicitly (Ragi and Chong, 2014). For instance, the lunar rover state vector would certainly need to include the rover’s $x$ and $y$ coordinates, but may also include time $t$. These three elements allow indirect access to other information about the environment, e.g., solar illumination, ambient temperature, communications coverage, and terrain properties.

Similarly, health-related elements can be included in the same state vector. For the rover, the battery charge would likely be in the state vector already for operational purposes. Adding battery temperature, however, would allow for better reasoning about the state of battery health when combined with information on ambient temperature, terrain, and recharge current. Thus, including even a few health-related elements in the state vector (already containing information about the environment and the general state of the system) can have a multiplicative effect on the amount of information available. The resulting size of the state vector may also end up being smaller than the union of separately maintained SHM and DM state vectors, as
redundant elements are eliminated.

The reward function $R(s, a)$ encodes the costs and rewards of being in a particular state (or taking a particular action in a state) and can be used, through the utility function, to induce the desired behavior. For many realistic problems, the reward function needs to combine costs or rewards associated with different state components. Several approaches have been proposed (Keeney and Raiffa, 1993), with additive decomposition being an effective option in many cases. The key property of the function is that by mapping multiple variables to a single number it allows computing $U(s)$ or $U(s, a)$ and translating a potentially complex DM formulation into an abstract utility maximization problem.

4.2.1 SHM Concepts Within the Unified Framework

One notable consequence of health-related components being integrated into a common state vector is that, from the computational point of view, the concepts of fault and failure become somewhat superfluous. If subsets $S_{\text{fault}} \subset S$ or $S_{\text{failure}} \subset S$ are defined for the system, the framework described above will not do anything differently for them. The only essential subset of $S$ is $S_T$ (the terminal states). Failure states may be part of $S_T$ if they result in termination of system operations; however, goal (success states) are members of $S_T$ also. The only difference between them is in their $U(s)$ values. As long as a component fault or a failure does not lead to a transition to a terminal state, actions that maximize the expected value of that state will be selected — which, as it happens, implements the “fail operational” philosophy (NASA, 2012).

In the remainder of the thesis, this unified approach is referred to as health-aware decision making (HADM). The rest of the major SHM concepts are incorporated into HADM as follows. Fault detection and diagnostics are subsumed in belief estimation and updating, although these operations are, of course, used for nominal belief states as well. Uncertainty management can now be purposefully incorporated into the decision-making process by augmenting $A$ with information gathering actions (Bonet and Geffner, 2000; Levesque, 1996; Spaan, Veiga, and Lima, 2015;
Weld, Anderson, and Smith, 1998), evaluated and selected in the same context as other types of actions. For actively controlled systems, predictive simulations are simply an integral part of \( U(s) \) calculation, where \( T(s, a, s') \) serves as a one-step “prognostic” function (with degradation models, if any). Whereas prognostic algorithms applied to systems with controllable degradation are limited in their predictive ability due to the lack of knowledge about future actions, here the \( U(s) \) calculation process is an exploration of possible execution scenarios. It thus combines \( s' \) or \( b' \) estimation with sequential action selection.

### 4.2.2 Degradation Models

The role of degradation models in policy computation is perhaps worth discussing in more detail. As noted in Section 3.5, a useful degradation model would be in the form of a state transition function \( T : S \times A \times S \rightarrow [0, \infty) \) — or \( T : S \times S \rightarrow [0, \infty) \) in the uncontrolled degradation case — for some relevant subset of the state vector elements. In the Bellman equation (Equation 2.3), \( T(s, a, s) \) plays a role in computing the value of a state and, consequently, in computing an action policy. It would stand to reason then that a more accurate degradation model would result in more accurate state values and better (higher value) policies computed for the system. Conversely, an inaccurate (incorrect) model would result in inaccurate state values being computed and poor-performing policies derived on their basis. In that case, a random policy may perform better. However, having a low-fidelity degradation model or an uninformative model may lead to computing a conservative, yet still better-than-random policy.

As an illustration, consider an extreme example where the lunar rover’s battery is modeled to transition from any state to any other state with equal probability density, \( T(s, a, s') = c, \forall s \in S, \forall a \in A, \forall s' \in S \), where \( c \) is a constant (e.g., the model battery is just as likely to predict transition from being fully charged directly to fully discharged as to any other state). The rover is tasked with visiting a set of \( N \) waypoints, each with an associated numerical reward \( r_i, i \in \{1, 2, \ldots, N\} \). Waypoints can be visited in any order and the objective is to maximize the cumulative reward. The state transition model suggests that the worst case outcome for any of the actions is transitioning
to a terminal (fully discharged) state. A greedy policy that recommends going to the closest unvisited waypoint may then be optimal or possibly one that recommends going to the waypoint $wp_k$, where $k = \arg \max_{j \in \{1, 2, \ldots, M\}} \frac{r_j}{d_j}$, $M$ is the number of unvisited waypoints, and $d_j$ is the distance from the current waypoint to waypoint $wp_j$. While such policies would likely not be optimal had a better battery model been available, they may nevertheless be an improvement over a random policy.

### 4.2.3 Implementation

Figure 4.1 shows the overall HADM operational loop (assuming state and action outcome uncertainty). Once the initial belief $b_0$ is estimated at time $t_0$, either an offline $\pi^*$ is referenced or an online $\pi^*$ is computed to determine $a_0^*$ (best action). The action is executed by the plant, transitioning to a new (hidden) state, and generating an observation $o_0$. The observation is then used to update $b_0$ (typically through some form of Bayesian updating) and the process repeats until a terminal state is believed to be reached.

![Figure 4.1: The main loop of health-aware decision making](image)

A variety of algorithms capable of handling state or outcome uncertainty can be used to implement the proposed approach (Browne and Powley, 2012; Kochenderfer, 2015). In systems where both state and outcome uncertainty are not a factor, traditional state space planning algorithms (Ghallab, Nau, and Traverso, 2016) would not need any modifications to produce plans for spaces of state vectors that include health-related components.

In the discussion up to this point, it has been assumed that optimization of solutions is desired. If that is not the case, HADM can also be formulated from the satisficing perspective, e.g., as a constraint satisfaction problem (CSP). CSP formulation could be done using the same mathematical framework, with the problem solved by applying existing CSP algorithms (Frank, Jónsson, and Morris, 2000; Ghallab,
Nau, and Traverso, 2016). In a CSP formulation, hard constraints could be defined by assigning a reward of $-\infty$ to undesirable states and soft (weighted) constraints could be defined using finite negative rewards, with all the other rewards set to zero.

### 4.3 Emergency Response versus HADM

As mentioned in Section 2.3, finding exact optimal policies for realistic complex systems operating under state and outcome uncertainty is thought to be a PSPACE-complete problem. Approximate solution methods typically work online, constructing $\pi$ for the current belief $b$ (or a fully observable state $s$) based on beliefs/states reachable from $b$ within the decision horizon (Browne and Powley, 2012). They also typically perform targeted sampling from $S/B$, $A$, and $O$, thus guarantees of true optimality can be harder to provide. Therefore, the following is proposed in this thesis:

1. **System Emergency Response (SER)** should be defined as an automated or semi-automated process that is invoked to maximize the likelihood of preserving the system’s integrity, regardless of the effect on operational goals.

2. An emergency response policy $\pi_{\text{SER}}$ should be computed separately from $\pi_{\text{HADM}}$.

In the space domain, an example of SER is commanding a spacecraft into *safe mode* until the emergency is resolved (Rasmussen, 2008). In aviation, it could be executing recommendations of a collision avoidance system (Kochenderfer, Holland, and Chryssanthacopoulos, 2013) or performing an emergency landing (Atkins, Portillo, and Strube, 2006; Meuleau, Plaunt, Smith, and Smith, 2009). Introduction of a separate SER system would likely require the introduction of $S_{\text{SER}}$, an additional subset of $S$ that defines the states where $\pi_{\text{SER}}$ is invoked versus $\pi_{\text{HADM}}$. Once again, however, $S_{\text{SER}}$ will not necessarily only contain system fault and failure states. For instance, states where environmental conditions warrant emergency response (e.g., solar activity interrupting teleoperation of the rover) would be included as well. While from a system architecture point of view SER may bear some resemblance to the SHM module in Figure 3.1, there are two important distinctions. First, SER would
not operate in parallel with primary decision making, avoiding the potential for conflicting actions. Second, rather than being tasked with returning the system to a "healthy" state, SER is meant to merely transition the system to a state not in $S_{\text{SER}}$, then return control to HADM. Since the scope of the SER problem is likely to be much narrower than that of HADM, it opens the possibility of computing, verifying, and validating $\pi_{\text{SER}}$ offline (Kochenderfer, 2015).

As sensors, computing capabilities, and DM algorithms improve, the fraction of the system’s state space that is under the purview of SER will decrease. Still, the need for an independent, “safety net” SER is likely to still be there for safety-critical functions. SER would cover situations where the primary HADM system may not be able to produce a suitable solution in a desired amount of time, serving as an equivalent of human reflexive responses triggered by important stimuli versus the more deliberative cognitive functionality of the brain. It could also provide dissimilar redundancy for critical regions of $S$, essentially implementing the Swiss Cheese Model (Reason, 1990). In the model, multiple different layers of protection for important system functions are proposed, illustrated as slices of cheese. The holes in the slices, representing weaknesses in individual layers, vary in size and position from slice to slice (and, also, often vary in time). A system failure may occur if holes in all the protective layers happen to momentarily align. A greater number of dissimilar protective layers can, consequently, reduce the probability of failure.

4.4 Computational Complexity of HADM

A potential implementation issue for the proposed unified approach is that it may result in an increased computational complexity by operating on state, belief, observation, and action spaces that are higher-dimensional or larger (i.e., contain an increased number of elements in the same dimensions) compared with a separated SHM/DM formulation. In this section, the implications of $S/B$, $O$, and $A$ dimensionality or size increase for problems with different types of uncertainties present are examined. The worst-case scenario (from a computational complexity point of view) is generally analyzed by assuming that all spaces are continuously valued and,
therefore, contain an infinite number of elements per dimension. It is further assumed
that optimality (or near-optimality) is desired for both SHM and DM, therefore both
separated and unified SHM/DM cases are formulated as search problems. The last
assumption also implies that computed decisions are not limited to being single-step
reactive, but can be sequential.

Finally, it is assumed that the number of independent dimensions may increase
for unified state, belief, and observation spaces, but will remain the same for unified
action spaces. To see why, consider the following lunar rover example. An assumption
is made that in a separated SHM/DM implementation, a single action, originating
either from SHM or DM, is chosen at each decision interval. The DM action space
may contain a \textit{drive} command, parameterized with real-valued heading and speed,
thus forming its own class of actions with an infinite number of elements. The SHM
action space may contain a \textit{cool down} command, parameterized with real-valued
duration, forming another infinitely large class of actions. Merging the two action
spaces will not introduce additional dimensions, as at each time step only a single
action would still be selected (one of the \textit{drive} options, for instance, or a \textit{cool down}
of a particular duration). Multi-dimensional action spaces are typically associated
with multi-agent scenarios formulated as a single problem or, perhaps, with scenarios
where multiple distinct components of a complex system need to be commanded
simultaneously (e.g., a robotic manipulator with multiple independently commanded
joints). Including health-related actions would not increase the number of dimensions
in most of these cases either, as health-related action classes would be added to already
existing dimensions.

4.4.1 Fully Deterministic Problems

If all sources of uncertainty can be neglected, classical search algorithms, such as
Depth First Search, Breadth First Search, or $A^*$ (Ghallab, Nau, and Traverso, 2016),
can be applied to compute exact plans in problems of a suitable size. The worst-case
computational complexity will be $O(|A_{SHM}^D| + |A_{DM}^D|)$ for the separated formulation
and $O(|A_{SHM} \cup A_{DM}|^D)$ for the unified one, with an overall worst-case computational
CHAPTER 4. HEALTH-AWARE DECISION MAKING

complexity increase factor $O\left(\frac{|A_{SHM} \cup A_{DM}|^D}{|A_{SHM}|^D + |A_{DM}|^D}\right)$. Approximate algorithms, including those with the anytime property, can be used to trade optimality guarantees for performance in larger problems (Boddy, 1991; Burfoot, Pineau, and Dudek, 2006; Hansen and Zhou, 2007). If a problem formulation includes a continuously valued action space, it may need to be sampled; in that case, $|A|$ is the number of samples.

4.4.2 Problems with Outcome Uncertainty

Section 3.4.3 showed that for an actively controlled system operating in the presence of uncertainty, informing SHM mitigation or recovery action selection with prognostic predictions can be ineffective. Therefore, rather than estimating the computational complexity of SHM implemented as the sequence in Figure 3.2, for the purposes of this analysis the assumption is made that both SHM and DM problems are formulated as MDPs. Given the focus of this thesis on complex model spaces, use of MCTS or comparable solver is also assumed. The computational budget of forward simulations used by prognostics to construct a $p$(RUL) distribution can instead be allocated towards building an MCTS policy tree, for instance.

As described in Section 2.6, in MCTS Monte Carlo simulations from an initial state $s_0$ are used to build a policy (search) tree and, ultimately, estimate the expected utility of each action available in $s_0$ (as MCTS is tree-based, continuously valued action spaces would need to be sampled). The error in approximating the optimal policy depends on how many simulations passed through each node of the tree. For simplicity, it is assumed here that the action space size $|A|$ is the same for all states and that each valid action is being invoked exactly $N$ times on every level of the tree, up to a depth $D$. Given these assumptions, $N^D|A_{SHM} \cup A_{DM}|^D$ simulations would be required for a tree of depth $D$ in the unified case versus $N^D(|A_{SHM}|^D + |A_{DM}|^D)$ in the separated case. Thus, the computational complexity increase for the unified case can be estimated as being $O\left(\frac{|A_{SHM} \cup A_{DM}|^D}{|A_{SHM}|^D + |A_{DM}|^D}\right)$. 
4.4.3 Problems with State and Outcome Uncertainty

Analogously to the MDP case, the assumption is made for the case of state and outcome uncertainty that both DM and SHM problems are formulated as POMDPs. POMCP and DESPOT (Chapter 2), being representative examples of modern online POMDP solvers, are used for the analysis. The effects of unified $A$ and $O$ spaces on the accuracy of approximating an optimal policy are discussed first, followed by a discussion of the implications a unified, potentially higher-dimensional $S$ could have on belief approximation and updating.

The error in approximating the optimal policy for POMCP will depend on the number of simulations that passed through each history/belief node of the tree (similarly to MCTS). Unlike MCTS trees, however, levels of history nodes in POMCP trees are interleaved with levels of observation nodes, resulting in $O(|A|^D|O|^D)$ history nodes for a planning horizon of length $D$. For continuously valued action spaces, $|A|$ is the number of sampled actions. An assumption is made, again, that enough simulations are executed to generate $|A|$ branches out of every history node. It is also assumed that $|O|$ is the number of branches under each observation node. For continuously valued multidimensional observation spaces, a strategy to add branches that results in a sufficient representation of the entire observation space may be required. Each simulation executed on a partially constructed tree adds exactly one new history node (Silver and Veness, 2010); therefore $O(|A|^D|O|^D)$ simulations are required to construct the tree.

In a unified formulation, $O(|A_{SHM} \cup A_{DM}|)$ actions may be needed to represent the combined action space. The size of $O_{HADM}$, however, may need to be larger than $|O_{SHM} \cup O_{DM}|$ in order to adequately represent a higher-dimensional combined observation space. Thus, the POMCP computational complexity increase factor for the unified formulation can be estimated as $O((|O_{HADM}|^D|A_{SHM} \cup A_{DM}|^D)/(|O_{SHM}|^D|A_{SHM}|^D + |O_{DM}|^D|A_{DM}|^D))$.

In the case of DESPOT, Ye, Somani, Hsu, and Lee (2017) provide two theoretical results useful for computational complexity analysis. The first result bounds the error of estimating the value of any policy in $\Pi_{b_0,D,K}$ (as defined in Chapter 2, $b_0$ is the
current belief state, $D$ is the maximum depth of the tree, and $K$ is the number of pre-sampled scenarios used in constructing the DESPOT tree). The key implication of this result is that a DESPOT constructed with a small number of scenarios is sufficient for approximate policy evaluation. The second result shows that by optimizing this bound, a policy can be obtained that is competitive with the best small policy (the size $|\pi|$ of a DESPOT policy $\pi$ is the number of belief nodes in its policy tree). Constraining policy size is important to prevent overfitting, as a policy optimized for a finite number of sampled scenarios may not perform well in general.

In proving the first result, Ye, Somani, Hsu, and Lee (2017) show that for any given constants $\tau, \alpha \in (0, 1)$, any belief $b_0$, and any positive integers $D$ (tree depth) and $K$, every DESPOT policy tree $\pi \in \Pi_{b_0,D,K}$ satisfies

$$V_\pi(b_0) \geq \frac{1 - \alpha}{1 + \alpha} \hat{V}_\pi(b_0) - \frac{R_{\text{max}}}{(1 + \alpha)(1 - \gamma)} \cdot \frac{\ln(4/\tau) + |\pi| \ln(KD|A||O|)}{\alpha K}$$

(4.1)

with probability of at least $1 - \tau$, where $\hat{V}_\pi(b_0)$ is the estimated value of $\pi$ under a set of $K$ scenarios randomly sampled according to $b_0$, $R_{\text{max}}$ is the maximum reward, and $\gamma$ is the POMDP discount factor. This implies that all DESPOT policies in $\Pi_{b_0,D,K}$ satisfy the bound given in (4.1) with high probability. The policy estimation error (second term on the RHS) can be made arbitrarily small by choosing an appropriate $K$.

The second result shows that a near-optimal policy $\hat{\pi}$ can be obtained by maximizing the RHS of (4.1). Let $\Pi_D$ be the set of all policies derived from a DESPOT $D$ with depth $D$, constructed with $K$ scenarios sampled randomly according to belief $b_0$. For any arbitrary policy $\pi \in \Pi_D$ and any given constants $\tau, \alpha \in (0, 1)$, if

$$\hat{\pi} = \arg \max_{\pi' \in \Pi_D} \left\{ \frac{1 - \alpha}{1 + \alpha} \hat{V}_{\pi'}(b_0) - \frac{R_{\text{max}}}{(1 + \alpha)(1 - \gamma)} \cdot \frac{|\pi'| \ln(KD|A||O|)}{\alpha K} \right\},$$

(4.2)
then

$$V_{\hat{\pi}}(b_0) \geq \frac{1 - \alpha}{1 + \alpha} V_{\pi}(b_0) - \frac{R_{\text{max}}}{(1 + \alpha)(1 - \gamma)} \times$$

$$\times \left( \ln(8/\tau) + |\pi^*| \ln(KD|A||O|) \right) + (1 - \alpha) \left( \sqrt{2\ln(2/\tau)/K} + \gamma^D \right)$$  \hspace{1cm} (4.3)$$

with probability of at least $1 - \tau$.

In expression (4.3), performance of $\hat{\pi}$ — a policy maximizing (4.2) — is bounded relative to the performance of another policy, $\pi$. Since $\pi$ can be any policy in $\Pi_D$, an optimal policy $\pi^*$ can be chosen for the role. If $|\pi^*|$ is small, the approximation error of $\hat{\pi}$ is also small. If $\pi^*$ is large, but is approximated well by some small policy $\pi$ of size $|\pi|$, then $\hat{\pi}$ can be obtained with a small approximation error by choosing $K$ to be $O(|\pi| \ln(KD|A||O|))$.

Now consider how the RHS of (4.3) may change in a unified HADM formulation. Since an optimal HADM policy can be expected to perform at least as well as either SHM or DM policy, $V_{\hat{\pi}}(b_0)$ — i.e., $V_{\pi^*}(b_0)$ — will increase or remain the same. Rewriting the additive error on the RHS of (4.3) as

$$\epsilon = \frac{R_{\text{max}}}{(1 + \alpha)(1 - \gamma)} \times$$

$$\times \left( \ln(8/\tau) + |\pi^*| \ln(KD|A||O|) \right) + (1 - \alpha) \left( \sqrt{2\ln(2/\tau)/K} + \gamma^D \right)$$  \hspace{1cm} (4.4)$$

and assuming that an approximately the same $\epsilon$ is to be maintained for the unified HADM policy as for the separated SHM/DM policies, the ratio $K_{\text{HADM}}/(K_{\text{SHM}} + K_{\text{DM}})$ can now be estimated. Another assumption made is that $\alpha$, $\gamma$, $\tau$, $D$, and $R_{\text{max}}$ are the same for HADM as for SHM/DM. Consequently, the $R_{\text{max}}/((1 + \alpha)(1 - \gamma))$ multiplier will also remain the same. The second term inside the parentheses is small for realistic values of $K$ (i.e., hundreds). The numerator of the first term is dominated by $|\pi^*|$ (the size of an optimal policy). The ratio $K_{\text{HADM}}/(K_{\text{SHM}} + K_{\text{DM}})$ can, therefore, be estimated as being $O(|\pi^*_{\text{HADM}}|/(|\pi^*_{\text{SHM}}| + |\pi^*_{\text{DM}}|))$. Just as for POMCP, the scenarios would have to be executed for a potentially larger
number of actions on every level of the tree, thus the overall increase in DESPOT computational complexity due to unified $O$ and $A$ spaces will be $O(|\pi^*_\text{HADM}| \cdot A^\text{SHM} \cup A^\text{DM}^D / (|\pi^*_\text{SHM}| \cdot A^\text{SHM}^D + |\pi^*_\text{DM}| \cdot A^\text{DM}^D))$. An interesting implication of this result is that if a more compact optimal policy exists for HADM than for the separated SHM and DM (which may well be the case), the overall DESPOT computational complexity may actually decrease (at least the complexity attributable to the effects of unified $A$ and $O$ spaces).

In some partially observable problems, the greatest computational challenge may arise from the effects of a higher-dimensional state space. Both POMCP and DESPOT use particle-based belief representations and are typically coupled with a particle filtering algorithm (Gordon, Salmond, and Smith, 1993) to perform belief updating. Snyder, Bengtsson, Bickel, and Anderson (2008) and Bengtsson, Bickel, and Li (2008) find that particle filters that rely on an insufficiently large set of particles in high-dimensional spaces tend to collapse, with a single particle ending up with a weight close to unity. In order to maintain representational quality and prevent filter collapse, the number of particles in the set must grow exponentially with the number of space dimensions—or, more precisely, with the variance of the observation log likelihood, as Snyder, Bengtsson, Bickel, and Anderson (2008) demonstrate. For use with an online POMDP solver, such as POMCP or DESPOT, particles at the root of a partial belief tree form both the current belief approximation and the start states of simulations/scenarios used to construct the tree. The number of POMCP simulations or DESPOT scenarios needed will then be the greater of the two: the number required to maintain the quality of the optimal policy approximation or the number required for an effective state space coverage.

Several approaches have been proposed for handling high-dimensional state spaces in belief representation and updating. Roy, Gordon, and Thrun (2005) observe that in real-world POMDP problems, computing an optimal policy for the entire belief space is often unnecessary and that the beliefs relevant for decision making often lie near a structured, low-dimensional subspace embedded in the high-dimensional belief space. They, therefore, propose reducing the dimensionality of the belief space (prior to policy computation) by using the exponential family principal components.
analysis. Bengtsson, Snyder, and Nychka (2003) describe a nonlinear ensemble filter that can handle non-Gaussian probability densities as an alternative to particle filters for high-dimensional systems. Importance sampling has also been suggested as a way to design particle filters that are scalable to high-dimensional problems (Daum and Huang, 2003). Luo, Bai, Hsu, and Lee (2019) implement a version of the DESPOT solver that uses importance sampling in scenario selection, which, in particular, can be helpful for ensuring that the computed policy addresses rare events that otherwise would be difficult to sample. For HADM, this could mean the ability to capture policy responses to fault modes, especially those for which degradation models are not part of the overall state transition model \( T(s, a, s') \). With all importance sampling approaches, constructing a good proposal distribution is key. In most applications this has been done manually, relying on domain knowledge. Luo, Bai, Hsu, and Lee (2019) suggest a method for automatically constructing proposal distributions, specifically geared for POMDP policy computation.

Rebeschini and Van Handel (2015) provide a mathematical foundation for another approach to dealing with high-dimensional state spaces. The central idea of the approach is that the decay of correlations property, which could be viewed as a spatial counterpart of the stability property of nonlinear filters, can be exploited to develop local particle filters that scale well to high-dimensional settings. The authors also present an example algorithm, Block Particle Filtering, for which they prove an approximation error bound that is uniform in both time and dimension. Beskos, Crisan, Jasra, Kamatani, and Zhou (2017) extend a related idea into the Space-Time Particle Filter that combines local filtering along the space dimensions with global filtering in the time dimension to provide sub-exponential computational complexity in the number of space dimensions. The authors provide theoretical and numerical results showing consistency of the filter and its stability in high dimensions for certain model classes.
4.4.4 Practical Implications

The analysis presented above is not exhaustive across all possible types of DM methods. Its objective, however, is to illustrate that suitable methods for implementing unified HADM exist and while there may be increases in computational complexity relative to separated implementations, such increases are not insurmountable even in the most complex cases.

Fully deterministic HADM problems and problems with action outcome uncertainty can be solved optimally or near-optimally with a computational complexity factor increase of, at worst, $O(|A_{SHM} \cup A_{DM}|^D/(|A_{SHM}|^D + |A_{DM}|^D))$ over a separated formulation (here $D$ is equivalent to $H$, i.e., the planning horizon). While not negligible, such an increase should be manageable for the application domain being considered since $|A_{SHM}|$ is typically much smaller than $|A_{DM}|$, with most of the mitigation/recovery control accomplished through $|A_{DM}|$ (for many systems $A_{SHM} = \emptyset$, with $A_{DM}$ used both for controlling the system and managing its health). Figure 4.2 shows how the computational complexity may increase for different decision horizon lengths if SHM-specific actions ($A_{SHM}$) constitute 5% and 10% of the unified action space $A_{HADM}$, respectively. One way to potentially alleviate an increase in the action space size is by using state-dependent action spaces, thus reducing the average action branching factor.

![Figure 4.2: Examples of computational complexity increase for different decision horizons in fully deterministic problems and problems with action outcome uncertainty](image)

Partially observable HADM problems could present a more challenging case from the computational complexity point of view. In addition to the complexity increase
resulting from a larger action space, increases due to higher-dimensional state and observation spaces may play a role. It is anticipated, however, that dimensionality increases for unified $S$ and $O$ will be moderated by elimination of redundant dimensions. Additionally, algorithms like DESPOT exist for this class of problems, with computational complexity depending primarily on the problem structure (size of the optimal policy) rather than on the dimensionality of $S$ and $O$. It is quite possible that some HADM problems will have more compact optimal policies than their separated formulations by the virtue of having access to the unified model spaces.

Finally, many modern DM algorithms lend themselves well to parallelization. For instance, HyP-DESPOT (Cai, Luo, Hsu, and Lee, 2018) leverages massive parallelization on both CPUs and GPUs to demonstrate a speed-up factor of a few hundred times relative to the original DESPOT algorithm on several benchmark problems. It is reasonable to expect the emerging high-dimensional belief update methods referenced in the previous subsection to benefit from parallelization as well.

### 4.5 Discussion

Following an examination of issues with the prevailing SHM approach in Chapter 3, this chapter presented the rationale for comprehensively unifying—not just integrating—system health management with decision making in order to overcome these issues. An approach for accomplishing such a unification is then described, including showing how traditional system health management concepts map into it. On the other hand, it is proposed that the emergency response functionality be kept separate from unified DM/SHM in order to guarantee timely reactions, provide dissimilar redundancy, and allow for offline computation, validation, and verification of emergency response policies.

With the unified approach being suitable for systems of various complexity and types of uncertainties present, a wide range of existing and emerging computational methods can be used to implement it. While there may be an overall increase in computational cost under the proposed approach (as compared to a separated formulation, summarized in Table 4.1), advances in algorithms and computing hardware
generally make that a surmountable challenge.

Table 4.1: HADM computational complexity increase summary

<table>
<thead>
<tr>
<th>Problem type</th>
<th>Complexity increase versus a separated DM/SHM formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully deterministic</td>
<td>$\mathcal{O}((</td>
</tr>
<tr>
<td>Outcome uncertainty</td>
<td>$\mathcal{O}((</td>
</tr>
<tr>
<td>State and outcome uncertainty</td>
<td><strong>Attributable to state space dimensionality increase</strong></td>
</tr>
<tr>
<td></td>
<td>Exponential with the variance of the observation log likelihood (Snyder, Bengtsson, Bickel, and Anderson, 2008)</td>
</tr>
<tr>
<td></td>
<td><strong>Attributable to action and observation spaces size/dimensionality increases</strong></td>
</tr>
<tr>
<td>POMCP:</td>
<td>$\mathcal{O}(</td>
</tr>
<tr>
<td>DESPOT:</td>
<td>$\mathcal{O}(</td>
</tr>
<tr>
<td></td>
<td>(may result in a complexity decrease if a more compact $\pi^*_{DM}$ exists)</td>
</tr>
</tbody>
</table>

In return, the approach is expected to improve a system’s resilience to faults and its capacity to accomplish operational objectives, while potentially simplifying its informational architecture, reducing the overall sensor suite size, and combining modeling efforts. It also enables methodical exploration of design choices that affect both decision making and system health during operation. With the latter, the proposed approach takes a step towards the vision long held by the model-based AI community: to leverage the same set of models for different functions throughout a system’s lifetime.
Chapter 5

Algorithms for Complex POMDPs

Solving real world decision making problems modeled as POMDPs is a fundamentally challenging computational task. As the analysis in the previous chapter shows, unifying the two currently separated tasks — DM and SHM — brings with it several important benefits. However, these benefits come at an additional computational cost due in part to the increase in size and, possibly, dimensionality of action and observation spaces. This chapter reviews the state of the art in handling complex action and observation spaces and presents two algorithms, Adaptive Action Selection and Transition Correlation Clustering, designed to overcome some of the limitations of currently available methods. These algorithms are combined with an existing online POMDP solver, DESPOT, to form a new solver, LPDM. Performance of LPDM is then evaluated on an abstract POMDP with complex model spaces.

5.1 Introduction

In the early years of POMDP development, it was only feasible to solve POMDPs with relatively small discrete state, action, and observation spaces (i.e., with typical space sizes smaller than ten) using exact offline solution methods such as value iteration (Chapter 2). Few useful real-world problems could be modeled effectively given this limitation. The main thrust of research in the following years has been directed
towards improving the ability to handle large state spaces. This was mainly accomplished by sampling $S$ or $B$ and focusing on the subsets of $S$ or $B$ that are reachable from the current belief (i.e., deriving online policies), although offline sampling algorithms also exist (e.g., Pineau, Gordon, and Thrun, 2006). A significant improvement in the performance of sampling online POMDP solvers was achieved when algorithms patterned on Monte Carlo Tree Search, such as POMCP and DESPOT, were developed (Chapter 2).

While some progress in handling complex $A$ and $O$ spaces also took place (reviewed in the next section), it has been more limited, and so in most applications of MCTS-style POMDP solvers, action and observation spaces are discrete and similar in size to those in the more traditional offline formulations.

The fundamental reason for the limits on $|A|$ is that in building a policy tree, a MCTS-style solver creates child action branches for every belief (i.e., history) node. Typically, a branch is created for every action in $A$. Even if subsampled, as some of the methods described in Section 5.2 do, the total number of branches used (the branching factor) should still be limited. A large branching factor $\beta_a$ would result in a wide tree, with the number of nodes — and, therefore $|\pi|$ — growing exponentially with $\beta_a$. In order to maintain the same fidelity in estimating the utility function $\hat{U}_\pi$, the average number of simulations going through a node at some depth $d$ should remain approximately the same. Therefore, the total number of simulations executed should go up exponentially with $\beta_a$ as well.

MCTS-style solvers also branch on observations, creating branches out of every action node, and the same limitations apply to the observation branching factor $\beta_o$. Even though both POMCP and DESPOT, for example, essentially subsample observation branches (only including those in the tree that appear in a simulated scenario), enough scenarios still need to be executed to represent the relevant regions of $S$ and $O$ with sufficient accuracy. Yet increasing $\beta_o$ again means increasing the number of tree-building simulations exponentially, if maintaining the same fidelity of $\hat{U}_\pi$ is desired.

There is also an additional issue that is related to the branching factor but is unique to observation spaces. It has to do with handling observations that may
appear dissimilar, yet are indicative of the same or closely related system states. The following simple example illustrates the issue.

**Example 5.1.** Assume that some belief $b$ is represented by a set of three particles, i.e., $b = \{s_1, s_2, s_3\}$. Executing action $a$ in $b$ results in three transitions with three distinct observations generated: $\langle s_1, o_1, s'_1 \rangle$, $\langle s_2, o_2, s'_2 \rangle$, $\langle s_3, o_3, s'_3 \rangle$. POMCP or DESPOT would create three separate observation branches, $o_1$, $o_2$, and $o_3$, each leading to a new one-particle belief: $b'_1 = \{s'_1\}$, $b'_2 = \{s'_2\}$, and $b'_3 = \{s'_3\}$.

In practice, however, dissimilar observations may in fact be pointing to the same underlying system state. This is something that is often used in system health management to confirm or narrow down a fault diagnosis: data from two or more dissimilar sensors is analyzed to see if they correlate with the same fault signature. For instance, consider an abstract case of a launch vehicle in the process of being fueled, with data from three dissimilar sensors being monitored: an altimeter ($o_1$), a pressure sensor inside the fuel tank ($o_2$), and a fuel tank wall strain gauge ($o_3$). Only one of the observations can be generated as a result of a simulated transition.

If the tree-building scenarios being simulated with action $a$ result in the tank being overloaded with fuel, both $o_2$ and $o_3$ should have values outside their nominal ranges, while the value of $o_1$ will remain the same as before. If it can be recognized that $o_2$ and $o_3$ are related, $s_2$ and $s_3$ can be placed in the same new belief $b'_2 = \{s'_2, s'_3\}$, potentially resulting in a better approximation of an important system state (one where the fuel tank is over-pressurized). Subsequent simulations going through $b$ and involving $a$ would only be split between two observation branches, instead of three, possibly leading to better $U(b'_1)$ and $U(b'_2)$ estimates as well.

The issue highlighted in Example 5.1 and the other challenges discussed above are referenced again in the next section, which overviews related work. Section 5.3 then proposes a new methodology for handling complex action spaces and presents an algorithm, Adaptive Action Selection, based on this methodology. Section 5.4
CHAPTER 5. ALGORITHMS FOR COMPLEX POMDPS

76

describes Transition Correlation Clustering, a novel algorithm for handling complex observation spaces. LPDM, a POMDP solver that incorporates Adaptive Action Selection and Transition Correlation Clustering, is covered in Section 5.5. Section 5.6 provides information on the experiments conducted to validate performance of LPDM. The chapter is concluded in Section 5.7 with a discussion of the benefits the new algorithms provide and a summary of experimental results.

5.2 Related Work

Several approaches for handling complex action and observation spaces have been investigated by others. This section provides an overview of these approaches, including their limitations, and highlights some representative examples of prior efforts.

5.2.1 Actions

Of the two model space types under discussion, the problem of handling complex action spaces has been explored more extensively than the same problem for observation spaces. That is, to a large extent, because the need for intelligently selecting actions out of a large number of options in an uncertain setting precedes the emergence of POMDPs or even MDPs, perhaps going back to the *multi-armed bandit* problems, formulated in the 1930s (Slivkins, 2019; Thompson, 1933).

Discretization

Discretization is the simplest and still most common way of dealing with complex action spaces. With discretization, a large or continuously valued $A$ is subdivided into regions (bins), most often evenly sized, with one action picked to represent each bin. The main issue with discretization is that if there are too few bins, action selection will be imprecise. On the other hand, if precision is increased, the number of action options may become too large for existing MDP and POMDP solvers to manage within reasonable time and memory constraints. For instance, discretizing possible rover drive headings into just *North*, *South*, *East*, *West* bins is insufficiently precise.
On the other hand, having bins with a 1-degree width would result in $|A| = 360$, a very large action space for a typical modern MDP/POMDP solver. Finding the right balance in practice is not always easy. The issue becomes even more pronounced if $A$ is multidimensional. If rover speed has to be chosen along with a drive heading and the speed is discretized into 10 bins, with 360 heading options $|A|$ becomes 3600.

**Hierarchical Decomposition**

The basic idea behind hierarchical decomposition of action spaces is to organize the overall decision making process into smaller (presumably more manageable) subtasks, structured in a hierarchical manner. Some of these subtasks may require deriving a simpler general policy, while others may be accomplished by using finite-state controllers.

Many examples of hierarchical decomposition of POMDP action spaces exist, notably in the work by Pineau, Gordon, and Thrun (2003), Theocharous, Murphy, and Kaelbling (2004), Toussaint, Charlin, and Poupart (2008), and Lim, Hsu, and Lee (2011). For the VIPER mission in SHERPA (Chapter 6), in addition to the action-handling algorithm described in the next section, there is also an implicit assumption of hierarchical action decomposition. SHERPA is only reasoning about strategic, higher-level actions, assuming that mid-level actions (such as locomotion and obstacle avoidance) and low-level control will be accomplished by other systems. Yet this assumption still does not eliminate the necessity to handle large action spaces. Actions with strategic implications, such as driving in a particular direction or scheduling a recharge period of a certain duration, still need to have at least a large discrete domain to provide the necessary precision. In the VIPER Traverse Synthesis use case, multiple categories of such actions are considered simultaneously.

While in the VIPER example the high-, mid-, low-level hierarchy of decision-making tasks is fairly easy to discern, that may not always be the case for other applications. There has been some work on automatic discovery of action hierarchies, such as the work by Charlin, Poupart, and Shioda (2007) where hierarchy discovery is formulated as a non-convex optimization problem. An optimization of this type is often a formidable computational task and may not scale well in real-world applications.
Showing that a decomposed policy is optimal may also prove challenging.

**Double Progressive Widening**

Among the methods for supporting continuously valued action spaces with MCTS-style solvers is the *progressive widening* technique (Rolet, Sebag, and Teytaud, 2009). Progressive widening was improved upon by Couëtoux (2013), solving the problems of convergence to the optimal action by introducing Double Progressive Widening (DPW). DPW remains one of the more popular methods of dealing with continuously valued MDP action spaces. It has also been incorporated into POMCP for partially observable problems (Sunberg, 2018). When a policy tree is being built, for any state or belief node, DPW attempts to balance exploitation of actions already branching out of the node and exploration of new actions. A new randomly sampled action is added when a criterion based on the number of visits to the node and its current actions branches is reached.

DPW does, however, suffer from the limitation in its ability to explore large action spaces, as there is no guarantee that actions added one-by-one using the above method will provide a sufficient coverage of $A$.

**Blind Value Method**

In an effort to overcome the inability of DPW to explore large action spaces, Couëtoux (2013) introduced another method for handling complex action spaces — Blind Value (BV). This method uses a distance metric over the action space and is intended to first explore actions more distant from the already explored actions, then focus on the areas with a high concentration of high-value actions (as measured by UCB). This heuristic is implemented as the *BlindValue* procedure in Algorithm 5.1. During every selection iteration (*SelectAction* procedure of Algorithm 5.1), actions are picked and evaluated out of a randomly sampled pool of $M$ actions.

While BV is an improvement over DPW in terms of exploration capability, exploration is done in a somewhat unstructured manner and may not be ideal for identifying globally optimal actions. BV is, however, the most comparable approach found in the
**Algorithm 5.1** Blind Value algorithm

1: **procedure** SELECTACTION($A$, $A_{\text{exp}}$, $M$, $distance$)  
   \(\triangleright A_{\text{exp}}\) is the already explored subset of $A$,  
   \(\triangleright M\) is the size of the selection pool,  
   \(\triangleright distance\) is an actions distance metric, $distance : A \times A$.  
2: $A_{\text{pool}} \leftarrow$ uniformly sample $M$ times from $(A \setminus A_{\text{exp}})$  
3: $\sigma_{\text{known}} \leftarrow \text{STANDARDDEVIATION}_{a \in A_{\text{exp}}}(\text{UCB}(a))$  
4: $\sigma_{\text{pool}} \leftarrow \text{STANDARDDEVIATION}_{a \in A_{\text{pool}}}(\text{distance}(a,a_c))$  
   \(\triangleright a_c\) is the centroid of $A$  
5: $\rho \leftarrow \sigma_{\text{known}} / \sigma_{\text{pool}}$  
6: $a^* \leftarrow \arg\max_{a \in A_{\text{pool}}} \text{BLINDVALUE}(a, \rho, A_{\text{exp}})$  
7: **return** $a^*$  
8: **end procedure**

9:

10: **procedure** BLINDVALUE($a_{\text{test}}$, $\rho$, $A_{\text{exp}}$)  
   \(\triangleright a_{\text{test}}\) is an unexplored action  
11: $bv \leftarrow \min_{a \in A_{\text{exp}}} \rho \cdot \text{distance}(a_{\text{test}}, a) + \text{UCB}(a)$  
12: **return** $bv$  
13: **end procedure**

literature to the one developed over the course of this work (Section 5.3). Therefore it is one of the action branch expansion options implemented for DESPOT and is used in the experiments described in Section 5.6.

**Reinforcement Learning**

The final approach reviewed is to explore complex action spaces and develop policies for acting under uncertainty through reinforcement learning methods. Ross, Chaib-draa, and Pineau (2008), for example, use Bayesian reinforcement learning for POMDP model parameters to perform robot navigation with continuously valued actions. Dallaire, Besse, Ross, and Chaib-draa (2009) adopt a similar approach, but make use of Gaussian Process Dynamic Models. Lillicrap et al. (2016) developed a model-free algorithm using deep function approximators that can learn policies in high-dimensional, continuously valued action spaces.
The main limitation of reinforcement learning methods is that they require availability of one or more of the following to be effective: a comprehensive training data set, a suitable set of basis functions (in the case of Gaussian process models), or sufficiently detailed interaction with the environment. None of these requirements may be satisfiable for an application such as the VIPER mission, where a realistic training data set could be difficult to develop, a set of suitable basis functions could be challenging to identify due to the complexity of the system, and the price of an error while interactively learning from the environment may be unacceptably high.

5.2.2 Observations

With a growing interest in the recent years in modeling systems with richer sensory capabilities using POMDPs (see, e.g., Sunberg, 2018), the problem of handling complex observation spaces has started to get more attention. Perhaps not surprisingly, however, discretization still remains the most common method of dealing with observation spaces.

Discretization

As for actions spaces, discretization of observation spaces is subject to the curse of dimensionality. Discretizing even a single-dimensional continuously valued observation may require a large number of discrete “bins” in order to be useful. This is particularly true in the SHM context, where observation values need to have a fine enough granularity to detect subtle differences in system behavior and provide an early warning of developing faults. With multi-dimensional observation vectors, the size of a reasonably detailed discrete observation space may quickly become intractable. Consider a simple system monitored by just three sensors, with the output range of each sensor discretized into ten bins. This will result in $|O| = 10^3$—well beyond the capabilities of any standard tree-based POMDP solver to handle in an effective manner. Discretization would also not help with recognizing dissimilar, but related observations, such as those in Example 5.1.
Clustering, Compression, and Dimensionality Reduction

In the context of POMDPs, observations are used to identify or dynamically form belief states. There has been some limited work to directly cluster, compress, or reduce dimensionality of observation spaces based on values (e.g., Carlin and Zilberstein, 2008). There has also been work on applying clustering, compression, and dimensionality reduction techniques in belief spaces. Roy, Gordon, and Thrun (2005) used exponential family Principal Components Analysis to describe sparse, high-dimensional belief spaces using small sets of learned belief state features. Only the low-dimensional features are then used in planning. Li, Cheung, and Liu (2005a, 2005b) cluster beliefs using what they refer to as spatial and temporal distance metrics. The former refers to the distance within the belief space, the latter is a measure of evolution of a belief state over a number of time steps.

Clustering in belief spaces as a way of identifying important, distinct system modes (nominal or off-nominal) was also the approach initially pursued in this work. However, the challenges of incorporating domain-specific context when comparing or compressing belief states eventually became apparent (and similar challenges also apply to observation spaces, e.g., as illustrated in Example 5.1). Consider a simple rover scenario, where the state vector \( s \) contains three elements only: \( s = (x, y, T_b) \), where \( x \) and \( y \) are rover coordinates in meters relative to its starting point and \( T_b \) is the temperature of its battery in degrees Celsius. States \( s_1 = (20, 30, 35) \) and \( s_2 = (40, 60, 30) \) are similar from the decision-making point of view, as illumination and communication coverage conditions at the two locations are identical, the physical distance between them is not significant enough to make a difference in action selection, and the battery temperature is within the nominal bounds (between 10 and 40). However, \( s_1 = (20, 30, 35) \) and \( s_2 = (20, 30, 45) \) are quite different, as, despite the same location, the battery temperature in \( s_2 \) is outside the nominal bounds—indicating a possible fault. A sophisticated system model may be required for this type of an analysis in a real-world application. If deciding how to best compare or define a distance metric between two states is challenging enough, deciding how to effectively do the same for beliefs (e.g., probability distributions over the state space) adds yet another level of complexity.
Double Progressive Widening

Sunberg (2018) and Sunberg and Kochenderfer (2018) explore the use of modified DPW for handling continuously valued observation spaces, particularly in conjunction with POMCP. Sunberg and Kochenderfer recognize that applying DPW in its original form with a solver such as POMCP (which uses particle-based belief) would result in a policy tree where all belief nodes contain exactly one particle. That is because when an $s'$ and an $o$ are generated during a tree-building simulation, $s'$ can only be added to an existing child belief node if $o$ matches some prior observation exactly. With a continuously valued $O$, the probability of this happening is zero. Instead, Sunberg and Kochenderfer implement tree-building belief updates using weighted particles. In their approach, sets of particles generated as a result of simulating an action are added to all child belief nodes. The authors demonstrate that Partially Observable Monte Carlo Planning with Observation Widening (POMCPOW), as the new solver is called, performs better on several continuous POMDP problems than POMCP augmented with standard DPW, as well as several other solvers (including the standard version of DESPOT).

Exploratory limitations of DPW remain even in the modified procedure, however, as the procedure is agnostic to the value of an observation when deciding whether to add a new tree branch for it or not. It may, for instance, end up adding only closely-related observation branches without providing an adequate representation of $O$ and therefore exploring only a subset of relevant execution scenarios.

5.3 Algorithm for Complex Action Spaces

In discussing the proposed approach to handling complex action spaces it may be useful to take a step back and reexamine the elementary reasons for why exploring a variety of action options is necessary when developing a POMDP (or an MDP) policy, even if these reasons may appear obvious. For any $s \in S$ (and, similarly, $b \in B$) only one action matters: $\pi^*(s) = \max_{a \in A} Q^*(s, a)$. The optimal utility function $Q^*(s, a)$ is generally assumed to be defined $\forall s \in S, a \in A$, but otherwise may not be known,
especially for a complex, real-world system. The process of deriving an optimal policy is, in many cases, the process of approximating the optimal utility function (unless, for example, direct policy search is used). For an offline policy, the entire $Q^*$ may need to be approximated, while for an online policy approximating specific parts or points of $Q^*$ may suffice. For a typical tree-based MDP or POMDP solver (e.g., MCTS, POMCP, or DESPOT), point-wise estimates of $Q^*(s_n, a)$ — where $s_n$ is the current state — are done for a predefined, discrete set of points, i.e., actions in $A$. After the estimates are computed, $\pi^*(s_n) \in A$ is chosen, which is likely not the true global maximum of $Q^*(s_n, a)$ for continuous values of $a$.

Now, consider instead a problem formulation where, for a complex $A$, points (actions) used to find $\max_{a \in A} Q^*(s_n, a)$ could be chosen dynamically and sequentially. Consider also that the set of points sampled may need to be limited (e.g., the process of sampling/estimation may be computationally expensive or the action branching factor needs to be relatively small). This problem formulation would essentially be the same as for the general problem of expensive global black-box optimization (Jones, Schonlau, and Welch, 1998). What then follows is that the methods developed for expensive global black-box optimization should be applicable to choosing which actions to explore in searching for $\max_{a \in A} Q^*(s_n, a)$.

The choice of an actual method is less important and will likely depend on the application, as is usually the case with global black-box optimization problems (Muñoz, Sun, Kirley, and Halgamuge, 2015). The action selection algorithm created in this work, Adaptive Action Selection (AAS), is loosely patterned on Fast Simulated Annealing, FSA (Szu and Hartley, 1987). Unlike classical Simulated Annealing (Kirkpatrick, Gelatt, and Vecchi, 1983), which is strictly a local search, FSA allows occasional long jumps. The basic AAS algorithm is described next for a general tree-based POMDP solver that builds a policy tree with interleaved levels of belief and action nodes by executing consecutive simulations (Algorithm 5.2). Section 5.5 then shows how AAS is integrated into the LPDM solver.

AAS is invoked every time a simulation passes through a belief node $b$ and the decision to add an action branch is made (branches are not expected to be added on every visit to $b$). As in Blind Value, a distance metric defined over $A$ (system
action space) is required. As the metric is domain-dependent, the version used in the LPDM experiments is defined in Section 5.6 and the metrics used in SHERPA are provided in Chapter 6. When a new $b$ is added to the tree, $A_b$ can be seeded with pre-selected actions to speed up convergence or left empty. In the latter case $A_b$ will be initialized with a uniformly sampled $a \in A$. A “temperature” parameter $\tau$ governs the radius from the current best action $\hat{a}_b^*$ within which the new action is uniformly sampled. There are two “temperature” components used, each with the $[0, 1]$ range: $\tau_{\text{solver}}$ and $\tau_{\text{node}}$, with $\tau \triangleq \min\{\tau_{\text{solver}}, \tau_{\text{node}}\}$. The first component, $\tau_{\text{solver}}$, is computed based on solver search termination criteria, such as the remaining decision time, the remaining number of trials, or the current gap between upper and lower utility bounds at the root node (as may be the case for DESPOT). The second component, $\tau_{\text{node}}$, is determined by the $|A_b|/|A|_{\text{max}}$ ratio, where $|A|_{\text{max}}$ is the maximum action branching factor. When $\tau$ is high, an action further from the current $\hat{a}_b^*$ may be sampled. As $\tau$ becomes lower, actions closer to the current $\hat{a}_b^*$ are selected.

**Algorithm 5.2 AAS algorithm**

1. **procedure** ADAPTIVEACTIONSELECTION($b$, $A_b$, $\hat{a}_b^*$, $\tau_{\text{solver}}$)
   
   $\triangleright b$ is the current belief node
   $\triangleright A_b$ is the current action space at node $b$
   $\triangleright \hat{a}_b^*$ is the current best action at node $b$
   $\triangleright \tau_{\text{solver}}$ is the current solver “temperature”

2. $\tau_{\text{node}} \leftarrow |A_b|/|A|_{\text{max}}$
3. $\tau \leftarrow \min\{\tau_{\text{solver}}, \tau_{\text{node}}\}$
4. **if** $|A_b| < |A|_{\text{max}}$ **then**
   
   **return** SAMPLEACTION($b$, $A_b$, $\hat{a}_b^*$, $\tau$)
   
   $\triangleright$ SAMPLEACTION is problem-specific

5. **else**
6. **return** $\emptyset$
7. **end if**
8. **end procedure**

Variations on baseline AAS are possible, such as adding more than one action to $A_b$ at a time or varying $|A|_{\text{max}}$ with node depth. They will be explored in future work.
5.4 Algorithm for Complex Observation Spaces

To understand the intuition behind the proposed approach to handling complex observation spaces, it may be helpful to revisit the purpose of having observation branches in a POMDP policy tree in the first place. Unlike actions, where the value of an action \( a \) plays a direct role in generating a next state \( s' \), the actual value of an observation \( o \) provides only perhaps weak evidence of what the actual state of the system may be. In a policy tree with particle-based beliefs, \( o \) is used in assigning \( s' \) to some belief \( b' \); however, all subsequent decisions will be made on the basis of values of states constituting \( b' \). Therefore, in this work the focus is not on the numerical value of an individual observation \( o \), but rather on how it correlates with the state transition that generated it, as well as with the other transitions generated from the same \( b \) as a result of action \( a \).

This section presents an algorithm, Transition Correlation Clustering (TCC), that enables the use of continuous (or large discrete), possibly multi-dimensional observation spaces to form belief nodes in an online POMDP policy tree. TCC, sketched in Algorithm 5.3, is combined with DESPOT in Section 5.5, but should be usable with other online, tree-based POMDP solvers (e.g., POMCP).

It should be noted that in addition to a generative model \( G(s,a) \), as is common for tree-based POMDP solvers, TCC requires an observation model \( Z = p(o|s,a,s') \). It is, however, a model that would already be required by the frequently used belief update algorithms based on weighted particle filtering (as Sunberg and Kochenderfer, 2018, point out as well). It is also a model that may be easier to develop in practice than either an observation distance model (as required for observation space clustering methods) or a model for clustering in the belief space.

TCC (Algorithm 5.3) operates as follows. At a belief node \( b \), state transitions are generated for some action branch \( a \). A generative model \( G \) is applied to all state particles in \( b \), producing a next state, observation, and reward for each particle (line 4). The basic element used in the clustering process is a tuple \( \langle s,o,s' \rangle \), referred to henceforth as a transition. Each generated transition is passed to the AssignTransition procedure for assignment to a cluster. At the end of the process, each cluster of
Algorithm 5.3 TCC algorithm

1: procedure TransitionCorrelationClustering($\mathcal{T}$, $b$, $a$, $\text{maxClusters}$)
2:   $C = \emptyset$ >> the set of all observation clusters for $a$
3:   for all $s \in b$ do
4:     $\langle s', o, r \rangle \leftarrow G(s, a)$
5:     $C \leftarrow \text{AssignTransition}(C, \langle s, o, s' \rangle, a, \text{maxClusters})$
6:   end for
7:   for all $c \in C$ do
8:     $b' \leftarrow \text{CreateBelief}(c)$
9:     $\mathcal{T} \leftarrow \text{AddChild}(\mathcal{T}, b, a, c, b')$
   >> ADDCHILD creates an observation branch for cluster $c$
   >> and terminates it with $b'$
10:  end for
11: return $\mathcal{T}$
12: end procedure

transitions is turned into a new belief state. AssignTransition (Algorithm 5.4) iterates through the current set of clusters and first computes components of the correlation score $z$ for the input transition $\langle s, o, s' \rangle$ (the score is used later in the procedure for determining the best cluster assignment for $\langle s, o, s' \rangle$). The first score component, $z_{t\rightarrow c}$, measures how well observation $o$ from transition $\langle s, o, s' \rangle$ fits within the context of transitions already in a cluster $c$ (line 11). It is the sum of probabilities $Z(o|\bar{s}_i, a, \bar{s}_i')$, where $\bar{s}_i$ and $\bar{s}_i'$ are taken from a transition $\langle \bar{s}_i, \bar{o}_i, \bar{s}_i' \rangle$ in cluster $c$, $i \in \{1, 2, \ldots, |c|\}$. Conversely, $z_{c\rightarrow t}$ quantifies how well observations currently in cluster $c$ fit within the context of $s$ and $s'$ from the input transition $\langle s, o, s' \rangle$ (line 12). The two components are then normalized on lines 14 and 15 and combined into a single correlation score $z$ on line 16. If there is a good match for $\langle s, o, s' \rangle$ in one of the existing clusters, the transition is added to it. Otherwise (and if the maximum number of clusters has not yet been reached), the transition is placed in a new cluster. After the clustering process is complete and a set $C$ of observation clusters is formed, two more steps remain: (1) observation branches are formed for each cluster $c \in C$ and (2) each cluster $c \in C$ is passed to the CreateBelief procedure (Algorithm 5.5) to form a new
Algorithm 5.4 TCC cluster assignment procedure

1: procedure AssignTransition($C, \langle s, o, s' \rangle, a, maxClusters$)
2:     if $C = \emptyset$ then
3:         $C \leftarrow \{\langle s, o, s' \rangle\}$
4:     else
5:         $z^* \leftarrow -\infty$
6:         $c^* \leftarrow \text{null}$
7:         for all $c \in C$ do
8:             $z_{t \rightarrow c} \leftarrow 0$
9:             $z_{c \rightarrow t} \leftarrow 0$
10:            for all $\langle \hat{s}, \hat{o}, \hat{s}' \rangle \in c$ do
11:                $z_{t \rightarrow c} \leftarrow z_{t \rightarrow c} + Z(o|\hat{s}, a, \hat{s}')$
12:                $z_{c \rightarrow t} \leftarrow z_{c \rightarrow t} + Z(\hat{o}|s, a, s')$
13:            end for
14:             $\bar{z}_{t \rightarrow c} \leftarrow z_{t \rightarrow c} / |c|$
15:             $\bar{z}_{c \rightarrow t} \leftarrow z_{c \rightarrow t} / |c|$
16:             \texttt{▷ compare correlation of a transition with cluster $c$ to its own likelihood:}
17:             $z = \frac{1}{2}(\bar{z}_{t \rightarrow c} + \bar{z}_{c \rightarrow t}) / Z(o|s, a, s')$
18:             if $z > z^*$ then
19:                 $z^* \leftarrow z$
20:                 $c^* \leftarrow c$
21:             end if
22:         end for
23:     if $\text{score}^* \geq 1.0 \text{ or } |C| \geq maxClusters$ then
24:         \texttt{▷ assign to the best cluster found}
25:         $c^* \leftarrow c^* \cup \langle s, o, s' \rangle$
26:     else
27:         \texttt{▷ create a new cluster and add to the set of clusters $C$}
28:         $C \leftarrow C \cup \{\langle s, o, s' \rangle\}$
29:     end if
30: end procedure
Algorithm 5.5 TCC belief creation procedure

1: procedure CreateBelief(c)
2:   \( b \leftarrow \emptyset \)
3: for all \( \langle \tilde{s}, \tilde{o}, \tilde{s}' \rangle \in c \) do
4:   \( b \leftarrow b \cup \tilde{s}' \)
5: end for
6: return \( b \)
7: end procedure

belief \( b' \) that terminates the observation branch corresponding to \( c \) (lines 8 and 9 of Algorithm 5.3).

The TCC algorithm lends itself well to vectorized implementations, allowing, for example, to efficiently pre-compute \( Z(o|s, a, s') \) for all permutations of states and observations generated for action \( a \) out of belief state \( b \). The pre-computed values can then be used to speed up the clustering process. A deliberate choice was made in the design of this algorithm to favor simplicity and speed over clustering optimality. In future experiments more sophisticated clustering techniques, such as some of the methods developed by the correlation clustering community (Bansal, Blum, and Chawla, 2004), will be evaluated for suitability of use in the performance-sensitive setting of online policy tree construction.

5.5 LPDM Solver

LPDM is based on the Anytime Regularized DESPOT (AR-DESPOT), an anytime version of the original DESPOT (Ye, Somani, Hsu, and Lee, 2017). AR-DESPOT incrementally builds a partial policy tree \( D \) representing policy \( \pi \) using \( K \) pre-sampled scenarios. To guide tree construction and search within the tree, a lower bound \( l(b) \) and an upper bound \( \mu(b) \) are maintained on a specially defined regularized utility value for each node \( b \) in \( \pi \). This value, Regularized Weighted Discounted Utility
(RWDU), is defined as
\[
v_\pi(b) = \frac{|\Phi_b|}{K} \gamma^{\Delta(b)} \hat{V}_{\pi_b}(b) - \lambda |\pi_b|,
\]
where $|\Phi_b|$ is the number of scenarios passing through node $b$ (i.e., the number of state particles in $b$), $\gamma$ is the discount factor, $\Delta(b)$ is the depth of $b$ in the policy tree $\pi$, $\pi_b$ is the subtree rooted at $b$ (with $|\pi_b|$ as its size), and $\lambda \geq 0$ is the regularization parameter similar to those in many machine learning algorithms. The ratio $|\Phi_b|/K$ is an empirical estimate of the probability of reaching $b$, which then implies that $v_\pi(b) = \hat{V}_\pi(b_0) - \lambda |\pi|$, i.e. it is simply a regularized version of the general approximate utility function for policy $\pi$.

The algorithm conducts a series of exploratory simulations to expand $D$ and reduce the gap between $l(b_0)$ and $\mu(b_0)$ at the root node $b_0$. Each simulation traverses a promising path from $b_0$ to add new nodes to $D$. In particular, it repeatedly chooses and expands a promising leaf node and adds its child nodes to $D$ until the child node no longer appears promising. The algorithm then traces the path back to the root, performing backup on the upper and lower bounds of each node along the path. These exploratory simulations continue until one of the following termination conditions is reached: (1) the gap between $l(b_0)$ and $\mu(b_0)$ is reduced to some target level $\epsilon_0 \geq 0$, (2) the allocated planning time runs out, or (3) the maximum number of tree-building simulations is reached.

LPDM, outlined in Algorithm 5.6, retains the high-level structure of AR-DESPOT (the $K$ pre-sampled scenarios and the upper/lower bounds maintained on tree nodes RWDU values), while substantially changing the way a policy tree is constructed. Lines 10–13 compute the solver “temperature” parameter $\tau_{\text{solver}}$, used by AAS (Algorithm 5.2) for action branching. On line 14, when the tree exploration procedure $\text{Explore}$ is invoked (Algorithm 5.7), $\tau_{\text{solver}}$, $\text{maxClusters}$ (the desired maximum number of observation clusters), and $N$ (the maximum number of node exploitation visits) are passed into it.

$\text{Explore}$ traverses the LPDM policy tree $T$, picking action branches with the highest upper value bound (line 7) and observation branches with the highest excess
Algorithm 5.6 LPDM algorithm

1: procedure LPDM($b_0$, $maxTrials$, $N$, $maxClusters$) 
   ▷ $N$ is the number of exploitation visits to a node before adding an action branch
2: \quad $\Phi_{b_0} \leftarrow$ sample $K$ particles from $b_0$
3: \quad $T \leftarrow b_0$ \hspace{1cm} ▷ create a new policy tree $T$ with $b_0$ as its root
4: \quad $T.visits[b_0] \leftarrow 0$ \hspace{1cm} ▷ $T.visits$ is a list of node visit counters
5: \quad Initialize $\mu(b_0), l(b_0)$ \hspace{1cm} ▷ initialize upper and lower value bounds
6: \quad $A_{b_0} \leftarrow A_0$ \hspace{1cm} ▷ initialize the node’s action space ($A_0$ can be $\emptyset$)
7: \quad $\epsilon(b_0) \leftarrow \mu(b_0) - l(b_0)$
8: \quad $nTrials \leftarrow 0$
9: \quad while $nTrials < maxTrials$ and $t < t_{\max}$ and $\epsilon(b_0) > \epsilon_0$ do
10: \quad \quad $\tau_\epsilon = 1 - \epsilon_0/\epsilon(b_0)$
11: \quad \quad $\tau_{time} = 1 - t/t_{\max}$
12: \quad \quad $\tau_{trials} = 1 - nTrials/maxTrials$
13: \quad \quad $\tau_{solver} = \text{MIN}(\tau_\epsilon, \tau_{time}, \tau_{trials})$
14: \quad \quad $b \leftarrow \text{EXPLORE}(T, b_0, \tau_{solver}, maxClusters, N)$
15: \quad \quad $\text{Backup}(T, b)$
16: \quad \quad $\epsilon(b_0) \leftarrow \mu(b_0) - l(b_0)$
17: \quad \quad $nTrials \leftarrow nTrials + 1$
18: \quad end while
19: \quad return $a^* \leftarrow \arg \max_{a \in A_{b_0}} \mu(b_0, a)$
20: end procedure

uncertainty (line 8), the latter computed using Equation 5.2:

$$EU(b) = \epsilon(b) - \frac{|\Phi_{b_0}|}{K}\epsilon_0.$$  \hspace{1cm} (5.2)

Visitation counts for nodes in $T$ are kept, for use by AAS.

When the traversal encounters a leaf mode, the Expand procedure is called (Algorithm 5.8). AR-DESPOT, when expanding action branches from a leaf node $b$, expands all actions in $A$ at once. LPDM, in contrast, starts with an empty set $A_b = \emptyset$ ($A_b$ could also be initialized with a small “seed” set $A_0$) and uses AAS to grow $A_b$. Thus all node action spaces in a policy tree built by AR-DESPOT are the
same, while in a tree built by LPDM they may all end up differing from one another.

Algorithm 5.7 LPDM tree exploration procedure

1: procedure Explore(T, b, τ\textsubscript{solver}, maxClusters, N)
2: while ∆(b) < H and EU(b) > 0 do
3:     if CHILDREN(b) = ∅ or (T.visits[b] > N and |A\textsubscript{b}| < |A|\textsubscript{max}) then
4:         T ← Expand(T, b, τ\textsubscript{solver}, maxClusters)
5:     end if
6:     a\textsuperscript{*} ← arg max\textsubscript{a∈A\textsubscript{b}} μ(b, a)
7:     c\textsuperscript{*} ← arg max\textsubscript{c∈C\textsubscript{b,a\textsuperscript{*}}} EU(CHILD(b, a\textsuperscript{*}, c))
8:     T.visits[b] ← T.visits[b] + 1
9:     b ← CHILD(b, a\textsuperscript{*}, c\textsuperscript{*})
10: end while
11: return b
12: end procedure

After simulating state transitions initiated by action a for all particles in the leaf node b, AR-DESPOT assigns two or more follow-on states to the same new next-step belief if their observation values match. In LPDM, all of the state transitions out of belief b initiated by action a are also generated at the same time. However, the follow-on states from these transitions are grouped into beliefs using the TCC procedure (Algorithm 5.3), which does not require comparing observations directly — thus supporting complex observation spaces. After the new beliefs are formed, their action spaces are initialized by TCC with A\textsubscript{b} = ∅ and the tree exploration cycle continues.

In LPDM, Expand is invoked not just for leaf nodes, but also for in-tree nodes (line 3 of Algorithm 5.7), once the count of visits to an in-tree node since the last action added exceeds N. After the Explore procedure completes, the Backup procedure
Algorithm 5.8 LPDM tree expansion procedure

1: **procedure** \textsc{Expand}(\mathcal{T}, b, \tau_{\text{solver}}, \text{maxClusters})

2: \hspace{1em} \texttt{a}^{*} \leftarrow \arg \max_{a \in A_{b}} \mu(b, a)

3: \hspace{1em} a_{\text{new}} \leftarrow \textsc{AdaptiveActionSelection}(b, A_{b}, a^{*}, \tau_{\text{solver}})

4: \hspace{1em} A_{b} \leftarrow A_{b} \cup a_{\text{new}}

5: \hspace{1em} \mathcal{T} \leftarrow \textsc{TransitionCorrelationClustering}(\mathcal{T}, b, a_{\text{new}}, \text{maxClusters})

6: \hspace{1em} \textbf{return} \mathcal{T}

7: **end procedure

(Algorithm 5.9) is invoked, performing backup on the upper and lower value bounds of each node along the path to the root of \mathcal{T}. Similarly to AR-DESPOT, the backup procedure is based on Bellman’s equation (Equation 2.3). The tree-building process continues until the number of trials is exceeded, the maximum allotted decision time passes, or the gap between \(l(b_0)\) and \(\mu(b_0)\) is reduced to \(\epsilon_0\) or below (line 9 of Algorithm 5.6). It is believed that despite the differences in policy tree construction, LPDM retains AR-DESPOT’s convergence properties; a formal proof is a goal for future work.

Algorithm 5.9 LPDM tree backup procedure

1: **procedure** \textsc{Backup}(\mathcal{T}, b_{\text{current}})

2: \hspace{1em} \textbf{for all} \; b \; \textbf{on the path from} \; b_{\text{current}} \; \textbf{to} \; b_0 \; \textbf{do}

3: \hspace{2em} \triangleright \; \text{regularized weighted discounted one-step reward:}

4: \hspace{3em} \rho(b, a) = \frac{1}{K} \sum_{\phi \in \Phi_{b}} \gamma_{\Delta(b)} R(s_{\phi}, a) - \lambda

5: \hspace{3em} \mu(b) \leftarrow \max \left\{ l_{0}(b), \max_{a \in A_{b}} \left\{ \rho(b, a) + \sum_{c \in C_{b,a}} \mu(\text{CHILD}(b, a, c)) \right\} \right\}

6: \hspace{3em} l(b) \leftarrow \max \left\{ l_{0}(b), \max_{a \in A_{b}} \left\{ \rho(b, a) + \sum_{c \in C_{b,a}} l(\text{CHILD}(b, a, c)) \right\} \right\}

7: \hspace{2em} \triangleright \; l_{0} \text{ is the initial lower value bound}

8: \hspace{1em} \textbf{end for}

9: **end procedure

Note that AR-DESPOT implements a pruning procedure as part of its tree exploration algorithm, intended to eliminate tree branches with a low number of scenarios going through them and thus alleviate overfitting (Ye, Somani, Hsu, and Lee, 2017).
LPDM implements a similar procedure, although it is omitted from this discussion for brevity.

5.6 Experiments

Three sets of experiments were conducted using different variants of a well-known partially observable, continuously valued problem — the Light-Dark Domain (Platt, Tedrake, Kaelbling, and Lozano-Perez, 2010). The first set of experiments was focused on evaluating the performance of LPDM versus several configurations of AR-DESPOT on the canonical, two-dimensional partially stochastic variant of Light-Dark, as well as on its one-dimensional variant. In the second set of experiments, the Fully Stochastic Light-Dark (FSLD) problem was introduced (also in one- and two-dimensional variants) and similar performance evaluations are conducted. The third set of experiments aimed to evaluate how the decision-making performance of both AR-DESPOT and LPDM is affected by model uncertainty. This section describes the Light-Dark Domain, explains how the experiments were set up, and presents their results.

5.6.1 Light-Dark Domain

The canonical Light-Dark problem formulation implemented for these experiments follows the pattern provided by Platt et al (2010). In it, a robot is traveling on a two-dimensional (2D) plane where light varies as a quadratic function of the horizontal coordinate (Figure 5.1). The amount of light available at the current position determines the robot’s ability to localize itself (i.e., observe its position). The objective of the robot is to reach the goal position at the origin (marked with an X in Figure 5.1).

An additional, one-dimensional (1D) variant of the problem is also introduced in order to provide a better basis for LPDM performance assessment. These variants of the problem are referred to as LD 1D and LD 2D, respectively.

In the LD 2D variant, a state is a real-valued vector $s \in \mathbb{R}^2$, with named components $x$ and $y$ for convenience. An action is also a real-valued vector $a \in \mathbb{R}^2$. The observation function of the robot $Z(s_n) = [s_n.x, s_n.y] + \omega$ includes a zero-mean
Gaussian noise term $\omega \sim \mathcal{N}(0, \omega(x))$, with

$$\omega(x) = \frac{1}{2}(x_{mn} - s_n.x)^2,$$

(5.3)

where $x_{mn}$ is the minimum-noise horizontal coordinate. The reward function is defined as

$$R(x) = x^T Q x + a^T R a,$$

(5.4)

where $Q$ and $R$ are cost (negative reward) matrices of being in a particular state and taking a particular action in that state, respectively. In both LD 1D and LD 2D, system dynamics are linear, with no action outcome uncertainty, i.e., $s_{n+1} = s_n + a$.

In the 1D variant, the state plane is replaced with a line, $s \in \mathbb{R}$, and $a \in \mathbb{R}$. $Q$ and $R$ are scalars.

A fully stochastic Light Dark problem is also introduced, in order to increase the degree of difficulty for the algorithms. Its 1D and 2D variants are referred to as FSLD 1D and FSLD 2D, respectively. Their only difference from the canonical counterparts is that state transitions no longer deterministic, but instead modeled as $s_{n+1} = s_n + a + \nu$. For FSLD 1D, $\nu \sim \mathcal{N}(0, \xi a)$, where $a \in \mathbb{R}$ and $\xi$ is a scalar.
specifying the distribution variance relative to \(a\), referred to as the \textit{model uncertainty coefficient}. For FSLD 2D, \(\nu \sim \mathcal{N}_2([0,0], \xi a)\) where \(a \in \mathbb{R}^2\), and \(\xi\), once again, is the model uncertainty coefficient.

### 5.6.2 Modeling

Two separate POMDPs are defined for the 1D and the 2D problems. The state and observation spaces are real-valued, with values within each dimension limited to the \([-10.0, 10.0]\) interval. For use with AR-DESPOT, the observation space is discretized into evenly sized bins. The discrete action spaces used with AR-DESPOT are described in Section 5.6.3, while AAS \textit{SampleAction} procedures for the 1D and 2D problems are provided in Algorithms 5.10 and 5.11, respectively. State transition, observation, and reward models for the two problems are as defined in the previous section.

**Algorithm 5.10** AAS sampling procedure for Light Dark 1D

```plaintext
1: procedure SampleAction(b, \(A_b\), \(\hat{a}_b^*\), \(T\))
2: \(radius \leftarrow T(\max(A) - \min(A))\)
3: \(a_{\min} \leftarrow \max\{\min(A), \hat{a}_b^* - radius\}\)
4: \(a_{\max} \leftarrow \min\{\max(A), \hat{a}_b^* + radius\}\)
5: \(inSet \leftarrow true\)
6: \(\textbf{while} \ inSet \ \textbf{do}\)
7: \(a \leftarrow \text{uniformly sample from } [a_{\min}, a_{\max}]\)
8: \(\text{if } a \in A_b \text{ then}\)
9: \(\quad inSet \leftarrow true\)
10: \(\text{else}\)
11: \(\quad inSet \leftarrow false\)
12: \(\text{end if}\)
13: \(\textbf{end while}\)
14: \(\textbf{return } a\)
15: \(\textbf{end procedure}\)
```

AR-DESPOT/LPDM upper bound heuristics for 1D and 2D problems are similar: the robot is assumed to know its current state well enough to be able to go directly to
Algorithm 5.11 AAS sampling algorithm for Light Dark 2D

1: procedure SAMPLE_ACTION($b$, $A_b$, $\hat{a}_b^*$, $T$)
2: $radius \leftarrow T(\max(A) - \min(A))$
3: $a_{\min,x} \leftarrow \max\{\min(A), \hat{a}_b^*.x - radius\}$
4: $a_{\max,x} \leftarrow \min\{\max(A), \hat{a}_b^*.x + radius\}$
5: $a_{\min,y} \leftarrow \max\{\min(A), \hat{a}_b^*.y - radius\}$
6: $a_{\max,y} \leftarrow \min\{\max(A), \hat{a}_b^*.y + radius\}$
7: $inSet \leftarrow true$
8: while $inSet$ do
9: $a_x \leftarrow$ uniformly sample from $[a_{\min,x}, a_{\max,x}]$
10: $a_y \leftarrow$ uniformly sample from $[a_{\min,y}, a_{\max,y}]$
11: if $\langle a_x, a_y \rangle \in A_b$ then
12: $inSet \leftarrow true$
13: else
14: $inSet \leftarrow false$
15: end if
16: end while
17: return $\langle a_x, a_y \rangle$
18: end procedure

the origin, choosing action values that minimize the total number of actions that have to be taken. This is done by first taking the largest magnitude action available that does not exceed the straight-line distance to the origin, then covering the rest of the distance with smaller magnitude actions. The cumulative reward for this trajectory is then computed using Equation 5.4 and returned as the upper value bound.

Lower bound heuristics follow a similar pattern, with one key difference. It is assumed that in the worst case scenario, the only way for the robot to accurately estimate its position is to first go to the low-noise (lighted) region. From there, once localized, it is able to choose the minimal number of straight-line actions to get to the origin. The cost for this two-leg trajectory is then computed and returned as the lower value bound.
5.6.3 Setup

Parameters common to all Light-Dark experiments are summarized in Table 5.1. Values chosen for some of the 2D experiment parameters merit a more detailed discussion. From the discussion on computational complexity in Chapter 4, it follows that in order to ensure that a 2D policy tree is constructed with the same level of detail (referred to henceforth as policy fidelity) as a 1D policy tree, an MCTS-style POMDP solver operating on traditional discrete $A$ and $O$ spaces would need some of the 1D parameters scaled quadratically for the 2D case. These parameters include: $K$ (the number of particles in $b_0$), the maximum number of trials, the maximum $|A|$, as well as the number of $O$ bins (AR-DESPOT) or the maximum number of LPDM belief clusters. As can be seen in Table 5.1, however, some of these parameters are left the same in the 2D case and some are increased only moderately. This is done intentionally, in order to evaluate whether LPDM, with its adaptive capabilities, can do better than AR-DESPOT in a setting where computational resources are limited and where constructing a higher-fidelity policy is not feasible.

Table 5.1: Common parameters for Light-Dark experiments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>1D</th>
<th>2D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tests per scenario</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>State cost coefficient $Q$</td>
<td>0.5</td>
<td>diag(0.5, 0.5)</td>
</tr>
<tr>
<td>Action cost coefficient $R$</td>
<td>0.5</td>
<td>diag(0.5, 0.5)</td>
</tr>
<tr>
<td>Minimum noise $x$ coordinate</td>
<td>5.0</td>
<td>5.0</td>
</tr>
<tr>
<td>Planning horizon, $H$</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>$K$ (number of $b_0$ particles)</td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>Max tree-building trials</td>
<td>$10^4$</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Max $</td>
<td>A</td>
<td>$, LPDM (also $M$ for Blind Value)</td>
</tr>
<tr>
<td>$O$ bins (AR-DESPOT) and max clusters (LPDM)</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Two action spaces for each of the Light-Dark (LD) problems were specified for AR-DESPOT, standard and extended, with the former meant to evaluate the solver’s
performance when fewer actions are explored more extensively and the latter used to
instead try out more action options within the same simulation budget. These action
spaces are described in Table 5.2.

Table 5.2: AR-DESPOT action spaces for Light-Dark experiments

<table>
<thead>
<tr>
<th>Action space</th>
<th>Symbol</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard, 1D</td>
<td>$A_{s1d}$</td>
<td>${-1.0, -0.1, -0.01, 0.0, -0.01, 0, 0.01, 0.1, 1.0}$</td>
</tr>
<tr>
<td>Extended, 1D</td>
<td>$A_{e1d}$</td>
<td>$A_{s1d} \cup 2A_{s1d} \cup 3A_{s1d} \cup 4A_{s1d} \cup 5A_{s1d}$</td>
</tr>
<tr>
<td>Standard, 2D</td>
<td>$A_{s2d}$</td>
<td>2D permutations of $A_{s1d}$</td>
</tr>
<tr>
<td>Extended, 2D</td>
<td>$A_{e2d}$</td>
<td>2D permutations of $A_{e1d}$</td>
</tr>
</tbody>
</table>

Three sets of experiments (benchmarking with LD 1D and LD 2D, benchmarking
with FSLD 1D and FSLD 2D, and model uncertainty) were conducted. Their design
details are described next.

Benchmarking Experiments on Partially Stochastic Light-Dark Problems

Four starting states were chosen for each of the two canonical LD problems. There
were only two criteria for starting states selection: (1) they had to be representative of
regions with qualitatively different levels of state uncertainty and (2) their coordinates
had to be irrational numbers to make action selection more difficult. The following
1D starting states were used: $-2\pi$, $\pi/2$, $3\pi/2$, and $2\pi$. For the 2D case, the four
starting states were $[-2\pi, \pi]$, $[\pi/2, -\pi/2]$, $[\pi, 2\pi]$, and $[2\pi, -\pi]$.

Benchmarking Experiments on Fully Stochastic Light-Dark Problems

The setup for the experiments with FSLD 1D and FSLD 2D is virtually identical to
that for the canonical LD variants. The only difference is that uncertainty in action
outcomes is introduced, both for the “world” models and those used by the decision
making algorithms. The value of $\xi$ (Section 5.6.1) is set to 0.2.
CHAPTER 5. ALGORITHMS FOR COMPLEX POMDPs

Model Uncertainty Experiments

This set of experiments is designed as a quantitative evaluation of the effect of model uncertainty on decision-making quality and as a precursor to a similar set of experiments for VIPER rover models in SHERPA (Chapter 6). In it, a mismatch is introduced between the state transition models $T(s,a,s')$ used by the “world” simulations and those used by the solvers. The “world” state transition models are left deterministic (as in the canonical LD implementations); however in their counterparts used by the solvers, uncertainty in action outcomes is incorporated similarly to FSLD. The following values of the model uncertainty coefficient (Section 5.6.1) are used: $\xi \in \{0.1, 0.2, \ldots, 1.0\}$.

Two solver configurations are used in the experiments: LPDM with both AAS and TCC enabled and DESPOT with an extended action set. The latter is chosen as it consistently performed better in the benchmarking experiments than the other two DESPOT configurations. One starting state is used per Light-Dark configuration: $s_0 = \pi/2$ for LD 1D and $s_0 = [\pi/2, -\pi/2]$ for LD 2D. The rest of the parameters remains the same as in Table 5.1.

5.6.4 Results

Results of the experiments on the different versions of Light-Dark are provided in Appendix A. The number of steps taken to reach the goal location is reported for all the experiments; unlike the work of Sunberg (2018), however, optimization is done for the cumulative reward — i.e., utility — based on Equation 5.4 (same as done by Platt, Tedrake, Kaelbling, and Lozano-Perez, 2010). The rest of this section provides an analysis of the results for each of the three sets of experiments.

Benchmarking Experiments on Partially Stochastic Light-Dark Problems

Results of the performance benchmarking experiments for the LD 1D problem are summarized in Table A.1 and for the LD 2D case, in Table A.2. Both configurations of LPDM generally performed well, on par with or better than the different AR-DESPOT configurations. Out of the latter, AR-DESPOT with a large discrete action
set performed best overall (somewhat surprisingly), possibly because the set was well-spaced. While AR-DESPOT with Blind Value actions could choose from the same number of action options, those selections were more subject to chance and did not always result in the best choices.

While the advantages of AAS and TCC did not particularly stand out in the LD 1D experiments (likely because discretization bins provided sufficient coverage for the 1D A and O spaces), the value of the new methods became clear in the LD 2D experiments. The limits on the number of action and observation branches resulted in an insufficient coverage of the corresponding model spaces whenever discretization was used, which in turn led to inaccurate belief states and poor action choices.

**Benchmarking Experiments on Fully Stochastic Light Dark Problems**

Results of the benchmarking experiments for FSLD 1D and FSLD 2D problems are summarized in Tables A.3 and A.4, respectively. The results generally follow the same pattern as for the canonical, partially stochastic LD variants. The two configurations of LPDM generally do well on FSLD 1D, but not remarkably better than the highest-performing configuration of AR-DESPOT (one with an extended action set). The value of the action- and observation-handling methods in LPDM once again becomes apparent in the 2D case, where LPDM with AAS and discrete observations consistently outperforms all AR-DESPOT configurations and LPDM with both AAS and TCC enabled performs better still.

**Model Uncertainty Experiments**

Results for the 1D model uncertainty experiments are reported in Table A.5 and for the 2D case, in Table A.6. In the 1D case, decision making performance of both solvers degraded at roughly the same rate as $\xi$ increased, with LPDM perhaps performing better at the higher values of $\xi$. In the 2D case, while performance of both AR-DESPOT and LPDM degrades sharply as $\xi$ increases, the rate of degradation for AR-DESPOT is notably higher. These results can again be attributed to the lower inherent 2D policy fidelity, induced by the choice of algorithm parameters (as
discussed in Section 5.6.3). While the lower policy fidelity affects the performance of both solvers, LPDM is able to compensate for it better through adaptive action selection and a higher precision in belief formation.

5.7 Discussion

This chapter presented a set of algorithms designed to alleviate some of the potential computational complexity increases associated with HADM (Chapter 4). These algorithms are, however, intended to be general and applicable to a wide variety of decision making problems that involve complex state, action, and observation spaces.

The first of the algorithms, Adaptive Action Selection, took inspiration from Fast Simulated Annealing, a method for expensive global black-box optimization. AAS can handle complex action spaces in a POMDP solver while avoiding the problem of insufficient exploration, afflicting some of the alternative methods. Perhaps even more importantly, it could serve as a template for tailoring other existing global black-box optimization methods to problems formulated as POMDPs with complex action spaces. Convergence or performance guarantees associated with such methods may then also be transferable.

The second algorithm, Transition Correlation Clustering, is created to handle complex observation spaces within POMDPs. Rather than clustering in observation or belief spaces (which includes discretizing them), TCC clusters entire \( (s, o, s') \) state transitions for some action \( a \), creating next belief states on the basis of the resulting clusters. In TCC, the actual values of observations are only used through the observation function \( Z \), therefore the algorithm imposes no restrictions on either cardinality or dimensionality of the observation space \( O \). The number of observation branches (i.e., clusters), \( N \), does not need to provide an adequate coverage of \( O \), as is the case with discretization or clustering methods for observation spaces. Instead, \( N \) only needs to be as large as the number of distinct beliefs that should be analyzed in selecting the next action. These distinct beliefs can be thought of as system modes that may require specialized responses. For instance, possible system modes for the lunar rover after executing a drive command may include a nominal mode, a
low-battery mode, a communication dropout mode, and a motor fault mode. While \(|O|\) is likely to be very large, perhaps infinite, the number of these general modes of interest is limited for most systems. Another benefit of TCC is that observations that are dissimilar, but nevertheless point to the same system mode, can be clustered together, potentially increasing the fidelity of the belief state approximation.

The two algorithms were incorporated into the LPDM solver, built on the basis of AR-DESPOT, and three sets of experiments were conducted to assess its performance. The experiments were performed using four variants of a well-known benchmarking problem, Light-Dark Domain: 1D and 2D variants of the canonical, partially stochastic Light-Dark Domain, as well as 1D and 2D variants of a fully stochastic Light-Dark implementation. The 2D problem was made more challenging by limiting the increases in \(K\) (the number of tree-building scenarios) and in the maximum numbers of action and observation branches, relative to the 1D case (these parameters would have needed to be increased quadratically to guarantee the same policy fidelity as in the 1D case).

In order to isolate the performance effects of AAS and TCC, LPDM was compared against the original AR-DESPOT solver, which relies on discrete actions and observations, as well as a version of AR-DESPOT augmented with the Blind Value action selection method (LPDM itself can be run with either or both AAS and TCC enabled). AR-DESPOT configured with discrete actions had action spaces of two different sizes for each of the Light-Dark variants. The first set of experiments benchmarked LPDM and AR-DESPOT configurations on the canonical Light-Dark variants, the second did the same with the fully stochastic Light-Dark, while the third consisted of model uncertainty experiments where a mismatch between models used for “world” simulations and those used for decision making by the solvers was introduced.

A common pattern emerged across all three sets of experiments. When tested on the 1D variants of Light-Dark, LPDM either performed on par with the best configurations of DESPOT or showed a modest improvement relative to them. However, on the more difficult 2D problems LPDM consistently demonstrated superior performance. This is due to LPDM being able to (1) better utilize the limited number of
action branches with more context-appropriate action choices and (2) form more precise beliefs within the constraints on observation branching. Even in the LD 2D model uncertainty experiments, where the performance of both solvers dropped steeply as model uncertainty increased, performance of DESPOT with discrete actions and observations deteriorated at a notably higher rate.

In the next chapter, LPDM forms the foundation of SHERPA, a new decision support system applied to the VIPER robotic lunar rover mission. Decision support for VIPER is a real-world problem where all of the key thesis topics covered so far — system health management, decision making under state and outcome uncertainty, and modeling with complex state, action, and observation spaces — become important ingredients of the solution.
Chapter 6

Application: SHERPA

Robotic space missions rank among the most challenging human endeavors, due to the complexity of the systems involved, the remote nature of operations, and the harshness of the operating environment. From the early days of space exploration until today, operating such missions has typically required teams of human experts managing spacecraft health and making operational decisions, both tactical and strategic. However, even on those missions where communication delays are moderate and mission duration is long enough to accommodate decision making delays, relying on operational concepts that involve a large amount of human effort may result in substantial inefficiencies. As Gaines et al. (2016) illustrate on the example of the current Mars Curiosity rover mission, limitations in communication opportunities and the time expended on analyzing vehicle anomalies have resulted in productivity challenges and underutilization of the vehicle.

This chapter describes a new decision support framework for robotic space missions—System Health Enabled Realtime Planning Adviser (SHERPA), developed on the foundation of the Health-Aware Decision Making approach discussed in Chapter 4 and the algorithms presented in Chapter 5. SHERPA’s first application is the NASA Volatiles Investigating Polar Exploration Rover (VIPER) mission. The chapter overviews the general architecture of SHERPA, discusses VIPER-specific models and supporting algorithms, then presents a series of experiments designed to assess SHERPA’s performance on VIPER mission scenarios.
CHAPTER 6. APPLICATION: SHERPA

6.1 Introduction

A number of automated decision support tools for space mission operations have become available in the last two decades (with some notable examples discussed in Section 6.2). Over the course of the work described in this thesis, a new decision support system for robotic space missions was developed—System Health Enabled Realtime Planning Adviser (SHERPA). SHERPA is built on Health-Aware Decision Making principles and utilizes the LPDM solver as its primary reasoning engine. SHERPA’s architecture is intended to be flexible enough to address different needs throughout the lifetime of a mission. For example, in the mission definition phase, SHERPA may be used for engineering design studies, helping with sizing of the spacecraft batteries or determining the parameters of its communication system. Prior to launch of a planetary surface mission, SHERPA may assist with landing site selection and vehicle traverse planning. During the mission, SHERPA could suggest the next best action to take in a particular situation so that the chances of mission success are maximized.

SHERPA is first being used on the VIPER mission, currently under development at NASA Ames Research Center. The main goal of the mission is to land a robotic rover in a polar region of the Moon and use it to locate and characterize deposits of water ice and other volatiles. The mission will consist of two phases. The primary phase, intended to last at least 100 Earth days, will focus on collecting samples and data at pre-determined sites of interest. The second, extended mission phase will be devoted to calibrating and refining the theoretical distribution models of hydrogen-based volatiles (Section 6.4). Rover traverses in the second phase will be planned in a more dynamic manner, determined—to a large extent—by the data obtained during the preceding parts of the mission.

The VIPER mission is based on the earlier Resource Prospector (RP) mission concept (Andrews, Colaprete, Quinn, Chavers, and Picard, 2014). VIPER presents a number of traverse planning and resource management challenges, including those
particular to operating in lunar polar regions. Past and current Mars rover missions (e.g., Mars Exploration Rovers or Mars Science Laboratory) operated in near-equatorial locations where large changes in solar illumination and ambient temperature occur seasonally. In lunar polar locations, on the other hand, large changes in illumination (and therefore ambient temperatures) occur on hourly scales. Unlike the current Mars Curiosity rover, powered by a radioisotope thermoelectric generator, the VIPER rover will be powered by solar panels only. This implies the need for careful power management and scheduling of recharge opportunities in time-variant sunlit locations. The rover will be teleoperated from Earth, which necessitates maintaining a persistent communication link through the Deep Space Network (DSN)—also a difficult task in the lunar polar regions. The dynamic nature of solar illumination and communication coverage results in a large state space for the rover. Action execution uncertainty is introduced by the variability in the effective driving speed (due to the need to avoid obstacles, such as rocks and craters), variability in execution times of scientific activities, and irregularities in power consumption and generation. Uncertainty in state estimation results from the difficulties in determining the state of battery charge exactly and from having only approximate a priori information about volatile deposit locations and depths.

The SHERPA use case described in this chapter, Traverse Evaluation and Refinement (TER), is focused on optimizing rover traverses that were partially specified by VIPER’s science and operations teams for the primary mission phase. The second use case under development, Traverse Synthesis (TS), is expected to be used more in the extended mission phase. It is designed to take only high-level mission constraints and requirements as inputs and then generate complete traverses, including the associated science and recharge activity schedules. TS is described in Chapter 7 as part of ongoing work. While these use cases are based on RP/VIPER mission requirements and use the corresponding data, models, and rover specifications, they are general enough to be adapted for other robotic planetary exploration missions in the future.

SHERPA’s performance on the TER use case is benchmarked against rule-based policies designed to mimic likely decisions by human operators following mission flight rules and various strategies intended to maximize the mission’s chances of success. A
“prognostic” policy implementing one of the current approaches to decision making (Chapter 3) is also used for the purposes of comparison. In addition to scenarios where only recoverable system faults and delays may occur, experiments involving a more severe fault type are also conducted.

The rest of this chapter is organized as follows. Section 6.2 provides an overview of related work. The architecture and general capabilities of SHERPA are described in Section 6.3. Section 6.4 provides the relevant details of the VIPER mission, while Section 6.5 describes how the mission is modeled in SHERPA. The Traverse Evaluation and Refinement use case is the focus of Section 6.6. Validation and benchmarking experiments are described in Section 6.7. The chapter concludes with a discussion in Section 6.8.

6.2 Related Work

This section describes prior and related work on automated decision support tools for robotic space missions. The section is not intended to be an exhaustive survey, but rather as an illustration of how SHERPA improves on the current state of the art by comparing it with notable or representative examples in the following categories: activity planners/schedulers and combined activity/traverse planners. The categories were selected to cover the capabilities of the two SHERPA use cases developed so far for the VIPER mission: (1) Traverse Evaluation and Refinement and (2) Traverse Synthesis. Some of the tools described in this section have been applied to essentially the same problem as SHERPA: decision support for the Resource Prospector mission (the predecessor to VIPER). As mentioned in the Introduction (and discussed more thoroughly in Section 6.4), the nature of the VIPER mission requires not just planning of the rover’s path, but also careful management of its resources and scheduling of its activities.
6.2.1 Activity Planners/Schedulers

A number of planning/scheduling algorithms based on constraint satisfaction have been incorporated in mission support tools. One notable example is Extendable Uniform Remote Operations Planning Architecture (EUROPA), a constraint satisfaction planner that has been used on multiple NASA missions (Frank, Jónsson, and Morris, 2000). In particular, EUROPA was used as the core reasoning algorithm within the Mixed-initiative Activity Plan GENerator (MAPGEN) for the Mars Exploration Rovers mission (Bresina and Morris, 2007). It was also integrated into Ensemble, a ground operations tool used on NASA’s Phoenix Lander and Mars Science Laboratory missions (Bresina and Morris, 2006; Morris, Bresina, Barreiro, Iatauro, and Smith, 2011). More recently EUROPA was used in SPIFe, a software environment for creating activity plans, as well as in its successor Open Scheduling and Planning Interface for Exploration (OpenSPIFe), described by Bresina, Morris, Deans, Cohen, and Lees (2017). Several other space mission planning and scheduling tools built on constraint satisfaction principles are discussed by Chien et al. (2012).

SHERPA is distinguished from the examples above by four primary characteristics: (1) it reasons over probabilistic models of action outcomes and state uncertainty, rather than making deterministic assumptions or relying on margins defined a priori; (2) it pursues optimality of solutions rather than satisfiability; (3) it incorporates reasoning about the state of vehicle health (including making decisions about recovery or mitigation actions, if necessary); and (4) its capabilities extend beyond just activity planning and scheduling.

Multi-User Scheduling Environment (MUSE) introduced multi-objective optimization to the task of scheduling mission activities via use of evolutionary algorithms (Johnston and Giuliano, 2009). It was used on the Cassini mission to Saturn and is planned to be used for scheduling observations on James Webb Space Telescope. MUSE, however, is still limited to performing scheduling tasks only and does not incorporate reasoning about uncertainty or system health.
6.2.2 Combined Activity/Traverse Planners

Bresina, Dearden, et al. (2002) provide a good overview of the challenges inherent in the problem of combined traverse and activity planning for planetary rovers (including the various types of uncertainties likely to be encountered) and discuss why some of the early automated planning systems were unsuitable to solving it. More recently there has been a number of combined activity/traverse planners developed for exploration rovers, including some that use MDP and POMDP formulations. One of the first systems that successfully combined activity and traverse planning with vehicle resource management is TEMPEST (Tompkins, Stentz, and Wettergreen, 2004). TEMPEST relies on Incremental Search Engine—a graph-theory based, heuristic search algorithm optimized for planning and re-planning in high-dimensional spaces. TEMPEST was demonstrated on simulated Mars mission scenarios, as well as in field experiments in the Atacama Desert with a solar-powered Hyperion rover, tracking battery energy as the consumable resource (Tompkins and Stentz, 2004). Another combined planner, developed by Wu and Ju (2013), takes a constraint-satisfaction approach to the problem. Constraints include those on terrain slope, solar illumination, Earth visibility, thermal and energy state of the rover, as well as its data storage capacity. All constraints are translated to temporal ones and a mission timeline that satisfies them is generated (if one exists).

To deal with the complexities of the Resource Prospector problem, Bresina, Morris, Deans, Cohen, and Lees (2017) adopted a mixed-initiative approach to traverse planning. In their work, the user defines a sequence of locations (stations) where specific science tasks should be performed, as well as the sequences of activities to be executed at each of the stations. The following automated tools for traverse construction are provided as part of the overall Exploration Ground Data Systems (xGDS) architecture: (1) reachability from a station within N hours, (2) temporal bounds on station arrivals and departures, and (3) automated generation of detailed paths between stations. The reachability algorithm uses temporal sunlight and communication maps to identify all locations that can be reached from a specific starting point without losing solar illumination or communications anywhere along the route (assuming rover movement at a constant speed). The temporal bounds algorithm
works by propagating local arrival and departure bounds at various stations along
the traverse (determined by availability of sunlight at that station) throughout the
entire planned traverse, to determine the arrival/departure bounds that ensure the
completion of the mission. Finally, the station-to-station path generation compo-
nent is based on a sampling algorithm, called Heuristic-Biased Stochastic Sampling
(HBSS) (Bresina, 1996). The algorithm samples within a space of minimum-time
paths, which is determined via the reachability analysis.

Also motivated by Resource Prospector, Cunningham et al. (2014) developed
a hierarchical path planner that consists of low- and high-resolution components.
The planning algorithms for both components are based on A* (Hart, Nilsson, and
Raphael, 1968). The low-resolution planner is used for quickly finding long-distance
paths for strategic mission planning. Temperature, energy, and time costs for the
edges in the low-resolution graph are precomputed by the high-resolution planner,
which uses physics-based power, thermal, and kinematic models to determine these
costs. To help deal with the large search space, the time-varying environment is rep-
resented with reduced detail in homogeneous regions. Otten, Jones, Wettergreen, and
Whittaker (2015) apply connected component analysis to the same problem domain
to find interconnected regions in space and time that are continuously lit and satisfy
slope constraints. They then also use A* to find optimal routes within the compo-
nent tree. Surface Exploration Traverse Analysis and Navigation Tool (SEXTANT)
is another notable planner based on A* (Johnson, Hoffman, Newman, Mazarico, and
Zuber, 2010). SEXTANT can generate traverses given user-specified activity points
using various optimization criteria and constraints.

Unlike xGDS or the planner developed by Wu and Ju, SHERPA pursues optim-
mality in its solutions. It also supports formalized reasoning about uncertainty and
can operate in continuously valued state, action, and observation domains (some-
thing that is not possible with HBSS, A*-based approaches, or similar deterministic
algorithms). One of the earlier efforts that is comparable to SHERPA in reasoning
about uncertainty and pursuing optimality is the work by Smith (2007). In it, a
POMDP solver based on heuristic search principles was developed and applied to the
problem of efficiently exploring a planetary surface region for signs of extraterrestrial
life (LifeSurvey). The solver, Focused Real-Time Dynamic Programming (FRTDP), used lower and upper bounds on $V^*$ to direct forward exploration during search—similarly to DESPOT and LPDM. However, discrete state, action, and observation representations were used in the problem formulation. A technique called Conditionally Irrelevant Variable Abstraction (CIVA) reduced the number of discrete states from more than $10^{24}$ to less than $10^4$ (actual numbers depended on the scenario). The number of actions and observations were 7 and 21, respectively. System health management was not part of the problem formulation.

In a more recent effort, (Santana, Vaquero, et al., 2016) extend the traditional POMDP formalism into Chance-Constrained Partially Observable Markov Decision Processes (CC-POMDP) with a notion of probabilistic execution risk and develop Conditional Planning for Autonomy with Risk (CLARK) planning architecture that takes CC-POMDP models as input and generates conditional temporal plans for rovers as output. CLARK is capable of supporting hybrid (discrete and continuous) mission planning problems. Risk-bounded AO* (RAO*) solver is the component of CLARK that constructs risk-bounded CC-POMDP policies by performing heuristic forward search in the space of belief states (Santana, Thiébaux, and Williams, 2016). The solver operates in discrete model spaces; however, in the CLARK architecture it is coupled with pSulu chance-constrained path planner (Ono, Williams, and Blackmore, 2013) in order to accommodate models with continuous dynamics and constraints. CLARK was demonstrated on a relatively small, fully observable problem with five possible Mars rover locations of interest and a discrete set of 17 actions. Time-variant communication coverage by an orbiting satellite was part of the problem formulation, although dynamic illumination was not.

### 6.3 Architecture

SHERPA is intended to be a general and flexible framework to provide support for decision making under uncertainty during NASA space exploration missions. Its architecture is, therefore, modular and model-based, such that a variety of missions
and use cases can be supported (Figure 6.1). Modules describing the specifics of supported missions can be connected with use cases implementing support for different types of tasks, e.g., traverse planning or landing site selection. The support infrastructure provides a library of generic models that can be specialized for a particular use case and a particular mission, potentially saving development time and effort. It also provides a variety of other common tools and serves as the interface to decision making solvers and policies.

![SHERPA architecture diagram](image)

Figure 6.1: SHERPA architecture

The rest of the section provides further details on the various components of SHERPA’s architecture and discusses how system health management is accommodated within it.

### 6.3.1 Solvers/Policies

While LPDM is currently the default reasoning engine within SHERPA, in principle, a variety of other MDP/POMDP solvers and policies can be accommodated — if doing so is beneficial for a particular application. DESPOT, a prognostics-based policy, and
a number of rule-based policies are currently interfaced with SHERPA for performance benchmarking and validation purposes (Sections 6.6.2 and 6.6.3).

6.3.2 Support Infrastructure

The SHERPA support infrastructure provides interfaces to solvers/policies and a variety of tools to make definition of new missions and use cases easier. Visualization tools include graphical user interfaces, intended to help with monitoring progress of a mission and with interpretation of SHERPA’s action recommendations. Input data management tools handle a variety of digital map formats, traverse descriptor files, activity dictionaries, and other types of information. The simulation framework, assisted by benchmarking and statistics tools, can execute individual mission scenarios or automate runs of scenario batches with different solvers, problem parameters, and model fidelity settings. It can also execute individual mission scenarios.

A set of common models is also provided within the support infrastructure. These models include, for example, generic state transition models (e.g., for science activity execution) or observation models for common instrument types. The models can be specialized for a particular application.

6.3.3 Missions

A mission module within SHERPA contains the following: (a) mission parameters and constraints; (b) models of the operating environment; and (c) vehicle models and parameters. It is expected that similar types of missions will be able to share at least some of this information, which is why four broad mission categories are currently defined within SHERPA: rover, lander, orbiter, and flyby. Models and other data structures that are too specific to be part of SHERPA’s high-level support infrastructure, but are still general enough to be applicable to missions within the same category, are defined at this level. Resource Prospector and VIPER, both being rover missions, share many of the same models (e.g., for waypoint navigation), specializing them with their own sets of parameters.
6.3.4 Use Cases

A SHERPA use case is an MDP or a POMDP describing a particular decision making problem. VIPER mission use case examples already mentioned include Traverse Synthesis and Traverse Evaluation and Refinement. Other VIPER use cases currently under development are Landing Site Selection and Vehicle Parameter Optimization. All the current VIPER use cases are formulated as POMDPs, or, more precisely, as MOMDPs (Section 2.4), given that some of the state vector elements are considered fully observable. The only part of the decision-making process where mixed observability currently plays a role, however, is in “world” belief updates. For all particles constituting a belief, a fully observable state element evolves along a single trajectory, while a partially observed element evolves along multiple, particle-specific trajectories. As both DESPOT and LPDM currently treat all state vector elements as partially observable for policy derivation purposes, SHERPA use cases are referred to as POMDPs in the rest of this chapter to avoid ambiguity. Ongoing work to implement full support for MOMDP formulations in LPDM is described in Section 7.2.1.

All the typical POMDP components are required to define a use case, i.e., those constituting the \( \langle S, A, T, R, O, Z, \gamma \rangle \) tuple. If a use case is intended for multiple missions or mission types, then some of these components (such as \( S, A, \) or \( R \)) can be specified at the mission level. Transition, reward, and observation models can also be shared among different use cases. This is true for VIPER, for example, where the physics models describing state transitions for the rover and the observation models for the payload instruments are shared by all the current use cases. Each VIPER use case defines its own reward model, however.

6.3.5 Fault Handling

The faults expected to be encountered during a mission can be broadly classified into three categories: (1) faults that can be recovered from; (2) faults that can be accommodated; and (3) faults that result in mission termination (i.e., failures). Their handling in SHERPA is described below.
Faults that can be recovered from

It is assumed that recovering from a fault in this category may result in expenditures of time, energy, or other consumable resources, but otherwise results in no lasting effects on the health of the vehicle. Faults of this category are handled in SHERPA through modeling of action outcome and state estimation uncertainty. For instance, if the VIPER drill gets stuck on retraction during a drilling operation, but can still be successfully freed by temporarily increasing current to the drill motor above the nominal limit with no lasting damage to the motor, this would be considered a recoverable fault. Its only impacts on the mission are the extra time and battery charge consumed during the activity, which can be modeled as action outcome uncertainties.

Faults that can be accommodated

If a fault cannot be fully recovered from, but can be still be accommodated in a manner that allows at least a partial completion of mission objectives, it can be handled in SHERPA by augmenting the state vector with features specific to the fault and adding relevant state transition and observation models. A detailed example of such a fault is described in Section 6.7.1.

Faults that result in mission termination

If state transitions leading to faults of this type can be modeled, even in an approximate manner (e.g., through component degradation modeling), they will be simulated and accounted for as part of the normal policy computation process. A high negative reward can be assigned to mission failure states and a policy that optimizes the chances of avoiding them can then be computed. To improve handling of faults in this category with low probabilities of occurrence, techniques and tools developed for adaptive stress testing can be applied to generate targeted sets of policy building scenarios (Lee, 2019; Lee, Mengshoel, and Kochenderfer, 2019). For DESPOT and LPDM, these would consists of specifically selected random number sequences.
6.4 VIPER Mission

The VIPER mission is currently targeted for launch in November of 2023. The rover designed under its predecessor mission concept, Resource Prospector (RP), was not expected to survive the lunar night, therefore that mission was planned to last only 10–12 Earth days. In contrast, the VIPER rover (Figure 6.2)—while largely of a similar design—is being tasked with performing science activities over the course of multiple lunar days, with a targeted primary mission duration of 100 Earth days and an extended mission phase planned beyond that.

The remainder of this section provides information on the VIPER mission relevant to this work. A broader overview of the mission can be found in publications by Elphic, Colaprete, Shirley, et al. (2017) and Colaprete, Elphic, et al. (2015). Appendix B contains tables of mission and vehicle parameters used by SHERPA.

![Figure 6.2: The VIPER rover (artist’s conception, NASA)](image)

6.4.1 Candidate Operating Areas

Several potential operating areas for RP/VIPER have been considered, both near North and South lunar poles. North Pole areas include those neighboring Erlanger
and Hermite A craters. At the South Pole, candidate areas are located near Haworth, Shoemaker, and Nobile craters (Elphic, Colaprete, Shirley, et al., 2017). While the South Pole is currently the preferred destination for the VIPER mission, a change to a North Pole location is still possible.

The Hermite A area near the North Pole (Figure 6.3) was used in the experiments described further in this chapter, as some of early experiments were conducted when the North Pole was still considered to be the more likely destination. Hermite A is a crater 20 km in diameter, located at 87.8°N 47.1°W, adjacent to the much larger Hermite crater (109 km in diameter). Hermite crater is notable for being the coldest known location in the solar system, with temperatures reaching as low as 26 K.

![Orbital view of the general region](image1.png)

![Contour map of the Hermite A crater](image2.png)

Figure 6.3: Hermite A crater: the area used in the SHERPA experiments (NASA)

### 6.4.2 Scientific Objectives

The rover is expected to visit sites where orbital measurements and indirect geological analysis indicate a strong possibility of water ice and other hydrogen-based volatiles being near the surface. Of particular interest are permanently shadowed regions (PSRs), illustrated in Figure 6.5b, where the likelihood of finding volatiles within the
reach of the rover instruments is highest. These regions, however, represent significant operational challenges and risks, due to the inability to recharge the rover’s batteries within them and a decreased visibility for obstacle avoidance.

Using digital elevation maps (DEMs) of the polar region — as well as orbital survey readings, the age and size of craters, calculations of past and present solar illumination, and other factors — a model estimating the depth at which water ice and other volatiles of interest are likely to be found was created by the VIPER science team. In both the overall mission design and in SHERPA’s VIPER use cases, this model of the theoretical volatiles stability depth is used to define which science activities are performed at waypoints. One of the mission’s primary scientific objectives is to refine these ice depth models with the ground-truth data collected. The models are described in more detail in Section 6.4.6.

6.4.3 Vehicle

The VIPER rover is solar-powered, storing electrical energy in a Li-Ion battery. The latest iteration of the rover design includes three photo-voltaic (PV) panels, arranged on the sides and the rear of the vehicle. All three solar panels are near-vertical in order to maximize their exposure to sunlight in high-latitude operating areas.

The powertrain consists of four wheels, each driven by a separate electric motor. The most recent design allows for the wheels to rotate about the vertical axis so that optimal (or near-optimal) orientation of the rover relative to the sun can be maintained regardless of direction of travel. In case this feature is not implemented in the final design, however, SHERPA’s VIPER models include computations of the amount of sunlight illuminating the solar panels as a function of time, location, and rover orientation. Table B.1 lists rover design parameters relevant to this work.

6.4.4 Science Payload

VIPER science payload is designed to conduct a number of resource location and characterization activities. It consists of two neutron spectrometers, a mass spectrometer, and a drill — all briefly described in this section.
Near-Infrared Volatile Spectrometer System (NIRVSS)

This neutron spectrometer can sense surface frost and identify surface mineral mixtures by measuring the energy released by hydrogen atoms when struck by neutrons (Elphic, Colaprete, Heldmann, and Deans, 2015). It can also provide additional information on the ice form (e.g., crystalline vs. amorphous) and grain size. NIRVSS can be used while driving.

Neutron Spectrometer System (NSS)

The other neutron spectrometer onboard the rover, NSS, can be used to gauge volumetric hydration and elemental composition variations within the top one meter of soil (Elphic, Colaprete, Heldmann, and Deans, 2015). Like NIRVSS, NSS can be used while driving.

Mass Spectrometer Observing Lunar Operations (MSolo)

MSolo complements NIRVSS and NSS in analyzing mineral and volatile composition of the lunar soil (Ehrenfried, 2020). Its particular strength is in identifying low-molecular weight volatiles. To determine the chemical composition of soil samples, MSolo measures the mass-to-charge ratio of ions. The instrument can be used either while the rover is driving or while it is stationary.

The Regolith and Ice Drill for Exploring New Terrain (TRIDENT)

The TRIDENT drill will be used to extract soil samples from up to 1 m of depth (Zacny et al., 2016). The composition of the samples will then be examined by NIRVSS and MSolo, helping to determine not only the types of volatiles present in the soil and their concentrations, but also the depth from which they were extracted.
6.4.5 Mission Rules

The following high-level mission rules were assumed in the use cases:

**Minimum battery SoC**  
The battery is not be discharged below 40% of its maximum capacity during normal operations.

**DSN coverage**  
The rover is to be teleoperated at all times, thus persistent communications with Earth via the DSN are required. Under this rule, if the DSN link is lost, the rover is to stop its motion or stationary activities and wait for the connection to be regained.

**Maximum terrain slope**  
Rover paths cannot include slopes (positive or negative) greater than 15°.

**Pre-PSR battery SoC**  
The rover battery has to be fully charged before entering a PSR.

6.4.6 Information Available to SHERPA

In addition to the instruments in the VIPER payload package and some of its sensors, several other sources of information serve as inputs for SHERPA’s decision making. An overview of these sources is provided in this section.

**Activity Dictionary**

The *activity dictionary* provides information on the expected average power consumption and duration of various rover activities. Examples of activity dictionary entries are shown in Table 6.2 (values are approximate). In the table, durations denoted with an * indicate variable-length activities. Activity power consumption values account for all the necessary supporting subsystems, e.g., avionics, communications, and thermal control. The activity dictionary is used extensively within SHERPA’s generative model for the VIPER rover (Section 6.5.3) in computing state transitions.
Table 6.2: An example activity dictionary

<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration (sec)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle (stationary mode)</td>
<td>*</td>
<td>200</td>
</tr>
<tr>
<td>Roving / direct drive</td>
<td>*</td>
<td>200</td>
</tr>
<tr>
<td>Prospecting</td>
<td>*</td>
<td>400</td>
</tr>
<tr>
<td>Area of interest mapping (AIM) — 5 m × 10 m</td>
<td>2500</td>
<td>400</td>
</tr>
<tr>
<td>Panorama</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>Quick subsurface assay (50 cm)</td>
<td>4000</td>
<td>350</td>
</tr>
<tr>
<td>Deep subsurface assay (90 cm)</td>
<td>7000</td>
<td>350</td>
</tr>
<tr>
<td>Deep subsurface assay (100 cm)</td>
<td>8000</td>
<td>350</td>
</tr>
<tr>
<td>Shadow idle (stationary mode)</td>
<td>*</td>
<td>300</td>
</tr>
<tr>
<td>Shadow AIM — 5 m × 10 m</td>
<td>3000</td>
<td>500</td>
</tr>
<tr>
<td>Shadow quick subsurface assay (to 50 cm depth)</td>
<td>4000</td>
<td>500</td>
</tr>
<tr>
<td>Shadow panorama</td>
<td>300</td>
<td>500</td>
</tr>
</tbody>
</table>

Ice Stability Depth Models

The theoretical volatiles stability depth models (also referred to as ice stability depth models) have been developed based on the work of Colaprete, Schultz, et al. (2010), Paige et al. (2010), Siegler, Paige, Williams, and Bills (2015), and Siegler, Miller, et al. (2016). For use in SHERPA, the theoretical ice stability depth values are pre-computed for different prospective operating areas — with the same resolution as the maps described in the next section — and stored in 8-bit arrays. The values from the arrays are then used in conjunction with the output of VIPER’s scientific instruments to produce observations of ice depth at the current rover location (Section 6.5.2). These observation are, in turn, used in computing SHERPA’s belief state.

In the current VIPER concept of operations, theoretical ice stability depth values are discretized into four ice stability zone types for the purposes of strategic mission planning (Table 6.3) and ice stability depth zone maps are created on their basis. This and other types of maps used by SHERPA are described next.
Table 6.3: Lunar ice stability depth zone types

<table>
<thead>
<tr>
<th>Zone type</th>
<th>Description</th>
<th>Map color code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dry</td>
<td>Ice is not stable at shallower than 1 m</td>
<td>gray</td>
</tr>
<tr>
<td>Shallow stable ice</td>
<td>Ice is stable between 0 m and 0.5 m below the surface</td>
<td>green</td>
</tr>
<tr>
<td>Deep stable ice</td>
<td>Ice is stable between 0.5 m and 1 m below the surface</td>
<td>yellow</td>
</tr>
<tr>
<td>PSR</td>
<td>Ice is stable at or very near the surface</td>
<td>red</td>
</tr>
</tbody>
</table>

Maps

There are five digital map types currently used by SHERPA in its decision making. Digital elevation maps are used primarily to compute elevation gain or loss along a traverse segment (Figure 6.4a). Slope maps, derived from DEMs, are used for ensuring compliance with the maximum slope constraint (Section 6.4.5). An example of a slope map is provided in Figure 6.4b, where slopes below and above 15° are indicated with green and red pixels, respectively. Ice stability depth zone maps (example in Figure 6.5a) are derived from the ice stability depth models, described earlier in this section. They are used in the Traverse Synthesis use case to define the traverse waypoints and the associated sequences of sciences activities.

Two of the five map types used are dynamic: solar illumination and DSN (communication) coverage. An example of the former is in Figure 6.6a and of the latter, in Figure 6.6b. Maps of both types are pre-computed in 2-hour intervals.

In all the maps used in this work, one pixel represents an area of 20 m × 20 m. Higher resolution maps, 1 m × 1 m per pixel, are currently under development and will be supported by SHERPA in the near future.
CHAPTER 6. APPLICATION: SHERPA

(a) A Digital Elevation Map (DEM) composed from satellite data
(b) A derived map of acceptable slope regions

Figure 6.4: Hermite A crater topography (NASA)

(a) A map of expected ice stability depth zones (see Table 6.3 for the color legend)
(b) A derived map of permanently shadowed regions (PSRs)

Figure 6.5: Ice stability depth zones in the Hermite A crater region (NASA)
6.5 VIPER Mission Modeling

This section describes VIPER model components common to all of the current use cases (i.e., those defined at the mission level). These include the state space, the observation space, and the stochastic generative model. Model components specific to the Traverse Evaluation and Refinement use case are described in Section 6.6.

6.5.1 State Vector

A VIPER state vector $s = (x, y, t, e, d, c)$ contains both fully and partially observable elements. All are assumed to be continuously valued, therefore $|S| = \infty$ (both DESPOT and LPDM solvers within SHERPA are well-suited for dealing with large state spaces). The individual elements of the state vector are described below:

$x, y$ Rover coordinates (in meters) relative to the origin of the area maps. The coordinates are considered fully observable.
Time is included into the state vector as the third (also fully observable) coordinate, since some important derived quantities are time-dependent: illumination, communication coverage, and ambient temperature. Another key reason for having time in the state vector is that it enables use of variable duration macro-actions, such as driving to the next waypoint. Not only the same action can be used in some state $s$ to instruct the rover to drive to a waypoint 1 km away or 2 km, but uncertainty can be introduced in the duration of the drive, that can subsequently be reflected in $s'$.

Energy available is a partially observable quantity. While it is possible to integrate the current going into the rover batteries and the current going out (i.e., Coloumb counting) or use voltage readings to estimate the amount of energy the batteries can provide, these types of estimates are generally imprecise, particularly when large variations in battery operating temperature are involved. Such variations are expected for the batteries of the VIPER rover, even if they are always kept within a safe operating temperature range. On Mars Exploration Rovers (Novak, Phillips, Birur, Sunada, and Pauken, 2003), for example, the batteries are generally maintained between $-20\,^\circ\mathrm{C}$ and $30\,^\circ\mathrm{C}$, a rather wide temperature range. The available battery capacity will also depend on the magnitude of the current being drawn.

Ice depth at the rover location is also a partially observable quantity. It is estimated from the payload instrument readings and from the theoretical ice depth distribution models, which indicate at what depth ice is likely to be at in a particular location, if it is there at all.

The checklist element of the state vector is a vector itself, serving as a record of which activities have been performed and how many times. This element is used, for example, for determining when the minimum mission success criteria have been achieved. The checklist is fully observable and specific to a use case.
6.5.2 Observation Vector

The observation vector currently used in the VIPER models has the form of \( o = (d_i, d_m, \text{soc}) \). The first element, \( d_i \) is a combined ice depth estimate provided by the payload instruments (NSS, NIRVSS, and the drill); the second, \( d_m \) is an estimate obtained from the theoretical ice depth maps; and the third, \( \text{soc} \) is an estimate of the battery state of charge, assumed to be derived from the battery’s terminal voltage readings. The vector is output by the VIPER generative model, described next.

6.5.3 Generative Model

This section provides information on the stochastic generative model \( G(s, a) \) used to output next system states \( s' \) and observation vectors \( o \) for VIPER. The overall model contains physics models of the rover subsystems, as well as execution models for its driving, recharging, and science activities. Parameter values used in the physics models are listed in Table B.1, including all of the necessary efficiency coefficients. Uncertainty models for power draw, battery recharge efficiency, and other quantities included in \( G(s, a) \) are described in Table B.2.

Battery Charging

The expected power produced by the three VIPER solar panels is given by:

\[
P_{\text{array}} = J V (1 - \zeta) A_{\text{pv}} \sum_{i=1}^{N_{\text{pv}}} \cos \phi_i \cos \theta_i,
\]

where \( J \) is the current density, \( V \) is the operating voltage, \( A_{\text{pv}} \) is the PV panel area (per unit, assuming all panels have the same area), \( \zeta \) is the expected ratio of PV panel area occlusion by lunar dust, \( N_{\text{pv}} = 3 \) is the number of solar panels in the VIPER array, and \( \phi_i \) and \( \theta_i \) are the azimuthal and elevation angles between panel \( i \) and the sun, respectively (calculated using the current date and time, as well as the rover location and pose). The amount of energy produced from the array is:

\[
\Delta E = \eta_c P_{\text{array}},
\]
where \( \eta_c \) is the coefficient of charge efficiency. The current density and the voltage are computed while taking into account the effect of temperature \( T \) on PV efficiency according to the following equations:

\[
J = J_0 - (T - T_0) \Delta J_T, \tag{6.3}
\]

\[
V = V_0 - (T - T_0) \Delta V_T, \tag{6.4}
\]

where \( J_0 \) and \( V_0 \) are the current density and voltage ratings of a PV cell, respectively, at maximum power at the beginning of the PV panel’s life; \( \Delta J_T \) and \( \Delta V_T \) are the current density and voltage increments per degree Celsius of a PV cell; \( T_0 \) is the ideal operating temperature for the PV panels; and \( T \) is the actual operating temperature. The current working assumption is that \( T = 80^\circ C \). In addition to this factored loss, each panel is expected to be 10% occluded by lunar dust. A PV panel degradation term is omitted in this work, as its effect over the expected life of the rover was estimated to be negligible.

**Driving and Science Activities**

The expected energy loss associated with driving per unit of time \( \Delta t \) is computed as:

\[
\Delta E = \eta_d \Delta t (P_d + P_{da}) + \Delta U, \tag{6.5}
\]

where

\[
P_d = mC_r + C_{gear} + vC_{motor} \tag{6.6}
\]

is the power expended on driving, \( P_{da} \) is the power drawn by science instruments active during the drive, \( \eta_d \) is the coefficient of battery discharge efficiency, and

\[
\Delta U = mg\Delta z \tag{6.7}
\]

is the change in potential energy. Driving activities power draw \( P_{da} \) depends on the exact instrument requirements of the activity sequence and is computed from the data in the activity dictionary. In Equation 6.6, \( C_r, C_{gear}, \) and \( C_{motor} \) are the coefficients of
rolling friction, gear friction, and friction losses within the motor, respectively, while \( v \) is the rover speed within the driving segment. In Equation 6.7, \( m \) is the mass of the rover, \( g = 1.62 \, \text{m s}^{-2} \) is the surface acceleration of lunar gravity, and \( \Delta z \) is the change in elevation over the traverse segment.

Driving segments are typically calculated in one-minute intervals and integrated. The power draw associated with waypoint activity sequences, like with driving activities, is computed based on the payload instruments used during each activity and the relevant information from the activity dictionary.

### 6.6 Traverse Evaluation and Refinement Use Case

Traverse Evaluation and Refinement (TER) is a mixed-initiative use case for planning VIPER traverses, where a traverse template is provided, specifying the science activities to be accomplished at an ordered set of waypoints. TER runs simulations according to the template, while introducing outcome and state estimation uncertainties that are likely to affect the execution of the actual VIPER mission. Simulations are done with SHERPA selecting durations of the rover’s battery recharge periods with the intent of maximizing the likelihood of successful traverse completion. Recharge periods, during which the rover is stationary, are scheduled before starting the sequence of science activities at a waypoint and before beginning the drive to the next waypoint. TER evaluates the traverse simulations using a set of metrics and outputs an overall robustness score for the family of traverses defined by the template. TER can also generate an optimized schedule of recharge periods to be used during the actual mission.

In the rest of this section, the model components specific to TER are presented first. Additional policies, designed for evaluating SHERPA’s performance on TER (and for use as value bounds estimators), are then listed. Finally, the metrics used for performance assessment are described.
6.6.1 Modeling

The TER use case builds on the common VIPER model components described in Section 6.5, including retaining the same sources and models of uncertainty as described in Table B.2. These sources of uncertainty result in significant variations between different simulations of a particular traverse. As an illustration, Figure 6.7 shows how widely the rover battery state of charge can vary across 30 different instantiations of the same traverse template. Model components that are specific to TER (actions, observations, and rewards) are discussed in the rest of the section.

![Figure 6.7: Stochasticity of the VIPER rover battery SoC across 30 instantiations of the same traverse (recharge durations are selected by LPDM)](image)

Figure 6.7: Stochasticity of the VIPER rover battery SoC across 30 instantiations of the same traverse (recharge durations are selected by LPDM)

**Actions**

TER actions are modeled as two classes of macro-actions: recharge-and-activities and recharge-and-drive. Actions in the first class instruct the rover to recharge its batteries for a specified length of time, then perform the science activities assigned to
the waypoint. Actions in the second class instruct the rover to recharge its battery (again, for a specified duration), then drive to the next waypoint.

Since the duration of actions in both classes is variable, they form action subspaces comprised of either actions with discrete durations (for use with DESPOT, for example) or actions with continuous durations (e.g., for LPDM). In the latter case, the AAS action sampling procedure for TER is essentially identical to that of Light Dark 1D (Algorithm 5.10), with the continuously valued recharge duration taking the place of the $x$ coordinate.

In the TER use case, action spaces are state-dependent. If a state indicates that all the activities at the current waypoint have been completed, only recharge-and-drive actions will be available in it. Otherwise, the action space will consist of recharge-and-activities actions. Descriptions of discrete recharge durations used in TER experiments are provided in Table 6.7.

**Observations**

Since for this use case the traverse waypoints are specified as inputs, estimating the ice depth at a particular location is not essential (it becomes important in the Traverse Synthesis use case, however, where locations for performing science activities are partially chosen based on ice depth estimates). The only element of the VIPER observation vector (Section 6.5.2) used in TER is $soc$, the battery state of charge. For solvers relying on discrete observations, $soc$ is represented by 10 values in the \{0.0, 0.1, \ldots, 1.0\} set.

**Observation Model**

In the current model, $soc$ uncertainty is modeled via a Gaussian distribution with the mean equal to the remaining energy value $e$ obtained from the “world” state and the variance of $0.2e$. 
CHAPTER 6. APPLICATION: SHERPA

Reward Model

The rover is rewarded for successfully completing the science activities at individual waypoints and for finishing the entire traverse. The rover is penalized — given negative rewards — if the state of charge (SoC) of the battery dips below the minimum set by flight rules or if DSN coverage is lost. A large penalty is assessed for mission failure, e.g., if the rover goes into a PSR and completely discharges its battery performing activities there, with no possibility of recharge. Specific reward values used in the TER experiments are given in Table 6.7.

6.6.2 Rule-Based Policies

The rule-based policies developed for the TER use case are designed to mimic strategies a human operator may adapt for scheduling rover recharge periods. The policies vary in their complexity, with the simplest charging the battery to full capacity at every decision point. The most complex policy performs a deterministic prediction of energy consumption during the next leg of the traverse and calculates the recharge time needed to obtain the same amount of energy (subject to the battery capacity limit). Descriptions of the policies are provided below:

**Full Charge**  
At every decision point, recharge to full battery capacity.

**PSR**  
At a decision point immediately following a PSR, recharge to 60% capacity or above. At all other decision points, recharge to 80% capacity or above.

**After Activities Recharge**  
Do not recharge before starting activities at a waypoint. After completing the activities, recharge to match the SoC before the activities. If the SoC before the start of activities was less than the minimum allowed SoC, recharge to match the minimum.
Look Ahead

Perform a deterministic simulation of the next traverse segment using \( G(s, a) \) to estimate the amount of energy consumed during it. Compute recharge duration sufficient to maintain SoC above the allowed minimum throughout that segment of the traverse.

Zone Crossing

Follow the rules of the PSR policy and also charge to 100% before driving a traverse segment that crosses a boundary between ice depth zones.

Note that all the rule-based policies will set the recharge duration to zero in the absence of sunlight and all obey the flight rule requiring the battery to be charged to 100% of capacity before entering a PSR. Additionally, a DSN drop-out always results in waiting until the connection is resumed, regardless of the wait time calculated by the policy.

6.6.3 Prognostic Policy

In addition to the rule-based policies, a prognostic policy was also implemented for the purposes of comparison. It follows the pattern first described in Chapter 3, Example 3.4. The best action \( a^* \) in a belief state \( b \) is selected as

\[
a^* = \arg \max_{a \in A(b)} E[\text{RUL}(b, a)],
\]

where \( E[\text{RUL}(b, a)] \), i.e., the expected remaining useful life given that action \( a \) is taken in \( b \), is estimated using \( G(s, a) \) for mission simulations up to a horizon \( H \). Battery recharge durations are selected randomly during the simulations. The prognostic policy uses a weighted-particle representation of beliefs (same as LPDM), with the particle states serving as the different initial simulation states. \( Q(b, a) \) for all \( a \in A(b) \) are computed as weighted sums of the cumulative rewards obtained during the simulations.
6.6.4 Value Bound Algorithms

The upper bound algorithm for a belief state $b$ value simply assumes that the rest of the traverse starting from $b$ is completed successfully, collecting all available rewards and incurring no penalties. The lower bound algorithm is designed to use any of the rule-based policies described in the previous section to simulate the rest of the traverse from $b$ and compute the cumulative reward. After an initial performance assessment of the rule-based policies, After Activities Recharge was selected as the default lower bound policy due to its consistent performance on a variety of traverse templates.

6.6.5 Metrics

A set of metrics was designed to evaluate the potential success rate of traverses based on a particular traverse pattern, as well as their robustness to action outcome and state estimation uncertainties. The same metrics were also used to evaluate performance of different policies in Section 6.7 experiments. The metrics are described in Table 6.6. Note that some of the metrics are computed only for fully successful or only for partially successful traverses.

6.7 Experiments

Three sets of experiments were conducted to evaluate the performance of SHERPA (with LPDM as its policy generator) on the TER use case. The first set of experiments was done with scenarios involving only recoverable system faults and delays (Section 6.3.5). The second set included scenarios with a fault that could not be completely recovered from, but could still be accommodated to let the mission continue. Finally, the third set was designed to assess SHERPA’s performance in the presence of model uncertainty in a manner similar to the LPDM model uncertainty experiments discussed in Chapter 5. In order to provide a common basis for comparison, all of the experiments were conducted using the same Hermite A region traverse template designed and verified by a human expert.
Table 6.6: SHERPA TER performance metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Computed for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully successful traverses, %</td>
<td>Percentage of traverse executions where all waypoints were completed.</td>
<td>N/A</td>
</tr>
<tr>
<td>Visited waypoints</td>
<td>The number of waypoints for which all science activities have been completed</td>
<td>Partially successful traverses</td>
</tr>
<tr>
<td>Reward, $R/R_{max}$</td>
<td>Average cumulative reward accrued, normalized by the maximum possible reward $R_{max}$ for the traverse. $R_{max}$ is computed by summing all positive rewards on the traverse and ignoring potential penalties.</td>
<td>Fully and partially successful traverses</td>
</tr>
<tr>
<td>Duration, hours</td>
<td>Average traverse, computed separately for fully successful and partially successful traverses</td>
<td>Fully and partially successful traverses</td>
</tr>
<tr>
<td>Duration vs. nominal</td>
<td>The ratio of the actual duration of the execution to the nominal duration of the traverse. The nominal duration is the total distance of the traverse divided by the baseline effective speed of the rover, plus the expected duration of all waypoint activities.</td>
<td>Fully successful traverses</td>
</tr>
</tbody>
</table>

### 6.7.1 Setup

Parameters shared among the three sets of experiments are summarized in Table 6.7. The benchmarking experiments involved all of the rule-based policies, the prognostic policy, three DESPOT configurations, and two LPDM configurations (referred to collectively as the policies). Just as in the Light-Dark experiments in Chapter 5, DESPOT was configured with discrete observations and either a standard discrete, an extended discrete, or continuous Blind Value action set. LPDM used the AAS algorithm for handling action spaces and was configured with either discrete observations or the TCC algorithm.
Table 6.7: Common parameters for the SHERPA TER experiments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tests per scenario</td>
<td>100</td>
</tr>
<tr>
<td>Planning horizon, $H$</td>
<td>50</td>
</tr>
<tr>
<td>$K$ (number of $b_0$ particles)</td>
<td>25</td>
</tr>
<tr>
<td>Max tree-building trials</td>
<td>1000</td>
</tr>
<tr>
<td>Recharge duration range, min</td>
<td>$[0, 300]$</td>
</tr>
<tr>
<td>Standard discrete $</td>
<td>A</td>
</tr>
<tr>
<td>Extended discrete $</td>
<td>A</td>
</tr>
<tr>
<td>Max $</td>
<td>A</td>
</tr>
<tr>
<td>$O$ bins (AR-DESPOT) and max $O$ clusters (LPDM)</td>
<td>10</td>
</tr>
<tr>
<td>Waypoint reward, $r_{wp}$</td>
<td>200.0</td>
</tr>
<tr>
<td>Success reward, $r_{success}$</td>
<td>10000.0</td>
</tr>
<tr>
<td>Low battery penalty, $r_e$, per $dt$ spent at off-nominal</td>
<td>$-5.0$</td>
</tr>
<tr>
<td>DSN drop penalty, $r_{dsn}$, per $dt$ spent at off-nominal</td>
<td>$-1.0$</td>
</tr>
<tr>
<td>Failure penalty, $r_{failure}$</td>
<td>$-10000.0$</td>
</tr>
</tbody>
</table>

A particle-based representation was used for the “world” simulation belief state. The rule-based policies used the highest probability particle as their current state estimate, while the rest of the policies used the full belief. A simulation was considered failed if either the true “world” state or the most probable belief particle were determined to be failure states.

The traverse template specified the starting location in an area north-east of Hermite A crater and the target start time of September 17, 2021 at 02:16 UTC. The actual start time was subject to stochasticity, as described in Table B.2. Solar illumination and DSN coverage maps were provided for every two-hour interval up to 90 Earth days from the start time of the traverse. The traverse consisted of 25 waypoints (not counting the starting location), with science activity sequences
specified for each waypoint.

An execution was considered successful if science activities at all the waypoints were completed. The objective of decision making for all policies was to select battery recharge durations, as defined for TER in Section 6.6. The battery was recharged before performing the science activity sequence at a waypoint and also before starting the drive to the next waypoint, i.e., 50 sequential decisions per mission scenario. Rover speed was considered constant throughout the traverse. Experiment design details specific to each of the three sets are provided next.

**Experiments with Recoverable Faults and Delays**

As discussed in Section 6.3.5, recoverable faults are assumed to leave no lasting effects on vehicle health and may only result in a greater than expected power consumption or a time delay. Time delays may occur for other reasons as well, such as those caused by obstacle avoidance or by waiting for the next set of commands from Earth. Due to the dynamic nature of solar illumination and DSN coverage in the lunar polar regions, even moderate delays may require significant changes to operational plans.

This set of experiments represents the nominal case of VIPER operations and so the standard setup described earlier was used. A shorthand *nominal scenarios* is occasionally used through the rest of the chapter for convenience.

**Fault Accommodation Experiments**

In order to test SHERPA’s capability of dealing with faults that cannot be completely resolved, yet can be accommodated, a scenario was developed where one of the rover motors develops an increased friction fault. In the scenario, the fault is injected shortly after the departure from the starting waypoint. An assumption is made that maintaining the previously planned rover speed is desirable, as that would cause the least amount of disruption from the operational point of view. In order to maintain the same speed, however, more current is drawn into the affected motor. This results in a faster battery discharge and temperature rise inside the motor. If the temperature exceeds the maximum rated motor windings temperature $T_{w,max}$, winding insulation
may fail and the motor may experience a winding short fault.

For this set of experiments, overheating of the motor was treated as an event requiring emergency response (Section 4.3), analogously to losing DSN coverage. If $T_w$ reached $T_{w,max}$, control was handed over to a $\pi_{SER}$ which commanded the rover to stop and wait until the motor cooled down to at least $T_{w,n}$ — the upper limit of the nominal operating temperature range. Once the cool-down was completed, control was handed back to the decision making policy.

Fault accommodation experiments required several modifications to the general VIPER models, described next. While no changes were made to the solvers or rule-based policies themselves, all of them had access to the new models.

**State vector**  The state vector was augmented with vectors of motor winding temperatures and motor friction coefficients for each motor: $(T_{w,1}, T_{w,2}, T_{w,3}, T_{w,4})$ and $(C_{f,1}, C_{f,2}, C_{f,3}, C_{f,4})$, respectively. Motor winding temperatures $T_w$ were considered fully observable, while motor friction coefficients $C_f$ — partially observable (whenever motor indices are omitted in the rest of the description, applicability to any one of the motors should be assumed). $C_f$ values were normalized, with 1.0 defined as the nominal value. These values were used as multipliers for the values of current drawn by individual motors, i.e., a motor with a higher $C_f$ draws proportionally more current.

**Observation vector**  A vector of (noisy) motor current readings $(I_{m,1}, I_{m,2}, I_{m,3}, I_{m,4})$ was added to the observation vector. These were used to estimate $C_f$ values in particular.

**Uncertainties**  Uncertainty of the current going through a drive motor was modeled as a Gaussian distribution, with the mean equal to the expected current value $E(I_m)$ and the variance $\sigma^2 = 0.2E(I_m)$. 
The system simulator \( G(s,a) \) was augmented with thermal models of the drive motors. As the design for the mobility system of the VIPER rover was in its early stages when this set of experiments was being formulated, an assumption was made that the rover motors will be of the same class as those on the similarly sized Mars Spirit and Opportunity rovers. Those rovers relied on Maxon EC 32 drive motors (Lindemann and Voorhees, 2005); motor parameters relevant to this set of experiments are provided in Appendix B (Table B.3). The rest of the key parameters used in the new model components are also listed in Table B.3. The rated \( T_{w,max} \) for the Maxon EC 32 is 100.0\(^\circ\)C, but was assumed to be 85.0\(^\circ\)C in the experiments to account for a safety margin. The thermal state of a motor’s windings was modeled with the following equation:

\[
\frac{dT_w}{dt} = \frac{1}{C_{m,t}} \left( R_w I_m^2 + h_m (T_{m,h} - T_w) \right) dt,
\]

(6.9)

where \( R_w \) is the electrical resistance of the motor windings, \( C_{m,t} \) is the thermal inertia coefficient of the motor, \( h_m \) is the thermal transfer coefficient of the motor, \( I_m \) is the motor current, \( T_w \) is the motor windings temperature, and \( T_{m,h} \) is the motor housing temperature. \( T_{m,h} \) depends on whether the rover is operating in a sunlit area or in sun shadow (Table B.3). A modeling assumption was made that the VIPER thermal control system (consisting of both passive and active elements) is designed to keep the motors within the nominal operating temperature range (\(-40^\circ\)C to 60\(^\circ\)C), given the motor current \( I_{m,c} \) corresponding to the standard cruise speed on flat terrain. Under these conditions, \( dT \approx 0 \). Assuming sunlit operations, \( h_m \) was estimated as

\[
h_m \approx \frac{R_w I_{m,c}^2}{T_{w,n} - T_{m,\text{sun}}}.
\]

(6.10)
The increased friction fault was injected into Motor 3 by changing $C_{f,3}$ from 1.0 to 2.0 in the “world” simulation of the rover. That, in turn, increased the current drawn by the affected motor two-fold.

Model Uncertainty Experiments

In the model uncertainty experiments, the probability distribution governing the stochasticity of the power draw model in the baseline VIPER models (Section 6.5) was altered in order to assess the effects on the decision making performance. This stochasticity is normally represented by a Gaussian distribution with the variance of $\sigma^2 = \xi \mu$, where $\mu$ is the mean value and $\xi = 0.2$ (Table B.2). In the experiments, the variance was progressively increased using $\xi \in \{0.3, 0.4, \ldots, 1.0\}$.

As in the similar Light-Dark experiments in Chapter 5, the best-performing versions of DESPOT and LPDM were used: DESPOT with an extended action set and LPDM with both AAS and TCC enabled. No changes to the solver parameter values from those listed in Table 6.7 were made.

6.7.2 Results

Results from all three sets of SHERPA experiments are provided in Appendix C. This section presents an analysis of these results and, where merited, highlights their relevance to the key concepts of the preceding chapters.

Experiments with Recoverable Faults and Delays

Results of the experiments with recoverable faults and delays performed with the standard set of parameters are provided in Table C.1. It can be observed that some of the rule-based policies performed rather well. For instance, the After Activities policy was fully successful on 82% of executions and even when falling short, still managed to accomplish a high number of waypoints. The PSR and Look Ahead policies also did relatively well. It is useful, however, to take a closer look at the results for the two worst-performing policies: Full Charge and Prognostic.
That the Full Charge policy performed so poorly (no traverses completed successfully) may appear surprising at first. Taking complete advantage of every recharge opportunity would seem like a sound (if somewhat conservative) strategy. However, this result correlates well with the analysis presented in Section 3.3.1 and provides another, now more complex example of the issue of conflict between SHM and DM objectives.

Here, as in Example 3.2, the policy essentially implements a separated SHM function that is focused on restoring system health (battery charge) to nominal at the earliest opportunity. Also as in Example 3.2, this arrangement leads to suboptimal operational performance. By always waiting to recharge to full battery capacity, the rover got “trapped” in the sun and DSN shadows in every execution scenario (particularly in the second half of the traverse), never completing all 25 waypoints.

In the case of the Prognostic policy, the unimpressive performance (also no successful traverse completions) is attributable to the random policy used during the prognostic traverse simulations. Even if one of the action options for the current belief state was a better choice than the rest, its advantages were lost in the mostly uninformative estimates of the remaining useful life produced by picking all the subsequent actions up to the decision horizon randomly. Therefore, the overall policy essentially became random as well. With the maximum decision horizon of 50 steps, this problem was particularly acute in the earlier parts of the traverse.

All POMDP policies performed substantially better than the rule-based policies and the Prognostic policy. Within the POMDP group, LPDM performed better than the three DESPOT configurations, however the difference attributable to TCC appeared to be slight. Similarly to Light-Dark 1D in Chapter 5, this nominal set of scenarios (with sufficiently large margins of error) was a manageable challenge for the discrete representations of the observation space. The same, however, will not be the case in the fault accommodation experiments.

The set of experiments with the POMDP solvers performed with 100-particle beliefs (Table C.2) shows a clear, across-the-board performance improvement for all the solvers. Analysis of the data from the experiments revealed that this improvement was primarily due to the fact both DESPOT and LPDM use the particles in $b_0$ as
the starting states for tree-building scenarios. With four times as many particles, four times as many scenarios were simulated for each tree construction trial (the total number of trials remained the same). This typically led to smaller uncertainty gaps at the root \((b_0)\) nodes and, consequently, to better action selections. Of course this improvement was accompanied by an approximately four-fold increase in the average decision time.

**Fault Accommodation Experiments**

Results of the fault accommodation experiments are summarized in Table C.3. The most evident difference from the nominal scenarios is that the success rate for all rule-based policies and the Prognostic policy was zero, with only a few waypoints accomplished in each traverse. These failures did not occur due to the motor failure, as the automatically initiated cool-down periods prevented its overheating. However, these cool-down periods, combined with the policy-commanded recharge periods, delayed the rover enough within the first few traverse segments that it invariably got trapped in the sun shadows, DSN shadows, or both (the recharge periods were also somewhat longer on average than in the nominal scenarios, due to the increased current draw).

The prognostic policy also performed poorly and for the same reasons as discussed for the previous set of experiments. The random simulation policy — used to estimate the remaining useful life of the rover for each of the recharge duration options — obscured the advantages any particular option may have over the others. The problem, once again, manifested itself most prominently in the beginning of the traverse, where simulations were executed with a decision horizon at or near 50. Having less margin for error in this set of experiments than in the nominal set, the prognostic policy was generally unable to get the rover past the first few waypoints.

Both DESPOT and LPDM largely avoided becoming trapped in sun/DSN shadows at the beginning of the traverse. While some of the executions did fail in a similar region towards the end of the mission, that meant that a significant portion of the mission goals was nevertheless accomplished. A review of execution records showed that the POMDP policies succeeded on a larger number of scenarios in part
by selecting shorter recharge durations before driving to the next waypoint than the other policies. While seemingly counter-intuitive, the benefits of this strategy became apparent upon further analysis.

Arriving at a waypoint, the rover would generally perform a pre-activities recharge, then spend a considerable amount of time performing stationary science activities. There would then be another recharge performed before driving to the next waypoint. In every case, this sequence was sufficient to cool down the affected motor, so no additional cool-down time was required before starting the next traverse segment. Whereas in a nominal scenario, the rover would typically drive the entire segment non-stop (unless paused due to a dropped DSN connection), in the fault injected scenarios one or more cool-down periods would be required. These would also serve as additional recharge periods, therefore the pre-drive recharge period could be shortened to a duration just long enough to enable reaching the cool-down stop with a sufficient remaining battery charge. The shortened pre-drive recharges, in turn, helped the rover maintain a sufficient pace to avoid getting trapped in sun or DSN shadows.

While all POMDP solvers executed a version of the above strategy, in most cases LPDM (AAS, TCC) was able to execute it better. With the smaller safety margins than in the nominal scenarios, being able to estimate the state of battery charge more accurately and select recharge durations with more precision proved to be an advantage.

Model Uncertainty Experiments

The results of the Model Uncertainty Experiments for DESPOT and LPDM are provided in Tables C.4 and C.5, respectively. As in the Light-Dark experiments in Chapter 5, degradation of decision making performance can be observed for both solvers, although DESPOT again exhibited a higher rate of decline.

Interestingly, even when not able to complete an entire traverse at the higher model uncertainty settings, both LPDM and DESPOT accomplished approximately the same number of waypoints as the Full Charge policy or the prognostic policy did on the nominal scenarios. In the case of Full Charge the lackluster performance is the result of a poor match between the relatively simple rules in the policy and
the challenges of the test traverse. In the case of LPDM and DESPOT, however, as model uncertainty increases, the policies become more and more random. In that, they essentially approximate the performance of the prognostic policy. Thus the results appear to confirm that for the traverse template used in the experiments, an average of 20–22 waypoints is the range achievable on nominal scenarios with a near-random policy.

Another interesting trend that can be observed in the results is that as the percentage of successful traverses decreased with higher model uncertainty, so did the average duration of successful traverses. It appears that only during the traverses where the optimistic selections of short recharge durations happened to coincide with favorable execution conditions (e.g., low power consumption, short duration of science activities, no DSN-related delays) did the rover manage to complete all 25 waypoints.

6.8 Discussion

This chapter introduced SHERPA, a new decision support system for robotic space exploration missions and described its first application — the upcoming VIPER rover mission to the polar regions of the Moon. The mission presents several interesting challenges for mission planners, including the objective to operate through several lunar nights, as well as the highly dynamic nature of solar illumination and communication coverage near the lunar poles.

The chapter described one SHERPA use case developed for VIPER in detail. The use case, Traverse Evaluation and Refinement (TER), is intended for mixed-initiative rover traverse planning. A traverse template (science activities to be performed at an ordered set of waypoints) is provided as one of the inputs. Other inputs include different types of digital maps and a set of mission rules. TER evaluates the robustness of the proposed traverse by running simulations with state and action outcome uncertainty, while selecting durations of recharge periods that maximize the chances of the rover completing the traverse. The use case can also produce an optimized schedule of recharges to be used during the actual mission.
In order to evaluate the performance of SHERPA on TER, several rule-based policies were created that emulate potential operational strategies of human controllers. One of the policies, Full Charge, resembles a separated DM/SHM formulation discussed in detail in Chapter 3. In such a formulation, SHM is focused on restoration of system health at the earliest opportunity. Another policy, Prognostic, also has a connection to the discussion in Chapter 3. It reproduces the approach adopted by some contemporary SHM systems where decisions are made in part based on predictions of the system’s remaining useful life.

Performance of the rule-based policies, the prognostic policy, and the online POMDP policies generated by several configurations of DESPOT and LPDM was compared on three sets of experiments. The first set utilized scenarios with recoverable faults and delays (nominal), the second tested accommodation of faults that cannot be fully resolved, and the third consisted of model uncertainty experiments (involving DESPOT and LPDM only).

Some rule-based policies performed well on the nominal scenarios, completing a high percentage of simulated traverses or a significant number of waypoints even if not fully successful. The Full Charge policy fared poorly, however, highlighting one of the key issues with separated DM/SHM formulations: the difficulties in balancing potentially conflicting objectives of the two components. In charging the battery to its full capacity at every available opportunity, the policy did not account for the operational consequences this would entail. As a result, the simulated rover did not complete any of the traverses successfully, getting stranded in sun and DSN shadows along the way.

The Prognostic policy also did not do well, albeit for a different reason. The issue the policy encountered was first discussed in Section 3.4.3. Prognostic algorithms rely on forward simulations of the system (rollouts) to determine its remaining useful life. This approach work in situations where the degradation processes affecting system health are decoupled from the actions controlling the behavior of the system (the associated practical difficulties are discussed in Section 3.4.2). In a system where control actions may affect the state of its health (such as the VIPER rover), making prognostic predictions requires knowledge of future actions. In systems operating under
state and outcome uncertainty, there are three options for obtaining this knowledge: (1) by assuming a fixed plan, which could quickly become obsolete; (2) by relying on Monte Carlo simulations that select future actions randomly, i.e., using a random policy; or (3) by obtaining or computing a policy capable of selecting actions better than a random one.

Prognostic algorithms use either the first or the second option, as pursuing the third option would essentially require them to be transformed into algorithms for decision making under uncertainty, such as DESPOT or LPDM (limitations of simply using a policy intended for operational purposes are discussed in Section 3.4.3). Lacking a predetermined plan, the prognostic algorithm used in this work relied on the second option: a random rollout policy. Such a policy, by assuming that all actions are equally likely, may produce uninformative remaining useful life estimates (particularly for a complex system operating in a dynamic environment, such as the VIPER rover). Figure 6.7, for instance, illustrates how much variability may be present in execution of a traverse scenario due to the uncertainty and the action choices made.

On the nominal scenarios, the Prognostic policy performed similarly to the Full Charge policy: did not finish any of the traverses, although did manage to complete a reasonably high average number of waypoints. However, in the more challenging fault accommodation experiments, where one of the drive motors was injected with an increased friction fault, the Prognostic policy completed only a few initial waypoints per simulation. The rule-based policies did not do much better in this set of experiments either, unable to account for the need to periodically stop and cool the affected motor.

The two POMDP solvers, DESPOT and LPDM, generally performed well in both sets of experiments. As in the experiments conducted on the abstract Light-Dark problems in Chapter 5, the advantages of LPDM (and its ability to handle complex action and observation spaces) became more evident in the more demanding experiments of the fault accommodation set.

A subset of nominal experiments also followed up on the computational complexity analysis presented in Chapter 4, demonstrating how the accuracy of approximating an optimal policy $\pi^*$ by DESPOT or LPDM may be improved by increasing $K$, the
number of pre-sampled scenarios used in policy tree construction ($K$ also happens to be the number of particles used in the belief state representation). Increasing $K$ from 25 particles to 100 particles showed a clear performance improvement for both solvers, although at a correspondingly higher computational cost.

In the third experiment set, focused on model uncertainty, the same general trend could be observed as in the similar experiments in Chapter 5. Performance of both DESPOT and LPDM degrades as uncertainty increases, but with LPDM being able to maintain a lower degradation rate. Another notable observation resulting from this set of experiments is that at the higher levels of model uncertainty, when informed decision making by the two POMDP solvers became difficult, their results started matching that of the Prognostic policy. This served to confirm the level of performance achievable with a random (or near-random) policy on the nominal scenarios.

In addition to Traverse Evaluation and Refinement, other use cases are currently under development for the VIPER mission. Traverse Synthesis will generate rover traverses using only high-level mission requirements and constraints. It also serves as the foundation for two additional use cases: Landing Site Selection and Vehicle Parameter Optimization. These use cases, along with other ongoing and planned work on SHERPA, are described in the next, final chapter of the thesis.
Chapter 7

Conclusions

The original motivation for the work in this thesis was to explore ways of taking system health into account when making operational decisions in complex systems, particularly those in the aerospace domain. It grew into a more extensive reexamination of the prevalent approaches to system health management and led to the formulation of an approach that unifies it with operational decision making. The approach and the algorithms developed to enable it were subsequently put into practice in a decision support system developed for the upcoming NASA robotic rover mission to the Moon, as well as for other future space exploration missions. This chapter provides a summary of the thesis and a review of its main contributions, describes ongoing work, outlines plans for future efforts, then concludes with brief final remarks.

7.1 Summary and Contributions

The summary of the thesis and the review of its contributions provided in this section are, once again, organized into three parts: theory, algorithms, and application. The first part covers the material of Chapters 3 and 4, while the second and third do that for Chapters 5 and 6, respectively.
7.1.1 Theory

Complex dynamic systems, particularly in the aerospace domain, require two main functions for successful operation: (1) decision making (DM), encompassing planning, scheduling, and control and (2) system health management (SHM). The latter has typically included fault detection, fault identification/diagnosis, prognosis, and mitigation/recovery.

The first contribution of this work is reexamining the currently prevalent approach to system health management (thatformulates it as a separate problem from operational decision making) and identifying its limitations. In particular, two general limitations were found. The first is that separated formulations can result in conflicting SHM and DM objectives. The second is that such formulations can be ineffective due to mutually inaccessible model spaces. In addition to these general limitations, the thesis also showed why prognostics is not meaningful for most types of applications and is difficult to implement in an effective manner in the rest.

The second contribution of the thesis is formulating a unified approach to DM and SHM, Health Aware Decision Making (HADM), that overcomes the above limitations. In it the two problems are combined through the utility theory and state-space modeling. The thesis outlines how existing system health management concepts map into HADM and also discusses the distinction between HADM and system emergency response, proposing that the latter should be retained as a separate function. The thesis also provides a computational complexity analysis of HADM, determining that while HADM is expected to be a more effective approach than what is done currently, it may result in an increased computational cost in certain types of applications.

7.1.2 Algorithms

One category of problems where HADM computational cost increases may be a factor are the problems with uncertainty present in action outcomes and system state estimation. In complex realistic systems the degree of difficulty increases further when large, high-dimensional state, action, and observation spaces need to accommodated.
Recognizing that a significant body work focused on handling large, high-dimensional state spaces already existed, this thesis focused instead on developing better methods for handling action and observation spaces, specifically in problems formulated as partially observable Markov decision processes (POMDPs).

The third contribution of this thesis consists of:

- Developing a new methodology for handling large, high-dimensional action spaces in online POMDP solvers on the basis of global black-box optimization and providing a specific implementation, Adaptive Action Selection, patterned on Fast Simulated Annealing;
- Developing a new procedure for processing observations within online POMDP solvers, Transition Correlation Clustering (TCC);
- Combining AAS and TCC with an existing state-of-the-art online POMDP solver capable of handling large state spaces (DESPOT) to create the Large Problem Decision Making (LPDM) solver.

One of the key features of LPDM is the ability to recognize when two dissimilar observations are pointing to the same underlying system state, which is particularly useful for reasoning about system health. LPDM was benchmarked against DESPOT in a series of experiments and generally showed a superior performance, especially as the degree of problem difficulty increased.

7.1.3 Application

The fourth contribution of the thesis is creating a new decision support system for robotic planetary missions: System Health Enabled Realtime Planning Advisor (SHERPA). SHERPA is built on the foundation of the HADM approach and uses the new LPDM POMDP solver as its primary reasoning engine. SHERPA is being utilized in the development of NASA’s Volatiles Investigating Polar Exploration Rover (VIPER) mission and thus may become the first decision support system capable of formal reasoning under uncertainty to be used in a space application.
VIPER is currently scheduled to launch in December of 2023. The mission aims to explore one of the polar regions on the Moon, identifying and characterizing deposits of water ice and other volatiles. SHERPA currently implements the Traverse Evaluation and Refinement use case, where it takes an ordered set of scientific waypoints and activities associated with them as an input and evaluates the robustness of the resulting family of traverses to possible delays and malfunctions. SHERPA does so by intelligently executing numerous mission simulations, aiming for optimal selections of battery recharge periods. After the evaluation is complete, SHERPA can output recommendations for the durations of recharge periods throughout the traverse route that maximize the chances of successful completion of the mission objectives.

Other VIPER use cases are also currently in development, including Traverse Synthesis. In Traverse Synthesis only the high-level mission objectives and constraints are supplied as inputs to SHERPA, which then automatically generates full traverses for the rover that are, again, computed to be as robust as possible to delays and malfunctions. While the development of SHERPA was motivated by the VIPER mission (and its predecessor, Resource Prospector), the model-based architecture of the system is intended to easily adapt to future missions (including repurposing of use cases).

7.2 Ongoing Work

Ongoing work (both on the algorithm side and on SHERPA) is motivated by the needs of the VIPER mission. The LPDM solver is being enhanced to provide better support for mixed observability Markov decision processes (MOMDPs) and generate contingent plans from policy trees. A deep learning approach is being explored for use in belief value bounds estimation. In SHERPA, a number of new use cases is being implemented. This section explains the need for these enhancements and sketches out some of their implementation details.
7.2.1 Algorithms

While LPDM has so far shown itself to be a capable POMDP solver, the work on it is by no means complete. The first two enhancements described below are aimed at improving the accuracy of its decision making. Speeding up value bounds computations will allow for execution of a larger number of tree-building simulations within the same period of time, leading to better approximations of an optimal policy. Full support for MOMDP formulations is also expected to result in performance improvements, as well as in more accurate belief state approximations within an LPDM policy tree. The final enhancement, to enable automated generation of contingent plans, will help with expanding LPDM’s use in risk-sensitive applications.

Belief State Value Estimation with Deep Learning

As described in Chapter 6, online rule-based policy algorithms are currently being used in SHERPA for belief state value bounds estimation. While these algorithms are designed to be as efficient as possible, their computations still involve a certain degree of mission simulation. Invoked numerous times during the LPDM tree construction process, the computational resources consumed by the bounds estimation algorithms constitute a significant proportion of the overall amount.

In a joint effort with Sarah Feng (UC Berkeley), the use of recurrent deep neural networks for belief state value bounds estimation is being investigated. The early results show promise; however the work is far from being completed. If successful, this approach would enable an almost instantaneous value bounds estimation for belief states, improving the overall performance of LPDM. If trained as a policy approximator, producing action recommendations rather than value bounds, a deep neural network can also be used for performance benchmarking.

Support for Mixed Observability Markov Decision Processes

As described in the previous chapter, some of the elements in the VIPER state vector are considered fully observable, while others are partially observable. A basic support for distinguishing between the two types while performing belief updates in
the “world” simulations is currently implemented. The distinction is important in ensuring that while the fully observable elements are still subject to action outcome uncertainty, state transitions for them are performed identically so as to not introduce unwarranted state estimation uncertainty. For instance, while the exact distance the rover moves upon completing a drive command may be subject to some stochasticity, the $x$ and $y$ coordinates for all the particles in the current belief state should be updated in an exactly the same manner, as the new $x'$ and $y'$ coordinates are fully observable.

Implementing full support for MOMDP formulations in LPDM is expected to not only result in more accurate belief state approximations for problems where such a formulation is appropriate, but also improve the overall performance. As discussed in Chapter 2, with beliefs no longer needed to be maintained over the entire state vector, the overall computational complexity may be reduced substantially due to the lower dimensionality of the belief space.

Support for mixed observability in LPDM is being implemented by introducing provisions for factored state vectors, as well as by adopting a more sophisticated action structure. The latter will allow the same action variable to be used in either stochastic or pseudo-deterministic manner within the generative model, as described above.

**Generation of Contingent Plans from Policy Trees**

A contingent plan is a traditional fixed plan $\{a_1, a_2, \ldots, a_n\}$ augmented with one or more branches that are taken if certain conditions are satisfied during the execution of the main plan. A policy $\pi$ can be viewed as an extreme case of a contingency plan, where the number of “branches” that can be followed depending on the states/beliefs encountered during the execution may often be infinite.

In risk-sensitive applications such as space missions, it may be difficult to show with a high degree of certainty that following a policy (especially a complex online policy) will not lead to undesirable outcomes. That is why using fixed plans with a limited number of contingency branches (plans that can be thoroughly verified a
priori) has been the standard approach on most space missions. The current concept of operations for the VIPER rover also calls for a contingent plan with limited branching, at least for the primary mission phase.

A procedure to automatically generate contingent plans from LPDM policy trees is currently being implemented. It consists of the following high-level steps:

1. A set of branching conditions is defined (for instance, that a contingency branch should be taken at Waypoint 5 if the rover is more than 10 hours behind the expected time of departure from that waypoint).

2. The primary execution trajectory is traced out through the tree (starting at the root node $b_0$) by taking best action branches at belief nodes and taking observation branches with the highest cumulative probability mass in the associated transition clusters at action chance nodes.

3. Siblings of belief nodes along the primary trajectory are then checked against the set of branching conditions. If, for instance, a sibling belief node $b^+$ exists where the highest probability state particle indicates that (a) the rover is currently at Waypoint 5 and (b) that it is more than 10 hours behind the primary trajectory timeline, then $b^+$ is designated as a branching point. Step 2 is then performed for the subtree with the root at $b^+$ to trace out a contingency trajectory (contingency trajectories do not have branches, so Step 3 is only done for the primary trajectory).

### 7.2.2 Application

With the development of the baseline version of the Traverse Refinement use case largely finished, the main focus currently is on completing the Traverse Synthesis use case. In addition to assisting mission planners and operators with rover traverse design, Traverse Synthesis is intended to serve as the foundation for several additional use cases, two of which — Landing Site Selection and Vehicle Parameter Optimization — are also briefly described below.
Traverse Synthesis Use Case

Under the Traverse Synthesis (TS) concept of operation, SHERPA takes high level mission goals and constraints as inputs and generates families of traverses that satisfy both. Some of the constraints are the same as those described for the TER use case (e.g., on the battery charge level, on operations without communication coverage), while additional ones (e.g., on terrain slope) are also introduced. An example of a set of mission goals is that the rover needs to visit all four ice depth zones of interest (dry, shallow, deep, and PSR) and perform near-surface assay and volatiles analysis sequences of tasks in several locations of each zone. The POMDP action space includes drive commands with arbitrary headings, recharge periods of variable duration, and scientific tasks. The mission is considered to be successfully completed when all of the goals are accomplished.

A rule-based policy, Nearest Zone Policy (NZP), is used by LPDM for value bounds estimation. The policy is also used for benchmarking purposes, similarly to the TER rule-based policies. NZP implements the following set of rules:

- If there are tasks remaining in the current zone, complete them.
- If no tasks remain to be completed in the current zone, search for the nearest unvisited zone type and take the shortest path there.
- If the battery charge drops below 80% during a drive, stop and recharge.
- If sunlight is lost at the current location, finish any tasks in progress and start driving to the next destination.
- Recharge the battery to full before entering a PSR.
- If the DSN connection is lost, wait at the current location until the connection is restored.

In addition to NZP, traverses generated by human experts are being used for benchmarking purposes in the development of Traverse Synthesis. The TS use case is also intended to be eventually coupled with TER. If the TER use case executed on a
fully qualified “flight” traverse can be thought of as a minimum success phase of the VIPER mission, TS can evaluate the potential for an extended phase (and use that as an additional metric in evaluating candidate “minimum success” traverses). TS will start with the final TER state and, by synthesizing families of optimized “free-form” traverses, can inform how much additional scientific value can be obtained by extending the mission from that state.

The other TS-based use cases currently under development can be categorized as high-level optimization problems with TS serving in the role of an intelligent black-box mission simulator. Optimization based on complex black-box simulations is widely used, for instance, in the field of Multidisciplinary Design Optimization (Martins and Lambe, 2013). However, in most cases these simulations are uncontrolled and if there are actions to be selected within them, they are typically selected randomly. As mentioned earlier, the intent of the TS use case is to select actions as well or better than a human mission controller. Thus exploration of option space for these use cases is paired with optimized simulation capabilities. Two of the TS-based use cases are described next.

**Landing Site Selection Use Case**

In this use case TS is utilized to identify promising landing sites in the lunar polar regions. This is done by analyzing the expected scientific value achievable from each of the candidate locations during time periods of interest.

**Vehicle Parameter Optimization Use Case**

This use case is intended to assist in exploration of the VIPER rover design parameter space through the use of TS simulations. Examples of such parameters include the maximum battery capacity, number of cells in the solar panel array, and the maximum data transmission rate of the communication system. By varying desired parameters and executing sufficiently large numbers of TS simulations, the effects of parameter values on the scientific return of the mission can be analyzed and used in further trade studies that also bring cost, weight, and other types of metrics into consideration.
7.3 Future Work

Several new capabilities are under consideration for LPDM and SHERPA, both within the context of the VIPER mission and for other potential applications. LPDM has garnered interest for use in air traffic management and autonomous vehicle settings. Beyond VIPER, SHERPA is being discussed for use on lander missions, as well as for missions involving multiple robotic vehicles. This section describes the capabilities being considered and outlines possible approaches to their implementation.

7.3.1 Algorithms

The first two capabilities described are intended to improve LPDM’s efficiency in solving complex problems in the near term. The last two are the more ambitious efforts, intended both to enable the solver to play an expanded role in its current application in SHERPA and to prepare it for applications where interactivity and multi-agent support are important.

Observations Clustering Improvements

The correlation clustering approach adapted in this work for large, continuously valued observation spaces strives for balance between optimality of cluster assignments and algorithm performance. In the future, techniques developed in the field of correlation clustering (Bansal, Blum, and Chawla, 2004; Zimek, 2008) will be evaluated for applicability and possibly incorporated into LPDM. For instance, techniques described by Achtert, Böhm, Kriegel, Kröger, and Zimek (2007), Bonizzoni, Della Vedova, Dondi, and Jiang (2008), and Mathieu, Sankur, and Schudy (2010) appear promising in delivering clustering quality improvements without sacrificing real-time performance.

Parallelization

The capabilities of modern multi-core computer processors and GPUs make parallelization of computations an attractive option for improving performance of POMDP
solvers. Cai, Luo, Hsu, and Lee (2018) demonstrate the advantages of parallelizing some of the operations in DESPOT. Similarly, several opportunities for parallelizing computations exist in LPDM. These include value bounds estimation through use of heuristic policies, state transition generation by TCC and tree-building rollouts in general, as well as particle-based belief updating.

**Result Explanation Capabilities**

Explainable artificial intelligence is an emerging research area that aims to make solutions produced by artificial intelligence algorithms interpretable by humans and, in doing so, help build more confidence in them (Barredo Arrieta et al., 2020). It would be desirable to incorporate such capabilities into LPDM, given its use for decision support in SHERPA. The initial step may involve changing the current single-number representation of upper and lower bounds on node utility to a factorized form, e.g., a vector. This would allow bookkeeping of value in several categories of interest, e.g., multiple objectives or multiple types of system resources. The domain-specific generative model $G(s,a)$ would be required to output factorized rewards. Each element of a value bounds vector would be updated separately in the course of the value backup procedure following a tree-building simulation. The elements of a vector would only be combined when, for example, $\max_{a \in A(b)} \hat{Q}^*(b,a)$ needs to be determined for some belief $b$ during an in-tree traversal. Otherwise they can serve to explain the factors making a particular $\hat{Q}^*(b,a)$ the highest value for belief $b$.

The TCC algorithm within LPDM is also expected to prove useful in result explanation. Groupings of related, even if dissimilar, observations can assist in explaining an off-nominal mode, for example (as is already being done in many contemporary diagnostic systems). The focus on grouping highly-correlated states into beliefs should also make it easier to map beliefs to physical system modes.

**Extension to Multi-Agent Formulations**

While in principle the abilities of LPDM to handle complex action and observation spaces may allow it to solve multi-agent planning problems formulated as conventional
POMDPs, practical constraints on action and observation branching factors would likely limit this approach to the simplest multi-agent problems only. This is due to the fact that the dimensionality of joint action and observation spaces would increase proportionally to the number of agents.

Methods have been demonstrated, however, that exploit structure of multi-agent problems to make them computationally tractable. For instance, Amato and Oliehoek (2015) developed a POMCP-based solver that takes advantage of the locality of interactions among the agents in problems formulated as MPOMDPs. That enables a decomposition of the value function into a set of overlapping factors, which can significantly improve the efficiency of policy computation. The initial class of multi-agent problems that SHERPA is being considered for involves coordinated exploration of the lunar surface by multiple robotic vehicles—essentially a multi-agent version of the VIPER problem. Given that the vehicles are expected to have a networked ability to communicate with one another, an MPOMDP formulation may be sufficient. The factored statistics technique implemented by Amato and Oliehoek to approximate a factored value function is similar to what is already being planned for supporting results explanation in LPDM. Adding support for factorized policy trees, as Amato and Oliehoek do to manage joint observation spaces, is also being considered.

Should centralized multi-agent formulations prove to be too limiting, Dec-POMDP support may also be added. Generally, Dec-POMDP problems are considered to be NEXP-complete—as compared to PSPACE-complete for POMDPs and MPOMDPs (Bernstein, Givan, Immerman, and Zilberstein, 2002). However, approaches to making computation of Dec-POMDP policies tractable in the problem classes of interest have been proposed, e.g., by Oliehoek, Spaan, Whiteson, and Vlassis (2008), Omidshafiei et al. (2017), and Best, Cliff, Patten, Mettu, and Fitch (2019).

7.3.2 Application

For SHERPA and its VIPER use cases, work is being planned on improving results visualization and explanation, including integration of LPDM’s result explanation capabilities once they are developed. Additionally, use case models will be improved
by accommodating more fault types and action choices (e.g., rover speed selection).

Another area of future work is augmenting the current, purely random procedure of generating LPDM's policy-building scenarios — i.e., sequences $\phi = (s_0, \phi_1, \phi_2, \ldots)$, defined in Section 2.8 — with a method for selecting these scenarios more intentionally. The method will be based on adaptive stress testing (Lee, 2019; Lee, Mengshoel, and Kochenderfer, 2019) and the intent will be to create scenarios representing low-probability sequences of events that, nevertheless, could lead to premature mission termination. The scenarios would then be used to develop policies that maximize the chances of avoiding such adverse outcomes.

### 7.4 Final Remarks

The famed Stanford mathematical scientist George B. Dantzig (noted for his numerous contributions to operations research, statistics, computer science, economics, and industrial engineering) wrote the following in a historical overview of mathematical programming (Dantzig, 1991):

> In retrospect it is interesting to note that the original problem that started my research is still outstanding — namely the problem of planning or scheduling dynamically over time, particularly planning dynamically under uncertainty. If such a problem could be successfully solved it could (eventually through better planning) contribute to the well-being and stability of the world.

Remarkable progress has been made in the field of decision making under uncertainty since 1991, allowing more and more challenging problems to be solved. Dantzig’s vision, while still not realized, no longer feels so out of reach. It is hoped that this thesis, in a small way, contributes to the ability of making better decisions in complex real-world systems — especially when degradation of the system’s health is a factor — thus making it easier to accomplish more, explore further, and do so in a safer manner.
Appendices
Appendix A

Results of the Light-Dark Experiments

This appendix contains the results of POMDP solver evaluation experiments conducted on different variants of the Light-Dark Domain problem (referred to, for brevity, as Light-Dark). The experiments and their results are discussed in detail in Chapter 5. Results for the 1D and 2D variants of the canonical (partially stochastic) Light-Dark problem are in Tables A.1 and A.2, respectively. Similarly, Tables A.3 and A.4 contain results for the 1D and 2D variants of the fully stochastic Light-Dark problem. Results for the model uncertainty experiments on Light Dark 1D are in Table A.5 and those on Light-Dark 2D, in Table A.6. In the tables, $\mu$ denotes the mean value of a metric. It is accompanied by the standard deviation, $\sigma$, in parentheses.

Identifiers used in the tables for the different solver configurations are listed below:

- **SA** standard action space (discrete)
- **EA** extended action space (discrete)
- **BV** Blind Value algorithm action space (continuous)
- **AAS** AAS algorithm action space (continuous)
- **DO** discrete observation space
- **TCC** observation spaces handled by the TCC algorithm (continuous)
Table A.1: Solver benchmarking results, partially stochastic Light-Dark 1D

(a) $s_0 = -2\pi$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>16.90 (010.46)</td>
<td>-56.10 (026.70)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>12.04 (006.08)</td>
<td>-34.47 (013.94)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>14.86 (008.24)</td>
<td>-38.98 (018.93)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>10.98 (006.35)</td>
<td>-36.51 (018.04)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>11.06 (004.94)</td>
<td>-36.83 (018.75)</td>
</tr>
</tbody>
</table>

(b) $s_0 = \pi/2$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>14.82 (009.65)</td>
<td>-07.07 (007.53)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>10.98 (008.23)</td>
<td>-06.03 (004.74)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>13.12 (008.81)</td>
<td>-08.92 (008.50)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>10.88 (006.65)</td>
<td>-06.84 (007.03)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>09.85 (007.53)</td>
<td>-05.62 (005.62)</td>
</tr>
</tbody>
</table>

(c) $s_0 = 3\pi/2$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>10.12 (003.05)</td>
<td>-14.98 (000.41)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>10.34 (010.01)</td>
<td>-60.96 (114.53)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>10.78 (005.27)</td>
<td>-16.05 (006.45)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>07.38 (005.93)</td>
<td>-11.59 (026.72)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>06.91 (004.27)</td>
<td>-10.66 (014.03)</td>
</tr>
</tbody>
</table>

(d) $s_0 = 2\pi$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>13.24 (006.96)</td>
<td>-36.93 (003.54)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>08.74 (004.35)</td>
<td>-18.76 (013.44)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>10.42 (004.34)</td>
<td>-33.65 (004.62)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>07.98 (005.04)</td>
<td>-14.04 (001.74)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>07.21 (007.40)</td>
<td>-13.28 (006.12)</td>
</tr>
</tbody>
</table>
APPENDIX A. RESULTS OF THE LIGHT-DARK EXPERIMENTS

Table A.2: Solver benchmarking results, partially stochastic Light-Dark 2D

(a) $s_0 = [-2\pi, \pi]$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>46.26 (023.38)</td>
<td>-124.80 (058.26)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>31.52 (017.03)</td>
<td>-105.17 (083.12)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>35.80 (018.37)</td>
<td>-85.95 (048.95)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>29.62 (015.72)</td>
<td>-81.20 (047.71)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>25.82 (014.53)</td>
<td>-75.38 (044.96)</td>
</tr>
</tbody>
</table>

(b) $s_0 = [\pi/2, -\pi/2]$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>38.84 (025.42)</td>
<td>-22.96 (016.78)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>26.82 (014.18)</td>
<td>-17.62 (011.81)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>30.54 (014.39)</td>
<td>-18.49 (010.68)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>24.30 (012.52)</td>
<td>-15.73 (009.46)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>22.86 (012.20)</td>
<td>-13.41 (008.39)</td>
</tr>
</tbody>
</table>

(c) $s_0 = [\pi, 2\pi]$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>27.18 (012.83)</td>
<td>-49.02 (009.75)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>26.70 (014.95)</td>
<td>-25.33 (007.02)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>24.60 (011.42)</td>
<td>-23.87 (005.43)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>22.02 (010.82)</td>
<td>-21.75 (004.75)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>19.56 (007.93)</td>
<td>-18.80 (003.26)</td>
</tr>
</tbody>
</table>

(d) $s_0 = [2\pi, -\pi]$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>21.56 (015.42)</td>
<td>-44.87 (006.35)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>17.32 (008.18)</td>
<td>-20.80 (002.22)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>20.32 (013.49)</td>
<td>-19.92 (003.22)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>16.76 (007.52)</td>
<td>-17.84 (002.63)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>14.38 (007.97)</td>
<td>-16.19 (001.46)</td>
</tr>
</tbody>
</table>
Table A.3: Solver benchmarking results, fully stochastic Light-Dark 1D

(a) $s_0 = -2\pi$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>16.50 (009.53)</td>
<td>-57.41 (037.88)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>13.78 (009.67)</td>
<td>-35.66 (022.27)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>14.06 (011.31)</td>
<td>-37.20 (023.78)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>13.04 (009.39)</td>
<td>-35.45 (019.23)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>12.45 (009.18)</td>
<td>-35.91 (018.16)</td>
</tr>
</tbody>
</table>

(b) $s_0 = \pi/2$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>13.76 (014.51)</td>
<td>-09.58 (013.08)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>09.88 (005.66)</td>
<td>-09.17 (011.40)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>14.56 (012.40)</td>
<td>-15.24 (032.24)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>10.62 (007.28)</td>
<td>-09.82 (011.39)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>10.09 (006.21)</td>
<td>-08.73 (009.82)</td>
</tr>
</tbody>
</table>

(c) $s_0 = 3\pi/2$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>14.26 (008.74)</td>
<td>-26.23 (018.68)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>10.28 (009.99)</td>
<td>-21.64 (029.60)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>12.14 (008.09)</td>
<td>-40.56 (046.70)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>09.44 (006.62)</td>
<td>-20.28 (029.14)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>09.20 (004.27)</td>
<td>-18.43 (030.96)</td>
</tr>
</tbody>
</table>

(d) $s_0 = 2\pi$

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>14.18 (008.94)</td>
<td>-41.86 (009.25)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>11.02 (007.71)</td>
<td>-20.77 (010.92)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>10.04 (007.19)</td>
<td>-17.93 (008.38)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>09.44 (006.19)</td>
<td>-16.87 (008.17)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>08.95 (007.64)</td>
<td>-15.28 (008.59)</td>
</tr>
</tbody>
</table>
Table A.4: Solver benchmarking results, fully stochastic Light-Dark 2D

(a) \( s_0 = [-2\pi, \pi] \)

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, ( \mu (\sigma) )</th>
<th>Reward, ( \mu (\sigma) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>101.30 (083.26)</td>
<td>-299.73 (0304.50)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>092.28 (086.20)</td>
<td>-241.85 (0293.08)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>093.18 (086.49)</td>
<td>-234.89 (0190.39)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>078.72 (075.55)</td>
<td>-222.81 (0209.78)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>073.94 (077.38)</td>
<td>-205.40 (0198.56)</td>
</tr>
</tbody>
</table>

(b) \( s_0 = [\pi/2, -\pi/2] \)

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, ( \mu (\sigma) )</th>
<th>Reward, ( \mu (\sigma) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>100.10 (095.68)</td>
<td>-217.30 (0290.83)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>099.92 (095.02)</td>
<td>-182.40 (0282.83)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>110.18 (111.11)</td>
<td>-464.22 (1886.37)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>065.68 (062.45)</td>
<td>-127.18 (0210.40)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>062.21 (057.13)</td>
<td>-120.62 (0219.75)</td>
</tr>
</tbody>
</table>

(c) \( s_0 = [\pi, 2\pi] \)

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, ( \mu (\sigma) )</th>
<th>Reward, ( \mu (\sigma) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>130.98 (117.60)</td>
<td>-349.29 (0586.54)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>066.28 (068.23)</td>
<td>-136.29 (0235.80)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>088.14 (081.62)</td>
<td>-195.33 (0316.86)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>062.80 (069.11)</td>
<td>-125.02 (0214.06)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>061.04 (065.89)</td>
<td>-121.51 (0188.37)</td>
</tr>
</tbody>
</table>

(d) \( s_0 = [2\pi, -\pi] \)

<table>
<thead>
<tr>
<th>Solver (action mode, observation mode)</th>
<th>Steps, ( \mu (\sigma) )</th>
<th>Reward, ( \mu (\sigma) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-DESPOT (SA, DO)</td>
<td>082.16 (081.79)</td>
<td>-170.79 (0168.81)</td>
</tr>
<tr>
<td>AR-DESPOT (EA, DO)</td>
<td>066.58 (064.21)</td>
<td>-142.43 (0220.52)</td>
</tr>
<tr>
<td>AR-DESPOT (BV, DO)</td>
<td>102.42 (113.04)</td>
<td>-243.24 (0305.66)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>060.84 (073.69)</td>
<td>-138.31 (0248.14)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>056.29 (078.20)</td>
<td>-128.94 (0246.35)</td>
</tr>
</tbody>
</table>
### APPENDIX A. RESULTS OF THE LIGHT-DARK EXPERIMENTS

Table A.5: Model uncertainty experiment results, Light-Dark 1D

(a) DESPOT (EA, DO), $s_0 = \pi/2$

<table>
<thead>
<tr>
<th>Model uncertainty coefficient $\xi$</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>10.98 (008.23)</td>
<td>-06.03 (004.74)</td>
</tr>
<tr>
<td>0.10</td>
<td>11.91 (007.61)</td>
<td>-07.91 (007.40)</td>
</tr>
<tr>
<td>0.20</td>
<td>13.01 (007.78)</td>
<td>-09.93 (010.43)</td>
</tr>
<tr>
<td>0.30</td>
<td>15.42 (011.82)</td>
<td>-11.87 (012.94)</td>
</tr>
<tr>
<td>0.40</td>
<td>15.91 (013.87)</td>
<td>-17.52 (034.63)</td>
</tr>
<tr>
<td>0.50</td>
<td>19.17 (017.63)</td>
<td>-30.74 (067.10)</td>
</tr>
<tr>
<td>0.60</td>
<td>20.72 (021.65)</td>
<td>-38.43 (118.41)</td>
</tr>
<tr>
<td>0.70</td>
<td>21.34 (019.95)</td>
<td>-148.33 (828.81)</td>
</tr>
<tr>
<td>0.80</td>
<td>29.04 (030.02)</td>
<td>-331.27 (1824.78)</td>
</tr>
<tr>
<td>0.90</td>
<td>30.87 (034.00)</td>
<td>-411.48 (2371.27)</td>
</tr>
<tr>
<td>1.00</td>
<td>34.46 (046.99)</td>
<td>-613.89 (2965.14)</td>
</tr>
</tbody>
</table>

(b) LPDM (AAS, TCC), $s_0 = \pi/2$

<table>
<thead>
<tr>
<th>Model uncertainty coefficient $\xi$</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>09.85 (007.53)</td>
<td>-05.62 (008.11)</td>
</tr>
<tr>
<td>0.10</td>
<td>10.85 (007.74)</td>
<td>-07.72 (008.77)</td>
</tr>
<tr>
<td>0.20</td>
<td>12.15 (009.15)</td>
<td>-10.44 (012.11)</td>
</tr>
<tr>
<td>0.30</td>
<td>13.19 (010.29)</td>
<td>-11.76 (013.91)</td>
</tr>
<tr>
<td>0.40</td>
<td>13.65 (010.72)</td>
<td>-12.54 (035.09)</td>
</tr>
<tr>
<td>0.50</td>
<td>14.15 (014.19)</td>
<td>-26.19 (084.71)</td>
</tr>
<tr>
<td>0.60</td>
<td>16.59 (020.47)</td>
<td>-34.87 (112.96)</td>
</tr>
<tr>
<td>0.70</td>
<td>18.49 (021.51)</td>
<td>-144.11 (709.14)</td>
</tr>
<tr>
<td>0.80</td>
<td>22.33 (023.89)</td>
<td>-276.65 (1646.24)</td>
</tr>
<tr>
<td>0.90</td>
<td>24.60 (024.60)</td>
<td>-345.23 (1870.39)</td>
</tr>
<tr>
<td>1.00</td>
<td>27.32 (034.47)</td>
<td>-571.88 (2117.61)</td>
</tr>
</tbody>
</table>
Table A.6: Model uncertainty experiment results, Light-Dark 2D

(a) DESPOT (EA, DO), $s_0 = [\pi/2, -\pi/2]$

<table>
<thead>
<tr>
<th>Model uncertainty coefficient $\xi$</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>26.82 (014.18)</td>
<td>-17.62 (011.81)</td>
</tr>
<tr>
<td>0.10</td>
<td>61.60 (046.98)</td>
<td>-37.68 (031.22)</td>
</tr>
<tr>
<td>0.20</td>
<td>98.28 (100.93)</td>
<td>-102.40 (138.11)</td>
</tr>
<tr>
<td>0.30</td>
<td>155.44 (218.96)</td>
<td>-354.49 (392.32)</td>
</tr>
<tr>
<td>0.40</td>
<td>219.34 (314.51)</td>
<td>$-3.0 \cdot 10^4$ (2.9 \cdot 10^3)</td>
</tr>
<tr>
<td>0.50</td>
<td>403.85 (407.23)</td>
<td>$-6.4 \cdot 10^6$ (6.7 \cdot 10^7)</td>
</tr>
<tr>
<td>0.60</td>
<td>542.38 (552.74)</td>
<td>$-9.1 \cdot 10^6$ (4.9 \cdot 10^7)</td>
</tr>
<tr>
<td>0.70</td>
<td>688.86 (708.72)</td>
<td>$-2.4 \cdot 10^7$ (1.1 \cdot 10^8)</td>
</tr>
<tr>
<td>0.80</td>
<td>913.22 (1046.15)</td>
<td>$-8.7 \cdot 10^7$ (2.6 \cdot 10^8)</td>
</tr>
<tr>
<td>0.90</td>
<td>1154.02 (1258.83)</td>
<td>$-1.8 \cdot 10^8$ (5.5 \cdot 10^8)</td>
</tr>
<tr>
<td>1.00</td>
<td>1505.37 (1748.11)</td>
<td>$-5.2 \cdot 10^8$ (1.8 \cdot 10^9)</td>
</tr>
</tbody>
</table>

(b) LPDM (AAS, TCC), $s_0 = [\pi/2, -\pi/2]$

<table>
<thead>
<tr>
<th>Model uncertainty coefficient $\xi$</th>
<th>Steps, $\mu (\sigma)$</th>
<th>Reward, $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>22.86 (012.20)</td>
<td>-13.41 (008.39)</td>
</tr>
<tr>
<td>0.10</td>
<td>44.20 (032.17)</td>
<td>-27.69 (022.17)</td>
</tr>
<tr>
<td>0.20</td>
<td>88.93 (094.08)</td>
<td>-98.56 (144.73)</td>
</tr>
<tr>
<td>0.30</td>
<td>142.30 (129.31)</td>
<td>-290.47 (394.70)</td>
</tr>
<tr>
<td>0.40</td>
<td>207.98 (235.16)</td>
<td>$-4.1 \cdot 10^3$ (3.3 \cdot 10^4)</td>
</tr>
<tr>
<td>0.50</td>
<td>351.83 (391.56)</td>
<td>$-1.8 \cdot 10^6$ (9.4 \cdot 10^6)</td>
</tr>
<tr>
<td>0.60</td>
<td>467.46 (517.80)</td>
<td>$-6.2 \cdot 10^6$ (2.6 \cdot 10^7)</td>
</tr>
<tr>
<td>0.70</td>
<td>503.65 (607.23)</td>
<td>$-1.1 \cdot 10^7$ (5.0 \cdot 10^7)</td>
</tr>
<tr>
<td>0.80</td>
<td>791.53 (828.09)</td>
<td>$-4.6 \cdot 10^7$ (2.3 \cdot 10^8)</td>
</tr>
<tr>
<td>0.90</td>
<td>1039.07 (1148.27)</td>
<td>$-1.4 \cdot 10^8$ (4.2 \cdot 10^8)</td>
</tr>
<tr>
<td>1.00</td>
<td>1291.86 (1419.79)</td>
<td>$-3.3 \cdot 10^8$ (1.0 \cdot 10^9)</td>
</tr>
</tbody>
</table>
Appendix B

Mission Parameters Used in the SHERPA Models

This appendix contains the parameters used in modeling the VIPER lunar rover mission within SHERPA (Chapter 6). Since the VIPER mission is currently under active development, some of its parameters are either still undefined or cannot yet be released publicly. In both of these cases, approximate values were used.

General rover parameters used in the physics models are listed in Table B.1. Table B.2 describes the uncertainties modeled in the system state transitions. Finally, Table B.3 provides the additional vehicle parameters needed for the fault accommodation scenarios. As values for some of the drive motor parameters were not yet available, specifications for the similarly sized Mars Exploration Rover motors were substituted.
### Table B.1: Rover parameters used in the SHERPA models.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effective speed (speed-made-good)</td>
<td>$v$</td>
<td>$0.012 \text{ m s}^{-1}$</td>
</tr>
<tr>
<td>Rover mass</td>
<td>$m$</td>
<td>300.0 kg</td>
</tr>
<tr>
<td>Rolling friction coefficient</td>
<td>$C_r$</td>
<td>0.12</td>
</tr>
<tr>
<td>Drive gear power loss coefficient</td>
<td>$C_{\text{gear}}$</td>
<td>120.0 N</td>
</tr>
<tr>
<td>Motor power loss coefficient</td>
<td>$C_{\text{motor}}$</td>
<td>480.0 N</td>
</tr>
<tr>
<td>Maximum battery capacity</td>
<td>$Q_{\text{max}}$</td>
<td>4500 Wh</td>
</tr>
<tr>
<td>Number of solar panels</td>
<td>$N_{\text{pv}}$</td>
<td>3</td>
</tr>
<tr>
<td>Charge efficiency coefficient</td>
<td>$\eta_c$</td>
<td>0.95</td>
</tr>
<tr>
<td>Discharge efficiency coefficient</td>
<td>$\eta_d$</td>
<td>0.95</td>
</tr>
<tr>
<td>PV panel area (per unit)</td>
<td>$A_{\text{pv}}$</td>
<td>1.0 m$^2$</td>
</tr>
<tr>
<td>PV panel area lunar dust occlusion ratio (expected)</td>
<td>$\zeta$</td>
<td>0.1</td>
</tr>
<tr>
<td>PV cell current density at maximum power at the beginning of life</td>
<td>$J_0$</td>
<td>160.0 A m$^{-2}$</td>
</tr>
<tr>
<td>PV cell voltage at maximum power at the beginning of life</td>
<td>$V_0$</td>
<td>2.5 V</td>
</tr>
<tr>
<td>PV panel current density increment per °C</td>
<td>$\Delta J_T$</td>
<td>$-0.066 \text{ A m}^{-2} \text{ °C}^{-1}$</td>
</tr>
<tr>
<td>PV panel voltage increment per °C</td>
<td>$\Delta V_T$</td>
<td>$-0.007 \text{ V °C}^{-1}$</td>
</tr>
<tr>
<td>PV panel best operating temperature</td>
<td>$T_0$</td>
<td>30 °C</td>
</tr>
<tr>
<td>PV panel expected operating temperature</td>
<td>$T_{\text{pv}}$</td>
<td>80 °C</td>
</tr>
</tbody>
</table>
### Table B.2: Uncertainties modeled for the VIPER mission

<table>
<thead>
<tr>
<th>Variable</th>
<th>Distribution</th>
<th>Mean, $\mu$</th>
<th>Variance, $\sigma^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start time</td>
<td>Gaussian</td>
<td>planned start time</td>
<td>2 hours</td>
</tr>
<tr>
<td>Initial battery charge</td>
<td>Gaussian (half)*</td>
<td>full charge</td>
<td>0.2$\mu$</td>
</tr>
<tr>
<td>Power draw</td>
<td>Gaussian</td>
<td>expected draw</td>
<td>0.2$\mu$</td>
</tr>
<tr>
<td>Recharge efficiency</td>
<td>Gaussian</td>
<td>expected efficiency</td>
<td>0.2$\mu$</td>
</tr>
<tr>
<td>Effective speed</td>
<td>Gaussian</td>
<td>speed-made-good</td>
<td>0.2$\mu$</td>
</tr>
<tr>
<td>Activity duration</td>
<td>Gaussian</td>
<td>expected duration</td>
<td>0.2$\mu$</td>
</tr>
<tr>
<td>Actual ice depth at $(x, y)$</td>
<td>Gaussian*</td>
<td>modeled depth at $(x, y)$</td>
<td>0.2$\mu$</td>
</tr>
</tbody>
</table>

* Clamped to min/max

### Table B.3: SHERPA fault accommodation experiment parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor thermal inertia coefficient (estimated)</td>
<td>$C_{m,t}$</td>
<td>2.0</td>
</tr>
<tr>
<td>Nominal normalized coefficient of motor friction</td>
<td>$C_f$</td>
<td>1.0</td>
</tr>
<tr>
<td>Nominal operating temperature of motor windings</td>
<td>$T_{w,n}$</td>
<td>60.0 °C</td>
</tr>
<tr>
<td>Maximum operating temperature of motor windings</td>
<td>$T_{w,max}$</td>
<td>85.0 °C</td>
</tr>
<tr>
<td>Expected nominal motor housing temperature in sunlight</td>
<td>$T_{m,sun}$</td>
<td>25.0 °C</td>
</tr>
<tr>
<td>Expected nominal motor housing temperature in sun shadow</td>
<td>$T_{m,shadow}$</td>
<td>-40.0 °C</td>
</tr>
<tr>
<td>Motor windings resistance</td>
<td>$R_w$</td>
<td>2.28 Ω</td>
</tr>
<tr>
<td>Motor voltage</td>
<td>$V_{m}$</td>
<td>24.0 V</td>
</tr>
</tbody>
</table>
Chapter 6 describes the SHERPA decision support framework, its Traverse Evaluation and Refinement (TER) use case developed for the VIPER mission, and the experiments conducted on TER involving a variety of decision-making policies. Results of the experiments are provided in this appendix, consisting of the following three sets: experiments with recoverable faults and delays (Tables C.1 and C.2), fault accommodation experiments (Table C.3), and model uncertainty experiments (Tables C.4 and C.5). In the tables, metrics computed for the fully successful (FS) traverses only are labeled accordingly and so are the metrics computed solely for the partially successful (PS) traverses. The percentage of fully successful traverses and the normalized reward $R/R_{\text{max}}$ are the only metrics computed for all traverse execution instances within an experiment. As in Appendix A, $\mu$ denotes the mean value of a metric, followed by the standard deviation, $\sigma$, in parentheses. Identifiers used for the different DESPOT and LPDM configurations are also the same as in Appendix A.
## APPENDIX C. RESULTS OF THE SHERPA EXPERIMENTS

Table C.1: Recoverable faults and delays experiment results, TER use case

<table>
<thead>
<tr>
<th>Policy</th>
<th>Fully successful traverses, %</th>
<th>Visited waypoints $\mu(\sigma)$, PS</th>
<th>Reward, $R/R_{max}$ $\mu(\sigma)$, all</th>
<th>Duration vs nominal $\mu(\sigma)$, FS</th>
<th>Duration, hours $\mu(\sigma)$, FS</th>
<th>Duration, hours $\mu(\sigma)$, PS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSR</td>
<td>72.00</td>
<td>22.83 (0.81)</td>
<td>0.27 (0.24)</td>
<td>1.27 (0.05)</td>
<td>224.72 (09.25)</td>
<td>235.73 (15.94)</td>
</tr>
<tr>
<td>Full Charge</td>
<td>00.00</td>
<td>21.41 (1.35)</td>
<td>0.10 (0.07)</td>
<td>N/A (N/A)</td>
<td>N/A (N/A)</td>
<td>268.33 (22.86)</td>
</tr>
<tr>
<td>After Activities</td>
<td>82.00</td>
<td>18.78 (4.53)</td>
<td>0.27 (0.35)</td>
<td>1.24 (0.09)</td>
<td>219.93 (08.46)</td>
<td>203.12 (80.29)</td>
</tr>
<tr>
<td>Look Ahead</td>
<td>69.00</td>
<td>17.26 (5.87)</td>
<td>$-1.78$ (2.38)</td>
<td>1.20 (0.05)</td>
<td>212.98 (09.20)</td>
<td>139.63 (36.09)</td>
</tr>
<tr>
<td>Zone Crossing</td>
<td>41.00</td>
<td>20.35 (3.11)</td>
<td>0.20 (0.49)</td>
<td>1.31 (0.05)</td>
<td>232.44 (8.37)</td>
<td>275.68 (06.24)</td>
</tr>
<tr>
<td>Prognostic</td>
<td>00.00</td>
<td>22.07 (0.91)</td>
<td>$-0.42$ (0.42)</td>
<td>N/A (N/A)</td>
<td>N/A (N/A)</td>
<td>284.71 (09.59)</td>
</tr>
<tr>
<td>DESPOT (SA, DO)</td>
<td>88.00</td>
<td>17.45 (3.98)</td>
<td>0.28 (0.75)</td>
<td>1.24 (0.06)</td>
<td>219.85 (10.69)</td>
<td>155.04 (63.50)</td>
</tr>
<tr>
<td>DESPOT (EA, DO)</td>
<td>89.00</td>
<td>17.00 (1.73)</td>
<td>0.32 (0.97)</td>
<td>1.21 (0.04)</td>
<td>214.58 (08.07)</td>
<td>153.85 (70.82)</td>
</tr>
<tr>
<td>DESPOT (BV, DO)</td>
<td>88.00</td>
<td>18.48 (3.76)</td>
<td>0.27 (0.65)</td>
<td>1.22 (0.05)</td>
<td>216.27 (08.85)</td>
<td>156.89 (55.34)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>93.00</td>
<td>18.65 (6.03)</td>
<td>0.27 (0.91)</td>
<td>1.21 (0.04)</td>
<td>214.48 (07.41)</td>
<td>162.55 (70.03)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>94.00</td>
<td>17.00 (5.39)</td>
<td>0.26 (0.77)</td>
<td>1.20 (0.04)</td>
<td>212.70 (07.53)</td>
<td>147.70 (51.59)</td>
</tr>
</tbody>
</table>
# APPENDIX C. RESULTS OF THE SHERPA EXPERIMENTS

Table C.2: Recoverable faults and delays experiment results, TER use case (100 particle beliefs)

<table>
<thead>
<tr>
<th>Policy</th>
<th>Fully successful traverses, %</th>
<th>Visited waypoints $\mu (\sigma)$, PS</th>
<th>Reward, $R/R_{\text{max}}$ $\mu (\sigma)$, all</th>
<th>Duration vs nominal $\mu (\sigma)$, FS</th>
<th>Duration, hours $\mu (\sigma)$, FS</th>
<th>Duration, hours $\mu (\sigma)$, PS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESPOT (SA, DO)</td>
<td>92.00</td>
<td>23.75 (0.48)</td>
<td>0.33 (0.37)</td>
<td>1.25 (0.06)</td>
<td>221.81 (06.37)</td>
<td>277.38 (02.65)</td>
</tr>
<tr>
<td>DESPOT (EA, DO)</td>
<td>93.00</td>
<td>23.87 (0.41)</td>
<td>0.36 (0.32)</td>
<td>1.25 (0.06)</td>
<td>221.67 (11.13)</td>
<td>278.38 (03.72)</td>
</tr>
<tr>
<td>DESPOT (BV, DO)</td>
<td>93.00</td>
<td>22.35 (4.18)</td>
<td>0.32 (0.35)</td>
<td>1.25 (0.04)</td>
<td>221.30 (06.93)</td>
<td>254.85 (66.70)</td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>95.00</td>
<td>23.80 (0.45)</td>
<td>0.32 (0.33)</td>
<td>1.26 (0.04)</td>
<td>222.97 (07.00)</td>
<td>278.92 (03.02)</td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>96.00</td>
<td>24.00 (0.00)</td>
<td>0.32 (0.31)</td>
<td>1.26 (0.04)</td>
<td>222.71 (07.12)</td>
<td>279.56 (03.37)</td>
</tr>
</tbody>
</table>
Table C.3: Fault accommodation experiment results, TER use case

<table>
<thead>
<tr>
<th>Policy</th>
<th>Fully successful traverses, %</th>
<th>Visited waypoints $\mu (\sigma)$, PS</th>
<th>Reward, $R/R_{\text{max}} \mu (\sigma)$, all</th>
<th>Duration vs nominal $\mu (\sigma)$, FS</th>
<th>Duration, hours FS $\mu (\sigma)$</th>
<th>Duration, hours FS, PS $\mu (\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSR</td>
<td>00.00 02.28 0.03 (1.11) (0.05)</td>
<td>N/A 0.03 (N/A) (N/A)</td>
<td>N/A 0.07 (N/A)</td>
<td>023.04 (03.79)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full Charge</td>
<td>00.00 02.56 0.03 (1.96) (0.13)</td>
<td>N/A 0.03 (N/A) (N/A)</td>
<td>N/A 0.07 (N/A)</td>
<td>025.31 (05.86)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>After Activities</td>
<td>00.00 03.07 0.04 (2.25) (0.07)</td>
<td>N/A 0.03 (N/A) (N/A)</td>
<td>N/A 0.07 (N/A)</td>
<td>028.22 (07.54)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Look Ahead</td>
<td>00.00 02.31 0.03 (1.24) (0.10)</td>
<td>N/A 0.03 (N/A) (N/A)</td>
<td>N/A 0.07 (N/A)</td>
<td>026.88 (05.95)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zone Crossing</td>
<td>00.00 01.94 0.03 (1.81) (0.07)</td>
<td>N/A 0.03 (N/A) (N/A)</td>
<td>N/A 0.07 (N/A)</td>
<td>021.24 (07.13)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prognostic</td>
<td>00.00 02.05 0.03 (1.53) (0.11)</td>
<td>N/A 0.03 (N/A) (N/A)</td>
<td>N/A 0.07 (N/A)</td>
<td>023.03 (06.41)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESPOT (SA, DO)</td>
<td>69.00 17.22 0.12 (5.72) (0.61)</td>
<td>1.92 (0.09) (15.96)</td>
<td>340.32 (48.66)</td>
<td>228.37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESPOT (EA, DO)</td>
<td>73.00 18.93 0.14 (3.93) (0.82)</td>
<td>1.78 (0.07) (12.41)</td>
<td>315.49 (51.70)</td>
<td>225.74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESPOT (BV, DO)</td>
<td>75.00 18.38 0.15 (4.48) (0.79)</td>
<td>1.85 (0.11) (19.50)</td>
<td>327.91 (60.36)</td>
<td>230.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPDM (AAS, DO)</td>
<td>80.00 19.20 0.17 (2.24) (0.90)</td>
<td>1.76 (0.03) (05.30)</td>
<td>311.97 (31.17)</td>
<td>228.54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPDM (AAS, TCC)</td>
<td>82.00 19.56 0.17 (2.01) (0.77)</td>
<td>1.81 (0.05) (08.85)</td>
<td>320.82 (45.81)</td>
<td>232.28</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table C.4: Model uncertainty experiment results, TER use case, DESPOT (EA, DO)

<table>
<thead>
<tr>
<th>Policy</th>
<th>Fully successful traverses, %</th>
<th>Visited waypoints, $\mu (\sigma)$, PS</th>
<th>Reward, $R/R_{\text{max}}$, $\mu (\sigma)$, all</th>
<th>Duration vs nominal, $\mu (\sigma)$, FS</th>
<th>Duration, hours, $\mu (\sigma)$, FS</th>
<th>Duration, hours, $\mu (\sigma)$, PS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>89.00</td>
<td>17.00 (5.12)</td>
<td>0.32 (0.89)</td>
<td>1.21 (0.04)</td>
<td>214.58 (08.07)</td>
<td>153.89 (70.82)</td>
</tr>
<tr>
<td>0.3</td>
<td>82.00</td>
<td>21.73 (3.72)</td>
<td>0.25 (0.91)</td>
<td>1.22 (0.07)</td>
<td>217.24 (12.38)</td>
<td>225.60 (65.42)</td>
</tr>
<tr>
<td>0.4</td>
<td>62.00</td>
<td>20.76 (4.54)</td>
<td>0.17 (0.89)</td>
<td>1.22 (0.13)</td>
<td>217.03 (23.24)</td>
<td>226.93 (73.43)</td>
</tr>
<tr>
<td>0.5</td>
<td>44.00</td>
<td>22.43 (1.79)</td>
<td>0.14 (0.78)</td>
<td>1.21 (0.24)</td>
<td>215.53 (44.05)</td>
<td>272.59 (31.53)</td>
</tr>
<tr>
<td>0.6</td>
<td>12.00</td>
<td>22.47 (1.46)</td>
<td>$-0.12$ (0.60)</td>
<td>1.13 (0.33)</td>
<td>201.38 (58.62)</td>
<td>277.65 (22.70)</td>
</tr>
<tr>
<td>0.7</td>
<td>00.00</td>
<td>21.58 (1.00)</td>
<td>$-0.31$ (0.43)</td>
<td>N/A (N/A)</td>
<td>N/A</td>
<td>277.56 (13.09)</td>
</tr>
<tr>
<td>0.8</td>
<td>00.00</td>
<td>21.21 (0.88)</td>
<td>$-0.21$ (0.42)</td>
<td>N/A (N/A)</td>
<td>N/A</td>
<td>275.39 (14.32)</td>
</tr>
<tr>
<td>0.9</td>
<td>00.00</td>
<td>20.32 (1.08)</td>
<td>$-0.14$ (0.40)</td>
<td>N/A (N/A)</td>
<td>N/A</td>
<td>264.11 (14.73)</td>
</tr>
<tr>
<td>1.0</td>
<td>00.00</td>
<td>19.81 (1.15)</td>
<td>$-0.10$ (0.34)</td>
<td>N/A (N/A)</td>
<td>N/A</td>
<td>257.36 (12.95)</td>
</tr>
</tbody>
</table>
Table C.5: Model uncertainty experiment results, TER use case, LPDM (AAS, TCC)

<table>
<thead>
<tr>
<th>Policy</th>
<th>Fully successful traverses, %</th>
<th>Visited waypoints $\mu (\sigma)$, PS</th>
<th>Reward, $R/R_{\text{max}}$ $\mu (\sigma)$, all</th>
<th>Duration vs nominal $\mu (\sigma)$, FS</th>
<th>Duration, hours $\mu (\sigma)$, FS</th>
<th>Duration, hours $\mu (\sigma)$, PS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>94.00</td>
<td>17.00 (5.39)</td>
<td>0.26 (0.87)</td>
<td>1.20 (0.04)</td>
<td>212.70 (07.53)</td>
<td>147.70 (51.59)</td>
</tr>
<tr>
<td>0.3</td>
<td>85.00</td>
<td>15.79 (4.85)</td>
<td>0.24 (0.92)</td>
<td>1.21 (0.07)</td>
<td>214.78 (12.48)</td>
<td>140.12 (64.34)</td>
</tr>
<tr>
<td>0.4</td>
<td>73.00</td>
<td>17.89 (5.23)</td>
<td>0.17 (0.99)</td>
<td>1.20 (0.16)</td>
<td>212.13 (28.89)</td>
<td>171.96 (75.60)</td>
</tr>
<tr>
<td>0.5</td>
<td>49.00</td>
<td>22.25 (2.37)</td>
<td>0.06 (0.90)</td>
<td>1.18 (0.19)</td>
<td>209.05 (34.13)</td>
<td>257.07 (50.91)</td>
</tr>
<tr>
<td>0.6</td>
<td>35.00</td>
<td>22.59 (1.13)</td>
<td>0.01 (0.83)</td>
<td>1.15 (0.32)</td>
<td>203.91 (56.95)</td>
<td>277.09 (23.07)</td>
</tr>
<tr>
<td>0.7</td>
<td>19.00</td>
<td>22.39 (1.00)</td>
<td>$-0.02$ (0.65)</td>
<td>1.06 (0.38)</td>
<td>188.80 (68.83)</td>
<td>281.49 (07.28)</td>
</tr>
<tr>
<td>0.8</td>
<td>4.00</td>
<td>21.98 (0.97)</td>
<td>$-0.13$ (0.46)</td>
<td>0.79 (0.45)</td>
<td>140.45 (79.77)</td>
<td>280.68 (06.84)</td>
</tr>
<tr>
<td>0.9</td>
<td>1.00</td>
<td>21.20 (1.02)</td>
<td>$-0.14$ (0.40)</td>
<td>0.67 (N/A)</td>
<td>118.59 (N/A)</td>
<td>273.73 (14.19)</td>
</tr>
<tr>
<td>1.0</td>
<td>0.00</td>
<td>20.77 (0.97)</td>
<td>$-0.09$ (0.36)</td>
<td>N/A</td>
<td>N/A</td>
<td>267.73 (16.14)</td>
</tr>
</tbody>
</table>
Bibliography


Search in Belief Space”. In: *International Conference on Artificial Intelligence
Planning Systems* (cited on p. 58).

Approximation of Correlation Clustering and Consensus Clustering”. In: *Journal

and Human Settlement of the Moon and Cislunar Space – A Look Ahead at the
Possibilities Over the Next 50 Years”. In: *AIAA Propulsion and Energy Forum*
(cited on p. 6).

on Artificial Intelligence* (cited on p. 110).

Bresina, J. L., Dearden, R., Meuleau, N., Ramkrishnan, S., Smith, D., and Washington,
Challenge for AI”. In: *Uncertainty in Artificial Intelligence Conference* (cited on
p. 109).

GEN”. In: *IEEE International Conference on Space Mission Challenges for In-
formation Technology* (cited on p. 108).


verse Planning with Temporal-Spatial Constraints”. In: *International Conference
on Automated Planning and Scheduling (ICAPS)* (cited on pp. 108, 109).

In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1, pp. 1–
49 (cited on pp. 25, 60, 61).


BIBLIOGRAPHY


