Incorporating Human Cognitive Limitations Into Sequential Decision Making Problems and Algorithms

by

Sriram Gopalakrishnan

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved October 2022 by the
Graduate Supervisory Committee:

Subbarao Kambhampati, Chair
Siddharth Srivastava
Matthias Scheutz
Yu Zhang

ARIZONA STATE UNIVERSITY

December 2022

ABSTRACT

With improvements in automation and system capabilities, human responsibilities in those advanced systems can get more complicated; greater situational awareness and performance may be asked of human agents in roles such as fail-safe operators. This phenomenon of automation improvements requiring more from humans in the loop, is connected to the well-known "paradox of automation". Unfortunately, humans have cognitive limitations that can constrain a person's performance on a task. If one considers human cognitive limitations when designing solutions or policies for human agents, then better results are possible.

The focus of this dissertation is on improving human involvement in planning and execution for Sequential Decision Making (SDM) problems. Existing work already considers incorporating humans into planning and execution in SDM, but with limited consideration for cognitive limitations. The work herein focuses on how to improve human involvement through problems in motion planning, planning interfaces, Markov Decision Processes (MDP), and human-team scheduling. This done by first discussing the human modeling assumptions currently used in the literature and their shortcomings. Then this dissertation tackles a set of problems by considering problem-specific human cognitive limitations –such as those associated with memory and inference– as well as use lessons from fields such as cognitive ergonomics.

To my parents (Dhamayanthi Gopalakrishnan and Gopalakrishnan Srinivasan) and my brother (Pradeep Gopalakrishnan). Thank you for the love, support, and the space you gave me to chart my own path.

ACKNOWLEDGEMENTS

When I started my research journey, I was an enthusiastic researcher floundering in the research process with lofty and impractical ideas. I still have lofty ideas–which I hope now are more practical– and I think I've made progress in terms of research maturity. The lion's share of the credit for this belongs to my advisor Prof. Subbarao Kambhampati. Thank you helping me through my growing-pains as a researcher, and being a source of insights.

I'd like to thank my committee members (Prof. Siddharth Srivastava, Prof. Yu Zhang, and Prof. Matthias Scheutz) for understanding my work, and helping improve it. It takes a lot of effort to pore over research, and I greatly appreciate your support and guidance. Without you, my dissertation would not have come to fruition.

I would not have gotten started in research, if not for Dr.Hector Munoz-Avila. Thank you for collaborating with me on interesting ideas, and introducing me to the world of automated planning.

Along my research journey, I was fortunate to have collaborators outside Arizona State who worked with me. Thank you for sharing your time and energy with me. I'd like to thank Dr. Daniel Borrajo for his guidance, and fruitful collaborations. Your calm, methodical approach to looking at a problem helped me be more measured in how I thought about and presented my research. I was also fortunate to be able to work Dr. T. K. Satish Kumar on a very interesting problem; you set the standard for me on clever algorithms that connect across different disciplines. Dr. Ugur Kuter, thank you for being a part of my first research paper with Dr.Munoz-Avila, and providing me with early guidance. I'd also like to thank Dr. Cynthia Rudin who indirectly helped motivate the research direction herein. Your talk in 2020 at the International Conference on Automated Planning and Scheduling helped give me the

confidence needed to pursue my research direction. Your work helped me think more pragmatically about how our systems and algorithms would interact with humans.

I think completing a doctorate also requires having a good support system. My family gave me much of this. My parents (Dhamayanthi Gopalakrishnan and Gopalakrishnan Srinivasan) were particularly patient and supportive through the gruelling times and foul moods that happen during a PhD. My brother, who constantly encouraged me and gave me sage advise, was another pillar on this journey. A big source of support and joy was my yochan lab-family (Yantian Zha, Utkarsh Soni, Sarath Sreedharan, Mudit Verma, Sachin Grover, Zahra Zahedi, Alberto Olmo, Tathagata Chakraborty, Lin Guan, Anagha Kulkarni, Lydia Manikonda, Sailik Sengupta, Siddhant Bhambri, Karthik Valmeekam) and larger asu-family (Monica Dugan, Pamela Dunn, Jaya Krishnamurthy, Christina Sebring). You were there to help navigate through tough times, and share in the laughter to make things lighter. I'd like to particularly thank Yantian Zha, Utkarsh Soni, and Mudit Verma for working closely with me, and Sarath Sreedharan for helping me thinking through a lot of ideas. The final pillar of support were the people at my "home away from home" during my doctoral studies. Eric (Rick) Johnson, thank you for being a supportive and lively roommate and landlord for 5 years. It has been a pleasure. I'd also like to thank Adam Jesuadon, Steven Madler, and Shomit Barua who were fellow travellers during my journey at Arizona State University.

I'd like to end by thanking the people who make Arizona State University work. Thank you for making ASU possible. At ASU, I had the luxury of immersing my self in research, and met wonderful, supportive, and inspirational people. It has been a privilege to be a part of ASU.

TABLE OF CONTENTS

Page

LIST OF TABLES ix

LIST OF FIGURES xii

CHAPTER

1 INTRODUCTION 1

2 MODELING HUMAN COGNITIVE LIMITATIONS FOR SDM 6

3 MOTION PLANNING IN HUMAN-ROBOT SHARED SPACES 12

3.1 Motivation 12

3.2 Related Work 17

3.3 Problem Formulation 22

3.3.1 Measures for Position-based Predictability 23

3.4 Methodology 26

3.4.1 Computational Complexity 30

3.5 Experiments 31

3.6 Results and Analysis 32

3.7 Human-subject Experiments 35

3.8 Summary 41

4 TRADING OFF VALUE FOR REDUCED POLICY COMPLEXITY 42

4.1 Motivation 43

4.2 Related Work 45

4.3 Problem Definition 47

4.4 Delay Effect From Policy Confusion 50

4.5 Policy Computation Algorithm for SAMDP 51

4.5.1 Computing Reidentification Action Likelihoods 52

4.5.2 Translating to the Equivalent MRP 53

CHAPTER Page

4.5.3 GVPI Search Process 53
4.5.4 Trading Value for Reduced Complexity 54
4.5.5 Obtaining the Classification Likelihood Matrix 55
4.6 Experiments and Results 56
4.6.1 Warehouse Worker Domain Setup 56
4.6.2 Gridworld Experimental Setup 59
4.6.3 Results 59
4.6.4 Human Studies 62
4.7 Summary 67
5 ACCOUNTING FOR BEHAVIORAL RESPONSE TO UNCERTAINTY IN MDP 69
5.1 Motivation 69
5.2 Problem Definition and Human Model Used 71
5.2.1 Human Model 71
5.2.2 POMDP With Human Execution Under Uncertainty 72
5.3 Computing Human Model Parameters 76
5.4 Policy Computation for POMDP-HUE 78
5.4.1 Human Agent Policy Iteration(HAPI) 79
5.4.2 HUE Branch And Bound Policy Search (H-B&B) 80
5.4.3 Upperbound for Partial Policy Completions 82
5.4.4 Upperbound as Applied In H-B&B 85
5.5 Experiments and Results 87
5.5.1 Gridworld Experimental Setup 87
5.5.2 Gridworld Results 90

CHAPTER Page

5.5.3 Warehouse Worker Experimental Setup 93
5.5.4 Warehouse Worker Results 98
5.6 Human Subject Experiments 100
5.7 Related Work 104
5.8 Summary and Extensions 106
6 CO-PLANNING IN FACTORED STATE SPACES 107
6.1 Motivation 107
6.2 Related Work 108
6.3 Background 109
6.4 Problem Formulation 110
6.5 Embedding a Planning Domain 111
6.5.1 TGE-viz Graph Embedding Algorithm 112
6.6 Experiments and Results 113
6.6.1 Results and Analysis for Graph Embeddings 115
6.6.2 Mixed Initiative User Interface With TGE-viz 117
6.7 Summary and Extensions 121
7 TEAM TASK ASSIGNMENT 124
7.1 Motivation 124
7.2 Related Work 126
7.3 Problem Formulation 129
7.4 Solving APT Problems 130
7.4.1 Simulator 131
7.4.2 Search Algorithms 132
7.5 Experiments 133

CHAPTER Page

7.6 Results 137
7.7 Summary 137
8 CONCLUSION 139
8.1 Summary of Research 139
8.2 Recommendations for Computing Human Friendly Solutions 142
8.3 Avenues for Future Research 143
REFERENCES 145

LIST OF TABLES

Table Page

3.1 Average Time Taken for Varying Graph Sizes and Number of Terminal Nodes 36

4.1 Classification Likelihood Matrix (ϕ) for Warehouse Worker Domain, Where (S,m,l) Stands for (Small,medium,large) and “w" Means Bubblewrap Needed. Columns Sum to 100%. 57

4.2 Transition Likelihood Matrix; Rows Are States and Columns Are Actions. Recall Actions Maps 1:1 to States and So Have Matching Names. Each Entry is a Tuple of a Set of States From State Space S, and the Probability Of Transition to One of the States in That Set. Capitalized S is the Entire Set of States 58

4.3 Reward for State,Action Pairs; Rows Are States and Columns Are Actions. 58

4.4 Mean and Standard Deviation for the Number of Correct Actions and Total Actions Executed for the Simple and Difficult Policies 67

5.1 Policy Value Results for a 5x5 Grid. Each of the Primary Columns Changes One Experiment-Parameter, and Holds the Other Two Constant. Each Entry Has the Best Policy Value From Hapi, the Optimal Value Found by H-B&B, and the Ratio of the Two. All Values Rounded Down to Two Decimal Places. 90

5.2 Policy Value Results for a 10x10 Grid. Each of the Primary Columns Changes One Experiment-Parameter, and Holds the Other Two Constant. Each Entry Has the Best Policy Value From Hapi, the Best Upperbound Found by H-B&B After 30 Minutes, and the Ratio of the Two. All Values Rounded Down to Two Decimal Places. 91

5.3 Number of Nodes Opened by H-B&B Before Finding Optimal Solution, for a Policy Search Tree of Size 4^{25} In 5x5 Grid Experiments 92

5.4 The Total Time Taken (In Seconds) By HAPI, Followed by the Time Taken for H-B&B in 5x5 Grid Experiment Settings 92

5.5 The Total Time Taken (In Seconds) By HAPI, Followed by the Time Taken for H-B&B in 10x10 Grid Experiment Settings. H-B&B Was Terminated in 30 Minutes and Only the Tightest Upperbound Was Taken. 93

5.6 Transition Likelihood Matrix; Rows Are States and Columns Are Actions. Recall Actions Maps 1:1 to States and So Have Matching Names. Each Entry Is a Tuple of a Set of States From State Space S, and the Probability of Transition to One of the States in That Set. Note: Capitalized S Is the Entire Set of States 94

5.7 Reward for State,Action Pairs; Rows Are States and Columns Are Actions. 94

5.8 Classification Likelihood Matrix (p_c) For Warehouse-Worker Domain, Where (S,M,L) Stands for (Small,Medium,Large) And "W" Means Bubblewrap Needed. Columns Sum to 1000%. 95

6.1 Graph Metrics 116

7.1 Likelihood of Finishing Execution by Maximum Makespan for McTs and Tabu Search Given Different Configurations of Topological Depth (H), Number of Agents ($A = |A|$), Priority Levels ($P = |P|$)) And Maximum Makespan (M). 134

Table Page

7.2 Second Set of Results For Likelihood of Finishing Execution by Maximum Makespan for McTs and Tabu Search Given Different Configurations of Topological Depth (H), Number of Agents ($A = |A|$), Priority Levels ($P = |P|$)) And Maximum Makespan (M)..... 135

LIST OF FIGURES

Figure		Page
1.1	Dimensions of Human Involvement in SDM Problems	3
2.1	A Categorization of the Facets Pertinent to Human Cognition in SDM Problems	7
2.2	Increasing Levels of Realistic Approximation of Human Cognition; Resource-Constraints Approach Human Psychological Process Model. [43]	8
3.1	A Hospital Floor With Robot Navigation-Graph That Connects the Kitchen and Medicine Cabinet to All Patient Rooms. The Grid Squares Are the Vertices of the Navigation-Graph, and the Directed Lines Are the Edges; Only One Branching-Vertex (In Red) Has More Than One Outgoing Edge	16
3.2	A Hospital Floor With Robot Navigation-Graph That Uses Only the Shortest Paths to Go Between Kitchen and Medicine Cabinet to All Patient Rooms. The Grid Squares Are the Vertices of the Navigation-Graph, and the Lines Are the Edges; Every Vertex/Position Is a Branching-Vertex in This Example.	17
3.3	An Autonomous Guided Vehicle (AGV) Following a Path Laid Out in Tape	18
3.4	Example of Graph Minimization for Graph Size, and Minimization for Position-Based Predictability. Blue Vertices Are the Terminal Vertices That Must Stay Connected, and Vertices Highlighted in Red Are Branching Vertices.	26
3.5	WPC Measure With Varying Optimality Cutoff	34
3.6	WPC Measure With Varying Terminal Vertices	35

Figure Page

3.7 NV/NBV With Varying Optimality Cutoff 35
3.8 NV/NBV with varying terminal vertices 36
3.9 Average Suboptimality of Paths Chosen With Increasing Allowed-Suboptimality of Candidate Paths for GSC and BVC Cost Functions 37
3.10 Problems Given in the Human Subject Experiments; Problem 1 (Left) Has More Branching Vertices and Problem 2 (Right) Has Fewer. 37
4.1 Policies With Identical Value in an MDP, but Can Have Different Values After Accounting for Errors and Uncertainty Effects 45
4.2 Expected Value of Policies Generated With Varying ω in Gridworld . . . 60
4.3 Confusion Score of Policies Generated With Varying ω in Gridworld . . 61
4.4 Expected Value of Policies Generated With Varying ω in Warehouse Domain 62
4.5 Confusion Score of Policies Generated With Varying Ω In Warehouse Domain 63
4.6 Expected Value (Blue) and Confusion Score (Orange) of Policies Generated During Search in Gridworld 64
4.7 Expected Value (Blue) and Confusion Scores (Orange) of Policies Generated During Search in Warehouse Domain 65
4.8 The Simple Policy (Left), and Difficult Policy (Right) Given to Users to Execute 67
5.1 Additional State Added to MDP for State 1 to Account For Different Inference Likelihoods by the Human Agent 73

Figure Page

5.2 Example of a Partial Stochastic Policy (Left) Whose Probabilities for Every Action (A1,A2) Are the Same or Lower Than the Right Policy. "A^*" Is the Remaining Likelihood That an Action Can Be Assigned to (Optimizable). 83

5.3 The Policy on the Right–While Having Lower Total Probability Used Up–Will Not Have a Higher Value for Any Policy Completion Than the Optimal Completion for the Policy on the Left, if A1 Has a Higher Reward Than A2 Since the Probability of A2 Is Larger in the Right-Policy 84

5.4 Box Plot of HAPI Policy Values for Warehouse-Worker Domain With Varying Reward Noise Range; Values Normalized by the Policy Value From Branch and Bound Search. 97

5.5 Box Plot of HAPI Policy Values for Warehouse-Worker Domain With Varying Discount Factor (Γ); Values Normalized by the Policy Value From Branch and Bound Search 98

5.6 A Colored Gridworld Domain in Which an Agent Determines the States by the Color; Initial States Are Annotated as Well. 101

5.7 The Two Policies for the Second Phase of Human Subject Experiments. The Left Policy Is the Optimal Policy *Without* Considering the Effects of Uncertainty, and the Right Is the Optimal Policy After Accounting for Uncertainty. 103

6.1 Existing Visualization For Planners. Clockwise: SPIFe, Fresco, Conductor, WEBPLANNER 109

6.2 High-level algorithm for embedding a transition graph 112

Figure Page

6.3 Building the graph for embedding 113
6.4 Process to update the embeddings 114
6.5 Loop process for Procedure in Figure 6.4 115
6.6 Logistics Domain Embeddings 116
6.7 Barman Domain Embeddings 116
6.8 Plan Trace in Modified Logistics With Tge-Viz for the Goal of Delivering the Package to City 6 Location 3. 118
6.9 Alternate Plan Trace in Modified Logistics With Tge-Viz for the Goal of Delivering the Package to City 6 Location 3. 118
6.10 Interface For Collaborative Planning. 119
6.11 Filtering the Display to Only Propositions; Current State Propositions in Red, and All Others in Blue 120
6.12 Display of Action Information When an Action Node Is Highlighted . . . 120
6.13 Subplan Generated by Planner After User Clicks a Subgoal 121
6.14 Extending the Plan to the Next Goal From the Plan in Figure 6.13 122
7.1 Example of a Multi-Agent STN That Cannot Be Solved Without Preemption. 127
7.2 Categories of Rcpsp Problem From Operations Research Literature From [44] 128

Chapter 1

INTRODUCTION

The availability of troves of data, high-speed communication, and ubiquitous computing through the “Internet Of Things" [10] has resulted in large changes in the way our society produces and consumes. This is so much so that the term “Industry 4.0" [66] has been coined to denote a new industrial revolution (Industry 3.0 was the advent of computers into the workplace). Central to this revolution is the adoption and improvement of algorithms and methodologies from computer science [67]. As these technologies seep into more of everyday life –beyond their affect on industrial production– their impact on people has also come under scrutiny; this includes concerns about privacy [47][70], human performance/behavior[32], job automation[4], ethical/societal impacts of algorithms [108] have become important issues in public debate, research, and legislature. The need for better integration with humans has become a pivotal concern as automation becomes more sophisticated.

One of the ironies of automation [11] is that more advanced automation can also demand higher cognitive performance from humans in their roles as supervisors, teammates, or fail-safe operators. In order for Industry 4.0 technology –especially the algorithms underlying them– to interface better with people, we need to consider the limitations and preferences of people when designing or modifying existing algorithms.

Given the impetus to account for human limitations, the work herein focuses on how to incorporate such limitations and human needs into a set of Sequential Decision Making (SDM) problems and associated algorithms. This work is focused on the context of the human agent being an active participant (planner or actor) for improving or deciding outcomes in SDM problems; this is as opposed to a more

passive role such as an observer or consumer that might only require explanations or interpretable behavior, and not actively participate.

The difficulty of working with algorithms has received attention in single-step decision making; one such work is on classifier systems that considers how humans classify and gives explanations in a similar manner[21]. The use of classifiers in critical decision making, has brought algorithm inscrutability and biases under the microscope. A notable instance of this, is classifier use in the criminal justice system [111].

However, a lot of the emphasis in computer science research for usability from the algorithmic side has been in the context of the system explaining it's decisions or being interpretable [31], and less on other factors of human-machine interaction such as information load and attention demands. The work in this dissertation incorporates human-compatibility issues directly into algorithms and approaches in a set of sequential-decision making (SDM) problems. There is prior literature on humans in SDM problems; these typically incorporate humans as agents in the problem setting and use mental models with varying assumptions to mirror human limitations. We will discuss these modeling approaches in the following chapter. A general distinction between the approaches in this work and existing approaches to the problems tackled herein, is that we try to stay agnostic to the human computational model; we instead translate human cognitive limitations into resource limitations or problem-specific outcomes or models, and fold them into our algorithms. Another distinction with existing SDM research is that it is often the robot that works around the human's limitations or behavior, i.e. the onus is on the robot to carry or support the human. This ignores the question of how to compute policies *for* the human to execute, or how to empower them to perform better in SDM settings. This requires thinking about how human limitations interact with the problem and objectives; this is what

this dissertation focuses on. Each of the successive chapters will focus on a problem and make more detailed comparisons against the relevant literature for that problem.

On the topic of designing for human-compatibility, there is also a vast literature on human interfaces, and associated design philosophies [102][3] that consider how to represent information for humans. The field of cognitive ergonomics [51] broadly encompasses the study of how to design for humans (interfaces, products and the like). Research in this field has also identified the need for algorithmic advances that interface better with humans in order to improve adoption and efficacy, such as in algorithms for air traffic management [56]. We will borrow ideas from these related fields to apply in SDM problems.

When discussing the human's involvement in SDM, we study it as per the dimensions illustrated in Figure 1.1. We focus more on the active participation of the human

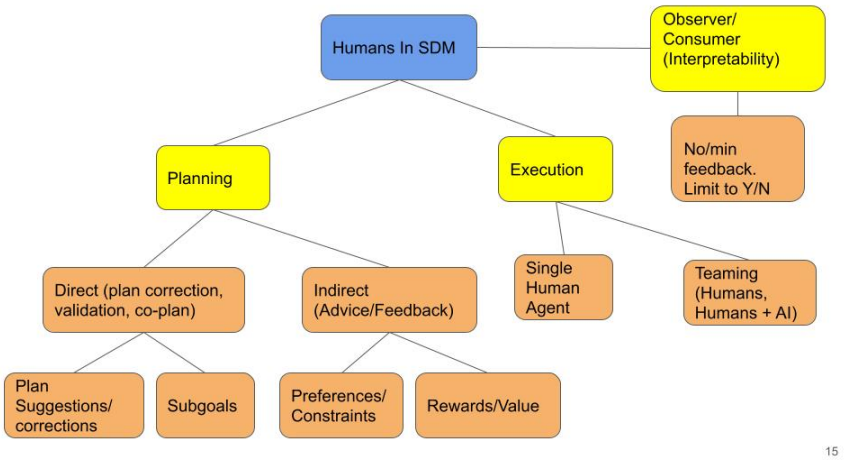

Figure 1.1: Dimensions of Human Involvement in SDM Problems

in the execution of policies/plans or in the planning process; not that the human just provides low-fidelity feedback (a more passive/consumer role). The primary focus of this dissertation is how to improve human involvement in planning and execution for SDM problems. There is existing work that considers humans in planning and

execution, but without emphasis on incorporating cognitive limitations for improving human involvement. We study how to improve human involvement through problems in motion planning, planning interfaces, Markov Decision Processes (MDPs), and human-team scheduling (presented in that order). The problems herein are :

- Robot motion in shared spaces with humans: We study the problem of constraining robot paths such that they are easily predictable with just the current position of the robot. This is to minimize the attention costs on the human, which will in turn reduce human effort for path planning in shared spaces (with robots).

- Co-planning in factored state space: In large factored state spaces, when a human and machine are co-planning, it can be hard for a human to bring to mind all the relevant alternative actions and pertinent state information. We tackle the problem of representing the state space and plans, and use principles from the interface design literature [102]. This work considers our memory limitations during co-planning, how to display pertinent information, and enable joint planning.

- Policy Complexity Minimization: When computing a policy for an MDP setting, if the policy is the same across similar states (by human perception), then the policy can be simpler or easier to follow. The reduced policy complexity can make execution faster and more accurate. This is what we explore with this problem, which considers our inferential limitations and attention.

- Policies that account for human response to uncertainty: When people follow a policy and they are uncertain about a state, they might take additional sensing actions to resolve the uncertainty and still make errors. Accounting for this

would help generate policies that work better with people. This problem considers a problem specific human behavior, i.e. our response to uncertainty when following a policy, and how it can be incorporated into the policy computation.

- Assignment and prioritization of tasks for human teams: The final problem we discuss in this work is on how to assign and prioritize tasks such that a team of humans –like software engineers– can better complete a set of interdependent tasks. We develop an approach to do so without constant central oversight that is typical of current planning and scheduling techniques, and explicitly account for preemption costs incurred when switching between tasks. This problem considers our cognitive costs when task switching, and how it factors into a plan for satisfying makespans (deadlines).

Each of the problems is given it's own chapter, in which we present the motivation for the problem followed by the it's formalization. In the motivation and problem formalization, we highlight the cognitive limits pertinent to each problem and how we tackle them. We then describe the methodology employed for each problem, and experiments done to validate our methods. We also discuss in each chapter, a comparison with related-work and how our emphasis on cognitive limitations distinguishes our work from others. Finally we present avenues for continuing the research direction in that chapter. Before we delve into these problems, we first discuss the pertinent aspects of human cognition (with respect to SDM) that has been explored in existing literature, as well as what aspects need more consideration.

Chapter 2

MODELING HUMAN COGNITIVE LIMITATIONS FOR SDM

The relevant components of human cognition for SDM can be broadly categorized into memory, attention , inference, and performance affectors. The last one is a catch-all for factors like the impact of stress, frequent task-switching, preferences/biases, and the like. The components are illustrated in Figure 2.1 and we will refer to these components and their relevant limitations –such as limited capacity for short-term memory– when discussing each of the problems we explore. While the components are displayed as separate categories, they do in fact affect each other; for example, inference is affected by attention, and together determine situational awareness [110] of a system or setting. This categorization of cognitive components is to help us to think about the different parts involved in cognition, and subsequently help us think of how aspects of a problem setting are connected to these components. Each component can be thought of as a resource with a finite capacity; a person has a finite capacity for attention, working-memory, capacity to juggle different tasks, and so forth. The primary approach taken in this work is to optimize for, or limit the demand on these cognitive resources. This is as opposed to building a model of the human agent to simulate behavior. We will shortly explain our reasons for choosing this direction by discussing the modeling assumptions in the existing literature and their shortcomings.

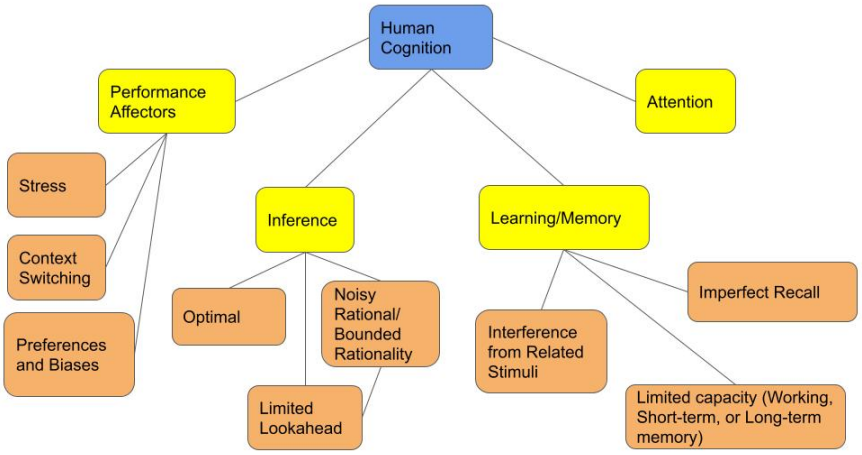

Figure 2.1: A Categorization of the Facets Pertinent to Human Cognition in SDM Problems

We first note that the idea of considering cognitive resource limitations when approximating the human computational process is already present in cognitive science literature [43]. From said literature, figure 2.2 discusses one way of thinking when approximating human cognition; one can approach the psychological process of humans by starting with an optimal computation process and increasing resource-constraints on the computational model. With this in mind, we discuss the assumptions and gaps with current approaches.

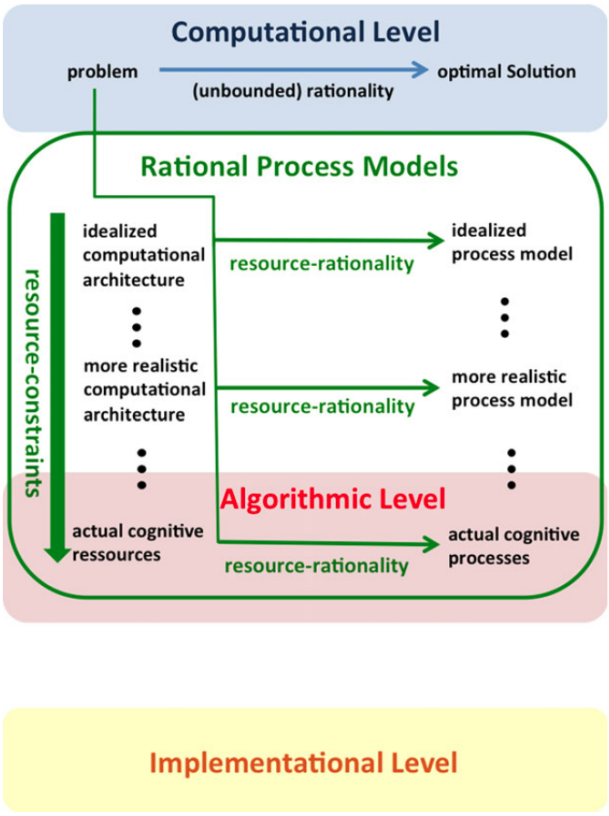

Figure 2.2: Increasing Levels of Realistic Approximation of Human Cognition; Resource-Constraints Approach Human Psychological Process Model. [43]

We start with existing assumptions about human inference and decision making. The baseline assumption is to consider humans as rational agents and that they take the action or choice that is optimal in the problem setting. This is often used as a starting assumption when incorporating humans into problem settings, or when focusing on the knowledge (model) differences to explain differences in inference [20]. An improvement on this assumption is bounded-rationality [93] which comes from behavioral economics and has been adopted in computer science literature as well. One models human decisions as being “bounded-rational" [68], sometimes also called “noisy-rational" or “Boltzmann rational". It is so called since the likelihood of a

decision is determined by a Boltzmann distribution where the typical energy term is replaced by the value of the decision. Another variation of bounded-rational behavior modeling is in work done by [112] that tries to model human agents as interleaving planning and execution; their variation on bounded-rational is through limiting the look-ahead in planning, since humans are limited in their capacity to rollout a full plan to a goal.

One can also learn a controller for mimicking human decisions based on samples of human action-trajectories. The generalizability of such a controller would naturally be dependent on the similarity between the test conditions and the training data, as well as the state representation of the controller. While it seems like a good idea, it maybe arbitrarily bad as recent work by [8] has shown. In the previously mentioned work, the authors show that (verbatim)" *(1) No Free Lunch result implies it is impossible to uniquely decompose a policy into a planning algorithm and reward function, and (2) that even with a reasonable simplicity prior/Occam's razor on the set of decompositions, we cannot distinguish between the true decomposition and others that lead to high regret. To address the issue of determining an appropriate reward or planning parameters of a person, we need simple 'normative' assumptions, which cannot be deduced exclusively from observations.*". These statements were made in the context of learning human preferences or reward functions from behavior. The normative assumptions refer to assumptions on the human's planning or reward that are needed to decompose the reward correctly and cannot be inferred from observations alone. The point we want to highlight here, is that a parameterized agent-model for representing humans that is fitted to trajectories is not a panacea (won't necessarily generalize well); one needs to make and use specific assumptions about human behavior for a particular setting. On top of this, one needs to remember that for some behaviors (policies), using just Markovian rewards would be inadequate to ex-

plain/capture a set of policies or trajectories [1]. So blind reward fitting is a bad idea.

Another issue with modeling humans is that research has shown humans deviate from optimality in systematic or non-random ways [55], so noisy or bounded-rational, and optimal inference models do not seem defensible. There has been work done that tries to accounts for systematic deviations when learning preferences [33], but this too would run into the same problem discussed in [8]. One may need instance specific assumptions and expectations of human behavior as espoused in [8]. Lastly, with respect to suboptimal-inference models of human agents, there is no explicit consideration given to memory limitations, attention limitations, or stress as causes of suboptimality, or how handling them can improve outcomes.

An example of an assumption when trying to predict how humans will act is risk-aversion, which would fall under "Performance-Affector" in the categorization in Figure 2.1. There is some work in the computer science literature that considers humans as agents preferring options with lower risk [64] and that may explain the lion's share of decisions in some problem settings. This is inline with the type of considerations we make in this document, and in Chapter 5 we make such an assumption on how humans might respond to uncertainty.

Other than human inference limitations, there is work that considers other cognitive limitations in existing computer science literature. One example of this is the approach in [79] and [78] which considers memory-limitations when predicting human decisions in a Partially Observable Markov Decision Process (POMDP). The authors state that the bounded-memory modeling decision is an instance of bounded-rationality, except the suboptimal behavior (state inference) is due to limited memory or recall biased towards recent events as opposed to inferential errors. Another work that considers a different human limitation is by [82]; the authors develop an approach

to account for human attention errors. In their approach, a model of systematic errors in feature detection (blindspots) is learned and used to determine when to transfer control to a human agent, and when not to.

In this work, we consider human limitations but we do not focus on explicitly modeling the human computational process. Rather, we develop algorithms and approaches that reduce demands on cognitive resources, or incorporate systematic human errors, while remaining agnostic with respect to the underlying human computational process. An analogous idea from the existing literature, is the work on Sparse Linear Integer Models (SLIM) [99]. This considers how humans work better with integer weights in linear models. The pertinent part of the problem that made cognitive effort harder was the granularity of the model-weight values; more decimal places made comparison and inference harder. SLIM was in turn inspired by the Apgar scoring system [18]. In that system, nurses would compute a score to evaluate the health of new-born infants by summing up integer-value scores associated with easily detectable features. The score would then determine the medical response (policy action).

Similar to the aforementioned approaches, we identify the key problem-aspects of certain sequential decision making problems that affect the cognitive demand made on the human agent. This can be the number of state-features for the human to pay attention to, or the memory required for tasks. These aspects are then explicitly incorporated into the problem as constraints or objectives. By staying agnostic with respect to details of the human computational process, we can help human actors while avoiding the pitfalls of (mis)fitting a cognitive model.

Chapter 3

MOTION PLANNING IN HUMAN-ROBOT SHARED SPACES

One of the guidelines in the work by [32] is to minimize the number of logical branches in a task in order to handle complexity and make interaction easier. We apply this to robot motion in spaces cohabited by humans in order to make robot motion more predictable. The SDM problem tackled here is path prediction, and we consider how limited human attention, and computational capacity can be accounted for when designing how a robot should move.

3.1 Motivation

When humans move in the same space together, we are often able to do so with little effort, and often make normative assumptions about other people based on an (often correct) shared theory of mind. It maybe a lot harder to infer robot motion, especially if they do not have the same modalities like other humans; the robots maybe trolley bots on wheels and not have a head and torso. So when robots cohabit the same space as people, easy predictability of robot motion becomes very important to allow the humans to co-navigate in the space. This is the work done in [40] and is the focus of this chapter.

To motivate this problem, let's consider multiple humans and robots moving in settings like hospitals or restaurants. When humans and robots are moving in the same space, it can get very challenging for both to move seamlessly due to the difficulty in mental modeling the other. If there are multiple robots, motion planning is made much harder for the human. For the human to plan their path, predictability of other agent's (robot) motion is very important.

The approach herein to this problem is built on a simple premise; If the human is given only the current position of the robot, then the more the number of possible future trajectories, the harder is the problem of bounding the robot's future positions. As in, the possible future locations of the robot increases, and this consequently makes it harder for the human to navigate in the space without potential path conflicts. Bounding the robot's motion is possible by constraining the possible trajectories a robot can take from any position, and minimizing the areas in which the robot can move.

One might ask, why not make the robot handle all the navigation effort? This is the direction of some of the existing research which has shortcomings such as the robot freezing [97] or moving haphazardly from being unable to compute a path that conforms with the motion of multiple people moving in the space.

We take a different approach from the trends in existing literature. If we assume the robots have a fixed set of tasks –such as serving food between the kitchen and tables, or transporting medicine between a cabinet and patient beds– then we could compute a restricted *navigation-graph* to make navigation easier for both humans and robots in the space, and keep each task's motion path costs within a multiplicative bound of optimal path costs. Such an approach can also reduce the hardware and computation requirements for the robots in the space. Limiting motion like this is what was done with Automated/Autonomous Guided Vehicles (AGVs) which are already in use for industrial settings, and even hospitals with mature technology [34]. AGVs followed a predefined path, often laid out with tape (like rails for a train) to move between positions (see Figure 3.3 source [85]). The AGV's grid (which determine what paths can be taken) were laid out for the sake of optimizing task output. In this paper, we propose computing the AGV grid (as a directed graph) for more predictability of trajectories by humans. Predictability is very important

for human robot interactions [61] and more predictability would improve adoption of AGV's in everyday settings like restaurants.

Constraining the robot's navigation-graph –where and how robots can move (like in Figure 3.1)– and communicating it to the humans in the space by markers or tape can help in two ways. One way is by simplifying the path planning of the robot; it is now helpfully limited to computing paths on the navigation-graph. Another way it helps is by making the robot's motion more predictable to the humans. It does this by limiting the space the robots can occupy and how they can move. This helps reduce the complexity of, or negate the need for mental modeling of the robot by the human. People can see that the robot is limited to following the laid out markers. Humans that know the path of the robots tend to help in the navigation by simply not blocking or adapting their movement to avoid conflicts, as humans in the space can be seen as non-competitive or collaborative. This idea of modeling the human as working with the robot in navigating the space was used in [97], [60], [77], [6].

The involvement of the humans in effective robot navigation is almost necessary when dealing with many humans and robots in the same space as the robot might freeze [97] [98] or move haphazardly because it was unable to compute a path that is conformant with the motion of all humans in the space. This is why predictability of robot path is very important, as it enables easy and effective involvement of the humans in the space.

This work focuses on the computation of the navigation-graph for robots to make predictability as easy as possible, and with minimal information. Minimal information and effort is important as we cannot expect the human –who has their own tasks– to invest non-trivial cognitive effort in predicting motion. This is why we focus on *position-based predictability*, where the future trajectory of the robots can be predicted or bounded from the current position alone. Since predictability is deemed to be a very

desirable quality in robot navigation from a human factors perspective [61] it would affect adoption or acceptance of robots in any setting. With respect to optimizing for cognitive resources, which is the thrust of this proposal, we reduce the attention and inference costs for the human. By reducing both, we reduce the situational awareness demands [110]. Situational awareness is broken into three levels in [110]: level 1 is perception of situational elements ; level 2 is information integration; and level 3 is projection of future status and actions of situational elements . We make level 1 and 3 easier as the perception information needed is just the current position, and projection of future states is made easier by limiting the robot motion and displaying the navigation graph/grid on the floor (AGV tape).

In Figures 3.2 and 3.1 a nurse is moving in the space with robots. Certain regions are blocked out (black) due to furniture, walls or other motion constraints. If the nurse (in the example) is moving down towards a patient's room in the middle, then in the case shown in Figure 3.2 the nurse wouldn't know how the robots would move and cannot easily come up with a non-conflicting path. The robots might also reverse their direction if their task priorities change. In Figure 3.1 the nurse can easily infer that the robots will only be moving away and can walk straight to the patient door. This is because the navigation-graph has only 1 "Branching-Vertex" (position), which are vertices with more than one outgoing edge. In Figure 3.2, every position is a branching-vertex for the robot's navigation and so the possible fewer positions are many.

Fewer branching vertices mean fewer possible trajectories from any position and so makes prediction or bounding of robot motion easier. This comes at the expense of path costs between positions associated with tasks (called terminal vertices). Reducing branching vertices while keeping path costs within acceptable bounds are the main objectives of our approach. This corresponds to one of the good design prin-

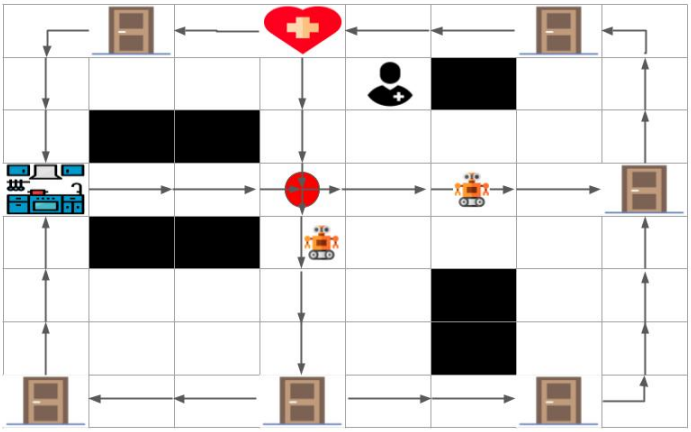

Figure 3.1: A Hospital Floor With Robot Navigation-Graph That Connects the Kitchen and Medicine Cabinet to All Patient Rooms. The Grid Squares Are the Vertices of the Navigation-Graph, and the Directed Lines Are the Edges; Only One Branching-Vertex (In Red) Has More Than One Outgoing Edge

ciples suggested by existing literature on how to design for humans by [32]; in that work they suggest to "minimize logic branches". Specifically (verbatim) *"Minimize complexity by reducing the linkages and conditional operations contained in the autonomy, avoiding modes with their multiple-branch logic as much as possible. "*. In our approach to this problem, that guidance translates to minimizing the number of branching vertices in the robot's navigation graph.

If we compute the robot's navigation graph with fewer branching vertices, we can use it to layout the path of Autonomous Guided Vehicles as in Figure 3.3. This makes the path of the AGV easier to predict for the human with less attention and information.

In this chapter, we will first discuss the relevant work on human-robot interaction with respect to motion. Then we will formalize the problem of computing the navigation-graph for position-based predictability, as well as introduce measures for

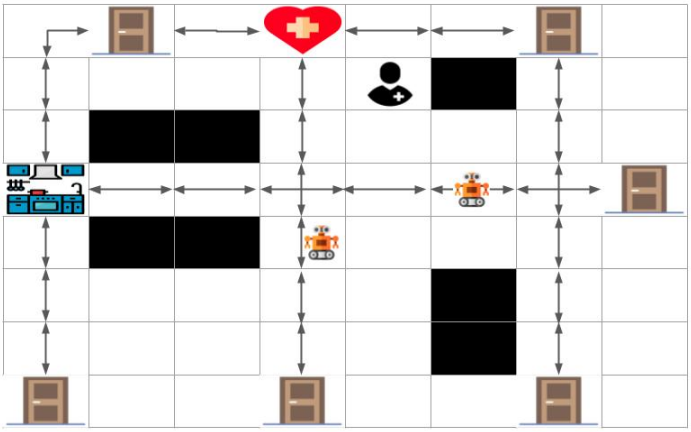

Figure 3.2: A Hospital Floor With Robot Navigation-Graph That Uses Only the Shortest Paths to Go Between Kitchen and Medicine Cabinet to All Patient Rooms. The Grid Squares Are the Vertices of the Navigation-Graph, and the Lines Are the Edges; Every Vertex/Position Is a Branching-Vertex in This Example.

position-based predictability. Finally we will talk about a hill-climbing approach to solving the problem and empirical evaluation for it.

3.2 Related Work

In the area of robot motion planning with humans in mind [29] [30], Dragan et al. assume that the robot has a model that can help predict what motion the human will infer, and move predictably according to that model. Along a related line of thinking [57] considers how the robot's actions communicate intent to the human, and use it for better collaboration. An implicit assumption in such work is that the human pays non-trivial attention to the actions of the robot to make this inference, which is defensible when the human and robot are doing the same or related tasks, or in a sparse setting like at home. In our settings of robot motion (hospitals, restaurants, banks), we expect the humans will at most glance at the robot or use peripheral vision

Figure 3.3: An Autonomous Guided Vehicle (AGV) Following a Path Laid Out in Tape

to determine where it is (not invest time intently observing the robot); the humans would be preoccupied with their own independent goal-directed behavior. There is also work [13] that considers inferring the hidden mental state of humans to infer their intent with POMDPs and use that to coordinate better. In all of the aforementioned work, if there are many people moving in the space, then it may be computationally intractable or just not possible for the robot to act and conform to all of the humans' intents or mental models. This leads us to the work on robot motion around larger groups of people.

In the literature that tackles robot navigation in groups/crowds, the approaches taken include modelling group dynamics using Gaussian processes [97] to infer how the crowds will move use it for navigation. Naturally this would require more hardware to sense and compute crowd dynamics and it is not clear how effective this actually is; to our knowledge, the approach has only been tested on constrained simulations. A more pressing concern is if there are any useful or predictable crowd dynamics in settings like hospitals. Such dynamics may exist in busy pedestrian crossings where there are general directions of motion. We think such predictable dynamics are unlikely to exist

in settings like hospital where movement is dictated by individual hospital staff goals which can be very hard to predict. Another concern is how seamless the motion will be as the predictability of motion is known to be important from a human factors perspective [61], this was not evaluated in the referenced work on crowd modelling with Gaussian processes [97]. We expect people moving around the robot without consistent dynamics would result in jerky, inconsistent movement. This leads to inconsistent signals given to the human, and hurt navigation coordination with the human. If robot goals or priorities can change dynamically, then the predictability of motion is worsened. In the approach we support, by restricting the robot to simply following the markers on the floor, which communicates a directed navigation-graph, there is much less uncertainty on how the robot will move. The only source of uncertainty is in how the robots will move/turn at branching vertices, and by minimizing the number of these, we further improve predictability.

Other methods for navigation in crowds include using Inverse reinforcement learning [59] [100], and Deep reinforcement learning techniques [23], [22] which also consider human gaze in the feature set. With IRL approaches, if motion dynamics of people change –which can happen with changes in furniture and space rearrangement– then the relearning time is not considered in such work. With our method, if the space configuration changes do not overlap with the navigation-graph, we can safely ignore it. If they do, we simply update the initial graph with the new space configuration (and capture which vertices and edges are blocked), and compute a new navigation-graph and replace the old one's markers/tape. Additionally, it is not clear how predictable the motion with IRL techniques is, as that aspect was not evaluated. Similar problems like motion predictability and robustness to environment reconfiguration would plague deep-RL methods([23], [22]). In these, experiments were done in simulations, and no consideration was given to the predictability of the motion

by humans. In contrast, AGV navigation using tape laid on the floor has a history of real-world use in hospitals and industrial settings. To bring this approach to more settings, we take the position that further improving predictability of motion is paramount for adoption in more commercial settings. The approach herein is about how to compute the navigation-graph that is laid out, such that motion is very easily predictable. This allows the humans to move easily in the space even with multiple robots, as they know the robot's hard motion constraints; they do not have to build much trust in the robot's navigation and collision avoidance abilities. This is likely to matter more with bigger robots, as they are perceived as more of a threat to physical safety [50]. On top of improving predictability, one can incorporate any additional advancements for soft robotics [9], collision avoidance, and human gaze detection for use in further improving the safety and acceptability in everyday settings.

Our approach also connects to recent literature on environment design, in which the environment is configured to make prediction by humans easier [62]. Obviously, we do not explicitly reconfigure the environment; we only limit the environment in the robot's model for its motion. In environment design work on predictability with respect to the human, one assumes the goal is known to the human, or that the human has observed a prefix of a plan's actions and can infer the rest assuming they can infer optimal path completions. These are assumptions that we want to remove for the settings we consider (restaurants, banks, hospitals). Humans in such spaces are not paying significant attention to the actions or know the set of goals of the (possibly) multiple robots moving, nor can they be expected to invest effort computing optimal path completions to these goals. There can also be new humans entering the space regularly with no familiarity. So making predictability as trivial as possible, and from just the current position (as AGVs do) becomes important.

With respect to the computation and algorithmic aspect of the approach herein,

our work computes a navigation-graph in a problem setting similar to Strongly Connected Steiner Subgraph(SCSS) [35] problem. Our optimization for position-based predictability is done with two objectives: (1) minimize the space (vertices and edges) used by the robot in it's navigation between terminal vertices; (2) minimize the number of branching vertices, and thus possible paths that a robot can take from any position. The former is what the SCSS problem does, but to the best of our knowledge no variants of SCSS explicitly considers minizing branching vertices. There are graph problems that consider branching vertices, but with only a single source node as far as we know; this is the problem of Directed Steiner Tree with Limited Diffusing vertices (DSTLD), which has applications in multicast packet broadcasting [105]. Solutions to DSTLD optimizes for the total edge-weight cost of the tree (Steiner tree problem), *and* limits the number of branching vertices (which they call diffusing vertices) but only from a single source node. They do not consider the problem of optimizing the number of branching vertices for paths between multiple source and destination vertices (which we do). There is also no thought given for individual path costs between vertices (only total edge weight). Another such problem in this vein of literature, is the Rectilinear Steiner Arborescence (RSA) [84] which is the rectilinear version (manhattan distances) of Directed Steiner Tree problem . In RSA, the objective is to compute the minimum length (sum of edge cost) directed tree, that is rooted at an origin point and connected to N vertices. The problem is known to be NP-complete [91] to compute a solution that is less than a specified length, and so NP-hard to optimize (minimize) for the total length. RSA (like DSTLD) is just for a single-source, and pays no heed to pairwise path costs between terminals.

3.3 Problem Formulation

The problem of graph minimization for position-based predictability is given by the tuple

$$P_{min} = \langle G, T, C, W \rangle \tag{3.1}$$

where

- G is the directed graph defined by its vertices and edges *(V,E)*. This graph captures the all the allowed motion of the robot. Any motion constraints like protected areas or one-way corridors are captured in the graph as well (dropped vertices, and one-way edges).

- T is the set of ordered vertex pairs that need to remain connected in the minimized graph. Each pair appears twice (both orderings of the pair) to represent both directions of connectivity. The vertices will be called terminal vertices, and we refer to the ordered pair of vertices as a "task", as we think of them as being associated to a task. These are locations of importance for a task like a medicine cabinet, or it could be the door connecting to an adjacent room that the robot needs to reach.

- C is the function that returns the cutoff cost (maximum allowed) distance associated to each task in T. The cost of the paths in the final minimized graph should be less than the specified corresponding cutoff.

- W is a function that returns the weight of each pair in T. This weight can be the probability of the task, or an importance weight (if some tasks need to be done more quickly).

The objective is to reduce the input graph G so that in the resultant (output) navigation-graph, position-based predictability is easier, *and* the path costs between

terminal vertices are above a threshold defined by the cutoff function C. Often a trade off will be necessary between path costs and predictability. Measures for position-based predictability are formalized in the following section. Note that we do *not* expect the human to predict the entire path suffix from the current position of the robot; people certainly do not predict other pedestrians' full trajectories before moving. Rather, we want to make it easier to bound the possible paths or positions that the robot might occupy. This information would help the human decide where to move for their next steps, and if the robot is likely to move in the same spaces.

3.3.1 Measures for Position-based Predictability

We describe herein measures for position-based predictability and justify their definitions.

Weighted Prediction Cost (WPC): If the navigation-graph has fewer branching vertices, and each of them having a smaller branching factor (number of out-edges), then it makes the motion more predictable for the human; fewer possible future robot trajectories for the human to consider. Vertices with only one outgoing edge require minimal thought, and trajectories with a sequence of vertices with just one outgoing edge are trivial to predict from just the current position of the robot; these are ideal conditions for predictability (trivial cost).

We first describe the WPC measure before defining it. There are two components in WPC, the first counts the number of branching vertices that appear on the paths for the robots tasks, and weights it according to the values given in W. This is then multiplied by the sum of the branching factor of those branching vertices. Two terms are necessary as graphs might have the same sum of branching factors but different number of branching vertices or vice versa. We need both the number of branching vertices to be fewer *and* their branching factor to be smaller to make

predictability easier. 3 branching vertices with branching factor 2 is treated as worse than 2 branching vertices with branching factor 3. This is because every branching-vertex breaks the ideal of just one trajectory for as many steps as possible. If the human has to cross one of the robot's paths after a branching vertex, the human may take a different path, or slow down to see how the robot turns (assuming they want to avoid conflict like in [97], [60]). Such a situation would occur more often with more branching vertices. So the count matters as well as the sum of branching factors (number of trajectories). The branching factor matters because fewer trajectories lets the human plan a path around any of the future trajectories more easily (less space covered). A smaller WPC implies a simpler graph of robot motion for position-based prediction. Zero WPC, meaning zero branching vertices implies the robots movements are all trivially predictable since there is only one action (direction) it can take from every position. WPC is defined as :

$$WPC(G,T,W) = \sum_{t \in T} \sum_{v \in SPV(G,t)} W(t) * 1[deg^+(G,v) > 1] \times \sum_{t \in T} \sum_{v \in SPV(G,t)} W(t) * deg^+(G,v) \quad (3.2)$$

where $SPV(G,t)$ returns the shortest path vertices for the input task (t) in the graph (G) being evaluated, and $deg^+(G,v)$ returns the outdegree of the vertex(v) in the graph(G). If there are multiple shortest paths for a task, then the path that contributes the least cost to WPC is used. What this translates to–in terms of the robot's motion–is that when the robot is moving on the path associated to T, the human observing it at any position will have to consider fewer (ideally only 1) possible future trajectories when trying to move and avoid any conflicts.

Weighted Ratio *NV/NBV*: where NV is the number of vertices and NBV is the number of branching vertices. This measure represents the number of vertices or

steps in a sequence with no branching, and is weighted by the task weights. If the weights are just the probabilities of the tasks, then this measure returns the average number of steps before encountering a branching-vertex. So a larger NV/NBV value implies longer paths with no branching, i.e. going one-way for longer distances; this makes path predictions using only the position trivial for many parts of the graph. NV/NBV is formalized as:

$$NV/NBV(G,T,W) = \frac{\sum_{t\in T}\sum_{v\in SPV(G,t)} W(t)}{\sum_{t\in T}\sum_{v\in SPV(G,t)} W(t) * 1[deg^{+}(G,v) > 1]} \quad (3.3)$$

What this means for the human in the space, is that they don't need to consider multiple paths for moving shorter distances. They just need to see that in the next "N" steps or units of time, the robot will not be near the human. This was illustrated in Figure 3.1. If the nurse was moving to the middle door at the bottom, then the human knows with a glance that the robots will only ever move further away from the human and never towards them, or cross them (as per the navigation-graph). The human doesn't need to infer the full path of each of the robots, which would require unnecessary effort. Even knowing both the robots' goals doesn't necessarily make the cognitive effort minimal. For the human (the nurse) to reach their goal (middle door) they just need to know or bound the immediate next few steps to determine a uninterrupted path to their goal (middle door). Any additional effort is unnecessary, and thus undesired.

An example of the type of navigation-graph optimization that this work does is shown in the rectilinear grid-graph in Figure 3.4. The blue vertices are the terminal vertices and vertices highlighted in red are branching vertices. In the figure, we show two graph minimizations; one that focuses only on graph size (number of vertices and

edges), and another minimization that optimizes for position-based predictability (fewer branching vertices and graph size).

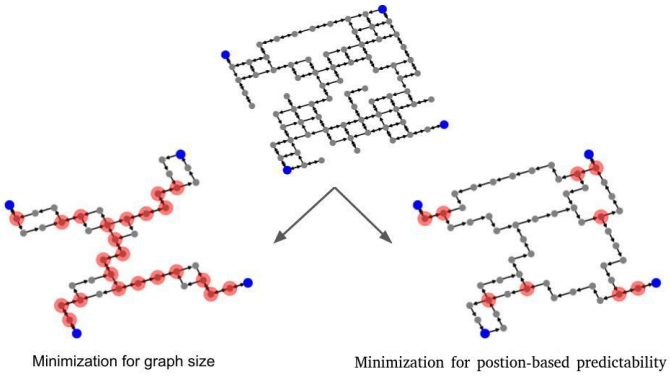

Figure 3.4: Example of Graph Minimization for Graph Size, and Minimization for Position-Based Predictability. Blue Vertices Are the Terminal Vertices That Must Stay Connected, and Vertices Highlighted in Red Are Branching Vertices.

Both these measures involve branching vertices. We provide supporting evidence for our intuition in the human subjects experiments that fewer branching vertices make position-based predictability easier.

3.4 Methodology

Our problem of (navigation) graph minimization for position-based predictability is one of constrained optimization. We want to find a graph that minimizes WPC and has a high value for NV/NBV, while connecting all the required pairs of terminal vertices in T within the cost constraints specified through C. To find such a minimized graph, we used a hill-climbing search approach. The two key components in our approach are (1) finding a diverse set of paths for each task (ordered pair of terminal vertices), and (2) evaluating (for comparison) the different graph candidates during the hill climbing search.

Before we go into the details, we highlight a challenging aspect of this problem which motivates our methodology choice. Specifically, the number of possible combination of paths between terminal vertices for a graph can explode in size based on the number of vertices and graph-connectivity; denser implies more path options. The difficulty in optimization comes from the fact that we cannot choose a path between terminal vertices independent of the other paths. This is because the number of branching vertices is not known until we combine the paths for all pairs of terminal vertices. This leads to a combinatorial explosion in the number of possible path combinations (and thus graphs), and so makes it challenging to compute the optimal graph.

To make the computation tractable, we limit ourselves to using a fixed number of paths for each task in T. This initial population of paths determines the quality of the outcome of the search, and choosing this population is the first step of our approach. In this regard, one set of helpful constraints are the cost cutoffs for the paths. This helps constrain our search space in terms of the paths to consider. The paths are obtained through shortest-path search for each ordered pair of vertices in T. After one path is found for a task, the weights of those edges are doubled and the process is repeated to help build a diverse set of alternative paths. After we build the candidate pool of paths per task, we start the hill-climbing search.

This brings us to the other part of the search process, and that is generating and comparing successive graph candidates. For generating new candidates for a given graph in the search, we only consider single-path replacements to the graph. We do this using the population of paths we stored in the first step. This is how new candidate graphs are generated during the search. We may replace or just drop a path if there exists another path in the candidate graph –using edges from other paths– that is within the cutoff. If a replacement breaks the connectivity between any

terminal vertex pair, it is not considered. We compare the cost of all the candidate graphs produced at each step, and greedily take the candidate with the lowest cost (we will discuss cost functions used for comparison soon). If no better candidate exists at a step in the search, we stop the search process. Lastly, we use a fixed number of random restarts (5 restarts) to repeat the hill climbing process, and use the best result over all. Each iteration starts with a graph that is a combination of randomly chosen paths for each task.

As maybe apparent, the cost or evaluation function is pivotal to the algorithm. We test our hill climbing approach with two relevant cost functions that are intended to minimize for the effort of an observer for position-based predictability. We start with discussing the baseline cost function, which we then modify for our proposed (main) cost function. The baseline cost function is simply the sum of weighted vertices and edges. We will call this the *Graph-size cost* or GSC. It is so called since a smaller graph with fewer vertices and edges would have a lower GSC cost. GSC is analogous to the optimization objective of the SCSS problem(see related work for SCSS description). Based on our correspondences with the authors in [35], [24] the SCSS algorithms therein were theoretical and not implemented. The connection with SCSS problem was part of the motivation for setting GSC as the baseline, as well as the fact that making the navigation graph as small as possible ought to help predictability; this stems from the intuition that a smaller graph could result in fewer possible paths to consider.

One difference between how we use GSC and the SCSS problem is that SCSS does not consider individual path costs between terminal nodes, which is important for our navigation problem; during search in graph space we ensure that the path between any pair of terminal nodes doesn't exceed the specified cutoff. The other difference with SCSS, is that the weights of the nodes and edges is affected by the weight of

the tasks on whose paths they appear in (in the computed subgraph). So the weights are dynamic unlike in SCSS. The importance of this is apparent when we consider the task weights as task frequency. The paths associated to more frequent tasks are taken more often, and so those paths should be more predictable. The weights are given in the problem input W. GSC is defined as:

$$GSC(G,T,W) = \sum_{e \in E(G)} \max_{\{t | t \in T, e \in SPE(G,t)\}} W(t) + \sum_{v \in V(G)} \max_{\{t | t \in T, v \in SPV(G,t)\}} W(t) \quad (3.4)$$

where $V(G)$ returns the vertices in the graph, $E(G)$ returns the edges in the graph, and $SPE(G,t)$ returns the shortest path edges for the task(t) in the graph(G), and $SPV(G,t)$ returns the shortest path vertices for the input task (t) in the graph (G).

We use the maximum of the task weights of the tasks whose shortest path uses that edge or vertex. This encourages the search to prefer graphs that reuse the same vertex or edge in the paths for multiple task. The drop in cost would be greater when paths of tasks with higher weights use the same edges and vertices. Note that while GSC doesn't explicitly consider branching vertices, it would implicitly favor fewer branching vertices when possible; a branching-vertex increases the number of edges for every step after it, as opposed to if the path didn't branch. This would increase the edge cost part of the GSC score. So any approach to minimization for position-based predictability should be no worse than this baseline's performance for the predictability scores defined in the problem formulation section.This brings us to our cost function which builds on GSC to explicitly incorporate branching vertices. We will refer to this cost function as *Branching-Vertices Cost* (BVC). The function

is as follows:

$$BVC(G,T,W) = WPC(G,T,W) * GSC(G,T,W) \tag{3.5}$$

where WPC is as defined earlier in Equation 3.2

Graph minimization with BVC as compared to minimization with GSC will tell us if simply minimizing the graph for the number of vertices and edges (GSC) is sufficient to optimize for our position-based predictability measures, or if it is better than BVC in human-subject studies. If so, then our BVC cost, that explicitly considering branching vertices, would not be necessary; the need for considering branching vertices is central to this work. GSC gives us a plausible baseline to use in human-subject experiments for position-based predictability. It represents the stance that minimizing just for the size of the graph is enough and ignoring branching vertices doesn't hurt predictability.

Note that we do not directly optimize to increase NV/NBV as that biases the search to use longer paths in the resultant graph for getting a higher value in this measure. The cutoff input in C is the *upper limit*, but we think it desirable to keep the paths as short as possible. So we do not explicitly incorporate the NV/NBV measure into the cost.

3.4.1 Computational Complexity

For our problem over rectilinear graphs, it is hard to compute optimal solutions as even a special case with all tasks having the same (single) source vertex is NP-Hard to optimize even if we only consider minimizing the edge-cost in the graph; that special case is the Rectilinear Steiner Arborescence (RSA) problem, which is discussed in the related work. So we resort to approximate methods like hill-climbing.

With respect to our algorithm's complexity, for the first step of computing the

population of paths for each task in T, the entire subroutine should take O(PopulationSize) * (|E|+|V|log|V|)); where (|E|+|V|log|V|) is from the running time of shortest path using Dijkstra's algorithm. The "PopulationSize" is the size of the set of shortest paths for the tasks used in the hill-climbing search. We will also present empirical results for the time taken in Table 3.1 in the Results Section(VI).

Since our approach is a hill-climbing approach, it is an anytime algorithm. We can stop at any point and get the best solution found until then. With more random restarts and letting the algorithm run longer, one can get better solutions as is expected with hill-climbing. This computational cost is one-time when setting up the space, and only needed again if the space configuration changes.

3.5 Experiments

To evaluate the graph minimization with our hill climbing algorithm, we run our algorithm on randomly generated 20x20 grids. This is intended to reflect a room or hall of a building. The robots have to move through this space between terminal vertices; terminal vertices could be doors connecting to other rooms, or locations relevant to a task. We start with a fully connected grid with adjacent vertices connected bidirectionally. All edges are of unit distance. Then we randomly drop 20% of the vertices and 20% of the remaining edges; dropping vertices and edges represents the spaces used for furniture, obstacles, corridors, walls etc. Lastly we arbitrarily select the terminal vertices from the remaining vertices. All ordered pairs of terminal vertices define the set T. We vary the number of terminal vertices selected from the set $\{3, 4, 6, 8\}$. When this parameter is *not* being varied (in the subsequent plots), the default value is 6. Another parameter that we vary in our experiments is the cutoff cost C for the path costs between the terminal vertices. The cutoff cost is set as a multiple of the shortest path cost for each pair of terminal vertices, so it is a bound

on suboptimality. If the cutoff is 2, then only paths that are less than or equal to twice the optimal cost are considered. In our experiment we vary this value from the set $\{1, 2, 3, 5\}$ where 1, means only optimal paths are considered. When this parameter is *not* being varied (in the subsequent plots), the default value is 3. Lastly, the weights W are randomly assigned to all tasks in the range $[0, 1)$ and normalized so they would sum to 1. In total, the experimental settings include every combination for the number of terminal vertices and cutoff bound. For each unique setting, we generate 10 random graphs (setting the random seed in the code from 0 to 9) and ran the algorithm with both cost functions on the same graphs. This allows us to get a range of predictability measures produced by the two approaches for the same setting.

With respect to algorithm parameters, the maximum number of candidate paths for each task ($t \in T$) used during the search process for all experiments was capped at 20; there may not exist as many paths within a specified cost cutoff, and so it is the maximum number. The search process considers combinations of these paths to find a good solution. The number of random restarts for the hill-climbing search was fixed at 5.

The algorithm was programmed in Python, and using networkx [45] for graph operations including generating random graphs, and shortest path computations. The experiments were run on a PC with Intel® Core™ i7-6700 CPU, running at 3.40GHz on Ubuntu 16.04 with 32 GB of memory.

3.6 Results and Analysis

We present the results comparing the two cost functions (GSC and BVC) side by side for each measure and the same dimension of variation. Each boxplot represents the variation for that measure in that setting. Recall that for the WPC measure,

lower is better, and for NV/NBV higher is better.

In Figure 3.5 and 3.7 we vary the suboptimality of paths allowed in the search. For those plots, the number of terminal vertices is kept constant at 6. The plots show position-based predictability measures in general improve as we allow for more suboptimal paths, i.e. the WPC measure decreases and NV/NBV increses. This is expected as there are more possible paths for the search algorithm to work with. The BVC cost function does consistently better than GSC for all settings.

In Figure 3.6 and 3.8, we vary the number of terminal vertices while keeping the maximum suboptimality of allowed paths at 3. The position-based predictability measures worsen with more terminal vertices. This is unsurprising as the search process has to satisfy more task paths in T which increases the likelihood that more branching vertices are needed since non-intersecting paths within the cutoff may not be possible. We see that the BVC cost function performs better than GSC here as well.

GSC does improve position-based predictability measures with an increase in allowed suboptimality. Reusing vertices and edges for different paths, (which GSC encourages during the search) can reduce the number of branching vertices. However, our data indicates that it pays to explicitly factor in the branching vertices into the cost; using the BVC cost function consistently gives better results in position-based predictability measures.

In Figure 3.4, we see one example of the typical case where GSC cost minimizing does worse than BVC for position-based predictability; BVC results in a much more predictable navigation-graph, similar to the images of the hospital setting in Figures 3.2 and 3.1.

One might be concerned that the reason BVC does better in the NV/NBV measure when the suboptimality allowed increases, is because longer paths are being

abused. In Figure 3.9, we show the average suboptimality of the paths in the minimized graph using GSC and BVC while varying the allowed suboptimality of the initial candidate paths. The number of terminal vertices is fixed at 6. While the average suboptimality does increase, BVC keeps it lower than 2-suboptimal, and is comparable to GSC. This is because BVC (like GSC) considers the edge cost; so longer paths are avoided as they would increase the cost considerably. So the NV/NBV measure is improved more by the fewer number of branching vertices, and not by the use of very long paths.

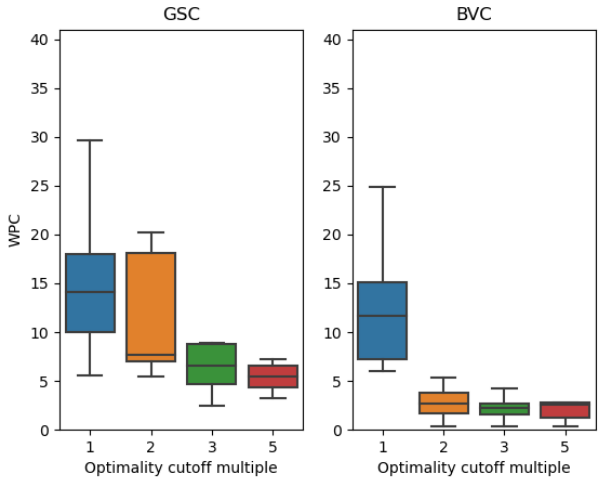

Figure 3.5: WPC Measure With Varying Optimality Cutoff

Lastly, we present the data for time-taken for different settings in Table 3.1. The factor that affects time the most is the number of terminal nodes, rather than the graph size. One might expect as much since the search is over combinations of paths between terminals. More terminal nodes would imply more paths are needed for connectivity, and the search space is commensurately larger as well.

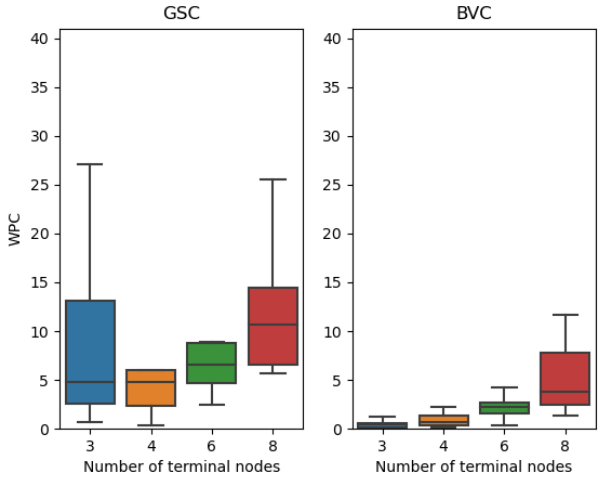

Figure 3.6: WPC Measure With Varying Terminal Vertices

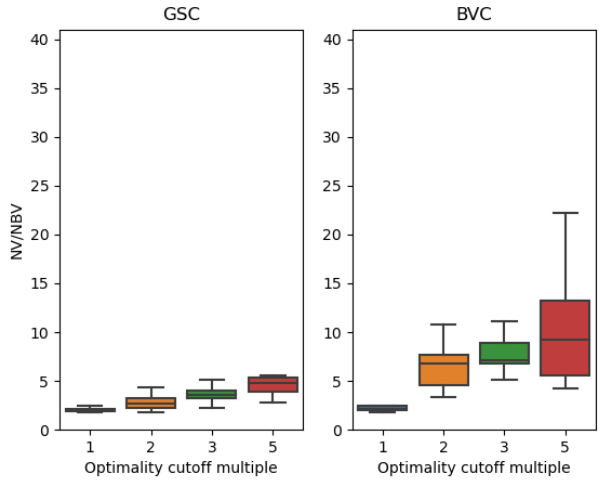

Figure 3.7: NV/NBV With Varying Optimality Cutoff

3.7 Human-subject Experiments

In our human-subject experiments, we sought to evaluate if minimizing branching vertices really does make a difference on the cognitive effort in reasoning about paths. The problems we presented to the participants (Figure 3.10) were made from a 6x6 grid with 20% of the edges removed, and 5 arbitrarily chosen terminal nodes. There

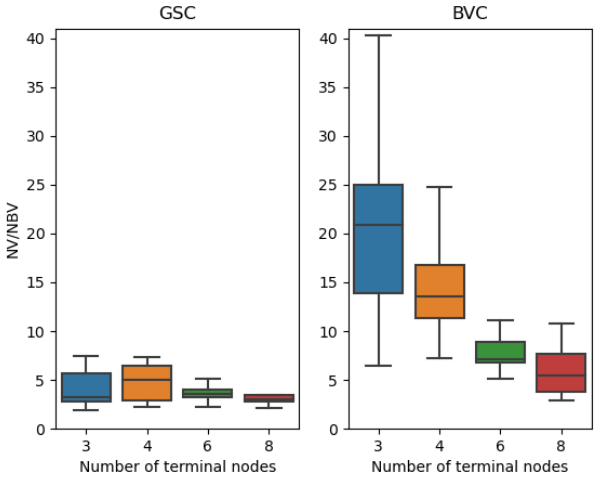

Figure 3.8: NV/NBV with varying terminal vertices

Graph Size	Terminal Nodes	Time Taken
20x20	8	334
40x40	8	798
30x30	8	551
20x20	10	1176
30x30	10	2152
40x40	10	3005

Table 3.1: Average Time Taken for Varying Graph Sizes and Number of Terminal Nodes

were two robots in each problem, and a path defined for the human to move, as illustrated in Figure 3.10. The robots moved on the blue circles as allowed by the black arrows, and the human moved on the yellow circles according to the red arrows. All of them move 1 step at the same time. We asked the participants to determine at what step the human and a robot would collide (we make it easier by telling them there would be a collision). We presented two problems: problem 1's graph was produced from the 6x6 grid by optimizing with GSC and had fewer vertices(14 in number) but more branching vertices(3); problem 2 was optimized by BVC and had more ver-

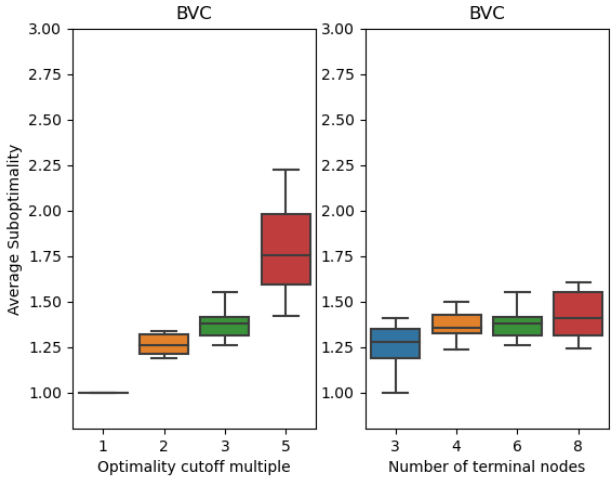

Figure 3.9: Average Suboptimality of Paths Chosen With Increasing Allowed-Suboptimality of Candidate Paths for GSC and BVC Cost Functions

tices(16) and fewer branching vertices(1). Our objective was to see if it took less time for the same user to answer correctly for the problem with fewer branching vertices. We chose such an evaluation approach as we wanted to compare the time taken when thinking/reasoning about paths when only the number of branching vertices changed. We try to control all other variables such as the number of steps in the human's path, and the number of overlapping positions with the robot's navigation graph.

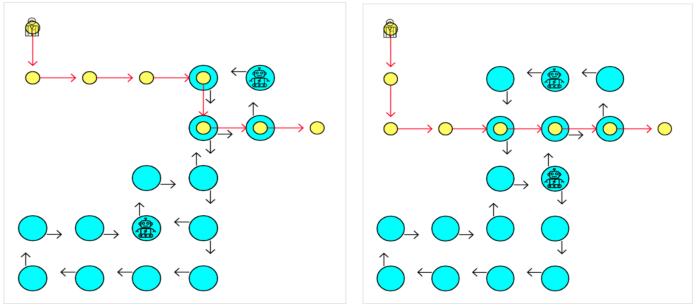

Figure 3.10: Problems Given in the Human Subject Experiments; Problem 1 (Left) Has More Branching Vertices and Problem 2 (Right) Has Fewer.

We considered that there could be a wide disparity in the cognitive ability of people to reason about paths, so we adopted the *within-subject* approach in our experimental design; we compare the performance difference with respect to the same person, and not across people. The null hypothesis is that the mean time difference is 0 (no impact of branching vertices). If our assumption on the effect of branching vertices was sound, then the mean time difference should be less than 0 (takes less time with fewer branching vertices). We fix the human path, rather than let the participant choose their own path, as we wanted to control the variables of the study as much as possible, to isolate the effect of branching vertices.

Before we presented the problems to the subject, we first show an animated example of the human and robots moving in a different grid. We showed examples of a safe path and a path with a collision. Then we presented a practice problem with two robots and a fixed human path. The subject had to get the practice problem correct before proceeding to the test problems; the practice problem was on a different grid(problem setting) than the test cases. The two test problems were presented in a randomized order. The subject can only proceed to the next problem after getting the first problem correct; this was needed so we could compare the time taken, which in turn reflected relative difficulty and cognitive effort. At the end of the experiment we debriefed participants by explaining the purpose of the experiment, and gave them the option to reach out to us for more information. Our human subject study had IRB approval.

To make the comparison fair, the human path in both problems was of the same length (7 steps), and overlapped with the robot positions the same number of times (3 positions) as can be seen in Figure 3.10. In both problems, the human collides with a robot on step 5, which the participants ofcourse did not know. So, they had to rollout the path for the robots for the same number of steps before they could

infer a collision. We matched the human path and robot navigation-graph as closely as possible; this was possible since they came from the same initial graph. We made sure that atleast one branching-vertex in both problems was involved in determining when the human collided with the robot. So the major difference in the two problems are the additional branching vertices in problem 1, which would naturally create more paths to consider. We hypothesized that this should make the user take longer to answer (more difficult).

We also worked to minimize the carryover effect in within-subject experiments; showing problem 1 first can make problem 2 easier due to familiarity. By first presenting the animated example and a practice example, we sought to sufficiently prepare the user, and give less of a learning benefit after doing either of the test problems first. We also randomized the order of problems to avoid order affecting the results.

All participants were from the Amazon Mechanical-Turk(Mturk) service [28] and filtered by "masters" qualification for users (users that had a good track record).There were 36 subjects in total. We tracked the time taken and guesses made for each problem. Of the 36 data points, 7 were removed. Of the 7, 2 were removed as it was apparent they were randomly guessing (we recorded guesses and time of the guess). 4 were removed because they were extreme outliers, as in the subjects took in excess of 201 seconds for a problem. We computed the outlier cutoff using the interquartile rule [86]. By this rule, we add $1.5 \times IQR$ (inter quartile range) to the $75th$ percentile value to get the upperbound; the lower bound similarly calculated from the $25th$ percentile value, and this was below zero and so ignored. Note that the outlier data points also supported our hypothesis as time taken for problem 2 was much less than problem 1, but we eliminated them due to them being extreme outliers which would hurt the paired t-test evaluation. The last remaining data point was removed because the user solved both problems in 2 seconds which implied they either knew

the answers or repeated the test under a new ID. Once we have the time taken for solving each problem, we take the difference between the time taken for problem 1 and 2 (for each user), and run a paired t-test [52]. We were looking for evidence at the 5% significance level (p-value 0.05). The mean difference in time taken was -47.53 seconds, i.e. problem 2 on average takes less time. The statistical significance of this result comes from the paired t-test analysis done with the Scipy library. The p-value computed is 0.0006, which means we can reject the null hypothesis of no difference in time taken with high confidence. If we only consider the data with problem 1 appearing first, the mean difference is -49.08 and the p-value is 0.0118. If we only consider the data with problem 2 appearing first, the mean difference is -46.5 and the p-value is 0.0185. So we can reject the null hypothesis, and we have strong confidence in the mean difference being lower in favor of fewer branching vertices. The average time to do problem 1 and 2 was 87 seconds and 38 seconds respectively. We certainly do not expect people to invest such amounts of time navigating around robots. The time taken was because we asked people to count the steps of the human and robot and indicate the earliest possible step at which they would collide. Counting to get the exact and earliest collision step, and having to consider combinations of path segments is why the time cost is higher. In reality, humans do not count to get the exact time or step of a path-conflict, only that we intuit there might be one and adjust our path. Our human-subjects problem was to evaluate the *relative* cognitive cost in path-reasoning when there are more branching vertices, and the data supports this. We expect this relative cognitive cost to translate to humans moving in the real world alongside robots using our approach.

3.8 Summary

In this chapter we presented the problem of computing the navigation-graph for position-based predictability. We then defined measures for position-based predictability with justifications for them. We presented and evaluated a hill-climbing algorithm to minimize graphs for position-based predictability. We also conducted human-subject studies to show that fewer branching vertices can make the problem of reasoning about robot motion from it's current position alone easier.

We posit that the navigation-graphs output by our approach can be used to determine how an AGV or any mobile-robot's grid/paths ought to be laid out; this would help make motion prediction easier for humans, especially when there are multiple robots moving in the space. Predictability would help make the robots feel safer, the space more comfortable, and thus help with the adoption of AGVs and other mobile robots in more settings.

The focus in this chapter was on the robot behaving so as to make it easier for the human to act, without controlling the human's actions or policy. In the next chapter, we will focus on how to compute policies for human agents.

Chapter 4

TRADING OFF VALUE FOR REDUCED POLICY COMPLEXITY

In this chapter, we focus on computing policies for humans that consider the errors we make and managing the complexity of policies. In the work by [32], one reason why it is important to compute a policy for human agents to execute –rather than just focusing on the automated system's policy– is Out Of The Loop (OOTL) performance. OOTL is when the human has become aware of an error with the autonomous agent, and has to take over. One must expect low situational awareness [32] as the overseer (human) might not pay attention as the autonomous agent works fine most of the time. When the human has to jump back in, the user maybe under stress to address the situation, and so the policy computed ought to consider the effects of stress and fallible execution. In a SDM problem, errors can cascade and so care must be taken on deciding appropriate policies to consider the errors we make. One of the guidelines in [32] is to reduce task complexity which advocates for reducing the number and complexity of actions. In the work we did [41], this translates to less complex policies, and we will define what complex means shortly. We compute these policies in the context of Markov Decision Processes (MDP). In this problem setting, while managing policy-complexity, we consider human perception (attention) errors and how that affects execution in a SDM problem, as well as human behavior under uncertainty. Human responses to uncertainty is something we have not seen treated

for SDM in the computer science literature, and will also be discussed in Chapter 5.

4.1 Motivation

MDPs have been used extensively in many applications([12],[53],[106]) but what if the agent that has to act in such a scenario is a human, the optimal policy maybe too complex to reasonably expect a human to execute it accurately or quickly. Our cognitive and perceptual limitations may result in mistakes, such as confusing similar states. We may also take longer to execute an optimal policy since it requires more cognitive effort to discern between similar states, which is necessary when the policy for those states are different. A sub-optimal but simpler policy that can be more faithfully and quickly executed can be preferable in some scenarios. Other than OOTL performance, there is precedent for preferring simpler policies in the medical literature; one example of this is the Apgar score[7]. It is a policy that relies on a simple scoring method to determine what action to take with newborn babies. Doctors and nurses are taught a simple scoring system on few easily measured signals to determine the health(state) of the baby and act according to this single score. A more complex policy that is conditioned on more or granular measurements and prior states could result in costly mistakes. Additionally, when the same policy has to be executed many times, a lower cognitive load would also be preferable to avoid over-taxing the human.

In this chapter, we specifically consider the problem of confusing similar states– what is called “state-aliasing" here– and how that could affect the value of a policy in Markov Decision Processes (MDP). We work with the assumption that a policy which uses the same action across similar states is easier to follow or execute (which we show in human studies), and that similar states can potentially be confused with each other (state-aliasing), especially under time pressure or other stressors. This

state aliasing can lead to errors in execution due to misidentifying states. It can also result in delays in execution when similar states have different actions in the policy and so the human might pause to resolve uncertainty. If the actions were the same across similar states, then there is no need to wait and discern the states properly. We tackle human response to uncertainty more in the next chapter.

The approach in this chapter is connected to prior work by [107], which considers state-aliasing through errors in perception (robot sensors). However, their objective is to improve the sensing policy (active perception) so as to have a better internal representation for the policy execution. Doing the same with humans would add to the difficulty of following the policy, and we cannot expect the user to consistently compute accurate posterior likelihoods given a sensing process. Instead, we take the likelihood of the human agent (mis)classifying states as an input. These classification likelihoods can be empirically determined through evaluating the human; we will discuss this more shortly.

To illustrate the problem, let us consider a simple version of a *Warehouse Worker* domain. In this domain, a worker is at the end of a conveyor belt and customer orders of different sizes arrive. The human has to decide the size of the box needed (small, medium, or large). If (for example) the difference in cost of box sizes is very small, then the simplest policy is to always use the large box. This would save on delays to decide the right action. If the policy actions are easily decided, then more orders are completed (greater throughput), and the company can profit more in the long run. More importantly, the cost of erroneous execution– trying to put a medium-sized order in a small box– is avoided. If one were to ignore policy execution errors and delays, the optimal policy computed for the original MDP could actually be suboptimal when it is executed by the human due to the delays and errors in execution. Such execution errors can especially be pronounced in high-stress situations, which tend to

cause people to miss perceptual cues (Tunnel vision/Tunneling hypothesis), and poor cognitive performance [94].

In this chapter, we formally define the problem of computing a policy for a State-Aliased MDP (SAMDP). In our definition, we describe how to model two effects of state-aliasing on human policy execution; the likelihood of inaction (delay) and the likelihood of erroneous execution (which we will connect to observation likelihoods and POMDPs). We also quantify the notion of policy-confusion likelihood. We then discuss a modified policy-iteration algorithm that searches for policies that optimize for value while considering delays and erroneous execution. Our algorithm also supports weighting the search to bias the search towards lower policy confusion likelihood; such policies can be easier for humans to follow.

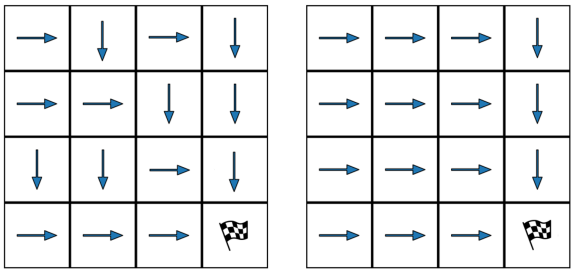

Figure 4.1: Policies With Identical Value in an MDP, but Can Have Different Values After Accounting for Errors and Uncertainty Effects

4.2 Related Work

In the scientific literature on teaching humans a policy to execute, there is a strong precedent for giving simpler policies to humans. As mentioned, the Apgar score [7] is one such example from the medical literature. "Super Sparse Linear Integer Models (SLIM)" by [99] is another example, as discussed in Chapter 2. Recall that in SLIM authors build sparse linear models with emphasis on smaller integer weights because

they make computation by humans easier and more reliable. The approaches focus on making inference easier, and in this problem we focus on aligning the policies across similar states to make policy-inference easier.

Prior work[65] on a connected problem of policy summarization and assume that humans will infer the same or similar policy-action based on similarity of states in the domain model of the human. They propose an Imitation Learning (IL) based summary extraction that uses a Gaussian Random Field model [114] for human policy extrapolation. They considered that people use the similarity between states for generalizing policy summaries to states that were not part of the summary. For one of their domains, they reported that 78% of their participants used state similarity based policy-summary reconstruction. The authors also argued that state similarity could have an effect regardless of the objective or reward for the state. This will be in accordance with our modeling choice of having the (mis)classification between states be independent of rewards and goals. Finally, their IL method led to better policy summaries, and using state similarities helped. This mirrors the approach here in that state-similarity –which in our case can lead to state aliasing or misidentification – ought to be considered for generating better policies.

Recall that one of the effects of state aliasing on policy execution is delayed execution, which we will model in our problem setting. This can bring time into the problem. Adding time and variable length action duration to MDPs can make it a Semi-Markov Decision Processes(SMDP). SMDP considers the case when time between one decision and the next is a random variable. This has some similarities to how we model delay time since the number of delay steps in a state becomes a random variable. However, there are two critical differences. First, the cause of the delay is not due to an explicit action in the policy nor dependent on the current state alone. Rather, the delay is due to state aliasing and policy differences between those similar

(aliased) states. Second, SMDPs are Markovian, whereas this chapter addresses a problem in which the actions of states are coupled and so becomes Non-Markovian.

If we ignore the policy execution delay due to confusion and just think about the state-aliasing, then this problem can be seen as a Partially-Observable Markov Decision Process (POMDP) problem where the likelihood of confusing states each other becomes the observation function of POMDPs. If we were to tackle that limited version of the problem with POMDP solvers, there is still the issue that we cannot give a POMDP policy to the human; a policy that is conditioned on belief state (likelihood of possible states). We cannot expect the human to track their posterior state likelihoods accurately. POMDP policies can also be defined by histories of observations. If we think of the state that the human inferred as an observation emitted by the state (the human only knows this "observation"), then the policy we return is akin to a POMDP policy conditioned on a history of 1 observation; these are called reactive controllers [69]. Note that this would be ignoring the delay effect from policy confusion.

4.3 Problem Definition

The problem of generating policies for a State-Aliased MDP (SAMDP) is defined by the tuple $\langle S, A, T, r, \gamma, \phi, p \rangle$. Each of the terms are defined as follows:

- S is the set of states in the domain;
- A is the set of actions including $a_\emptyset$ which is a state reidentification action
- $T : S \times A \times S \to [0, 1]$ is the transition function that outputs the likelihood of transition from one state to a successor state after an action. This includes the reidentification action transitions.

- $r : S \times A \rightarrow \mathbf{R}$ is the reward function. This includes the reward associated to reidentification actions.
- γ is the discount factor
- $\phi : S \times S \rightarrow [0, 1]$ is the likelihood of classifying the first state as the second . This is due to state similarities.
- $p : S \rightarrow [0, 1]$ is the likelihood of starting in a particular state.

The objective is to output is a policy $\pi : S \rightarrow A$ that seeks to maximize the policy value (equation 4.2) while mitigating the delays by keeping the policy confusion score low (equation - 4.3). One must account for the classification likelihoods when computing the policy since the policy that is actually executed (and the value) is affected by them. The Classification likelihood (ϕ) is the pivotal factor in this problem. If there was no classification of states (state aliasing), then it is a standard MDP, and policy iteration would solve it.

The similarity or rather the mis-classification of states due to the similarity can lead to policy execution errors. This can happen when the incorrect identification of states causes the actions of the aliased states to be executed in the current state. The probability of an action in a state for a given policy after accounting for the classification likelihoods is defined by Equation 4.1.

$$\pi_\phi(s, a, \pi) = \sum_{s' \in S} \phi(s, s') * \pi(s') \tag{4.1}$$

where $\pi(s, a)$ returns the likelihood of the action in the input state for the deterministic policy π, and π_ϕ is the probabilistic policy after consider effects of state-uncertainty. Due to this effect on policy, the problem setting becomes non-markovian as the action and transitions in one state is affected by that of another state. Note

that the likelihood of an action is further changed when we consider delay actions, which we will discuss in the following section. For now, consider $\pi_\phi(s, a, \pi)$ as the likelihood of taking an action under the effects of state uncertainty. The value after the policy after accounting for state-uncertainty is:

$$V_\phi(\pi, p) = \sum_{s \in S} p(s) * V(\pi_\phi, s) \tag{4.2}$$

where $V : \pi \times S \rightarrow \mathbf{R}$ is the value of the state for the policy π_ϕ; π_ϕ is the resultant policy after the effects of state-aliasing are applied to the input policy pi, as defined in Equation 4.1. The effect of aliasing is erroneous execution due to misidentifying a state, whose likelihood is captured by $\phi(.)$.

The state classification likelihood also determines policy confusion score, where the confusion score is formalized as:

$$CS(\pi, p) = \frac{1}{|S|} * \sum_{s_1 \in S} \sum_{s_2 \in S} \phi(s_1, s_2) * 1[\pi(s_1) \neq \pi(s_2)] \tag{4.3}$$

The confusion score will lie in the range $[0, 1]$ where a score of 0 implies all similar states have exactly the same action. A score of 1 can only happen if a state is always mistaken for another state and the policy mismatches.

The reasoning for this quantification of confusion is as follows; if any pair of states can be confused with each other, but the action in the policy is different, then it adds to the likelihood of policy confusion. How much it adds to confusion is determined by the likelihood of confusing the two states, given by ϕ, and the policy given to the human π; two very similar states with different actions adds more to the confusion score than two less similar states with different actions (less likely to confuse the two actions). So the classification likelihood is used as the weight when considering each pair of states whose actions in the policy do not match. For an example of policies

that have different policy confusion scores but comparable values, see the gridworld example in image 4.1 where the sole reward is obtained by transitioning into the bottom-right grid. Note that we do not assume that the classification-likelihood (ϕ) is symmetric; in some settings the state identification maybe biased.

4.4 Delay Effect From Policy Confusion

Recall that when state-uncertainty is a problem, we consider *two* effects on policy execution; incorrect policy execution, and delay in policy execution. The incorrect execution due to uncertainty was described in Equation 4.1.

The other effect is the action delay. After observing the real state, the human might infer that it is the (most likely) state s_1; this can be thought of as the maximum aposteriori probability (MAP) state in the human's mind after seeing the real state in the environment. However, the human might not be certain and think that it could be another state s_2 as well. If the actions are different in the policy for these two states, then one might spend additional time observing the environment again rather than act, resulting in a delay. If the actions were the same in the possible states, then the human can act without further delay. For example, in the Warehouse Worker domain, the policy is that all customer orders can be put in the large boxes. So, regardless of the workers confusion about the order size, they can act without further state-resolution and avoid delays.

We model the likelihood of delay in policy execution as increasing with the number of states that the current state could be confused with, and have different actions in the policy. We model this delay as a special action that represents the agent reidentifying the state due state confusion. We assume that the human knows the policy or has access to the policy via a chart or device, so there is no consideration for forgetting the policy. The likelihood of reidentification action –which is the cause of

delay in policy execution– is determined by the policy and state confusion likelihoods. We define this in Equation 4.4.

$$p(a_\emptyset, s) = \sum_{i \in \{1...|S|\}} \phi(s_i, s) * \sum_{j \in \{i+1...|S|\}} \phi(s_j, s) * 1[\pi(s_i) \neq \pi(s_j)] \tag{4.4}$$

This is the likelihood of one reidentification action. The likelihood of two reidentification actions is the product of two probabilities, and so forth with the likelihood of many successive reidentification actions becoming geometrically smaller.

Given this delay action ($a_\emptyset$), the probability of a policy action, previously defined in Equation 4.1 becomes:

$$\pi_\phi(s, a, \pi) = (1 - p(a_\emptyset, s)) \sum_{s' \in S} \phi(s, s') * \pi(s') \tag{4.5}$$

With Equations 4.4 and 4.5, one has the complete definition of the probabilistic policy ($\pi_\phi(.)$) that we model the human as executing for a given deterministic policy. With this, we can compute better policies that minimize the effect of uncertainty and tradeoff with policy-complexity.

4.5 Policy Computation Algorithm for SAMDP

Finding an optimal solution to the SAMDP problem is challenging since the problem is both non-markovian and requires optimizing two sometimes opposing objectives; namely the policy value and confusion score. To handle this, we adopt a modified policy iteration approach that we call Global Value Policy Iteration (GVPI). The confusion score of a policy is factored into the value computation by the reidentification actions which occur more often in complex policies. When selecting actions during policy iteration, we need to consider not just the local value effect, but also the

effect on the value of other states. This is because the possible state misidentification couples the policy of different states; this means the policy in one state affects the policy in another, and so affects the value of other states (possibly negatively). This can lead to update loops and never converge. So at each step of policy iteration, we consider the average value of over all states as the measure by which we update the policy.

When we evaluate a policy change in GVPI, we first compute the likelihood of reidentification actions for each state after the policy change. Then we compute the state transition likelihoods for that policy (including the reidentification action transitions), and compute the corresponding Markov Reward Process (MRP)[54]. Using this MRP we compute the value for all state using a closed form computation (will discuss shortly). We compute a score over the values for all states and use that to choose the action in policy iteration. Since we consider all state values, we dubbed this G̈lobal-Value Policy Iteration.̈ We did try a policy gradient approach as well (search over the space of soft policies), and found GVPI to perform better for our experiments. We now explain our approach in detail.

4.5.1 Computing Reidentification Action Likelihoods

In each policy iteration step of GVPI, we start with a deterministic policy. First, we account for the delay due to policy confusion. We compute this reidentification action likelihood as in Equation 4.4. The remaining probability $1 - p(a_\emptyset)$ is the likelihood of the human acting. Then we account for erroneous execution by considering the probability that the incorrect state was inferred. So the likelihood of an action being executed is defined by Equation 4.1. This gives us the updated policy π_ϕ that accounts for delays and erroneous execution.

4.5.2 Translating to the Equivalent MRP

After computing the updated policy, we compute the equivalent Markov Reward Process (MRP) associated to that policy; this is done by computing the transition likelihoods between ordered pairs of states based on the policy. The reward for each state is the reward for each action taken from that state, multiplied by the likelihood of that action being taken. This includes the reidentification action and the associated reward (cost) of that action in that state; it may not always be that reidentification action results in staying in the same state.

The reason we transform it to an MRP is that it allows us to exactly compute the values of all states in one closed form computation (Equation 4.6).

$$\vec{v_s} = (I - \gamma * P_{ss'})^{-1} * \vec{r_s} \tag{4.6}$$

Where I is the identity matrix, and $P_{ss'}$ is the probability of transition between two states, which is defined by the policy.

4.5.3 GVPI Search Process

After computing the value of the states, we do not just sum or take the average of the state values, rather we consider the average of the inverse of state values as in Equation 4.7.

$$ps(\pi, p) = \sum_{s \in S} p(s) * \frac{1}{V_{\pi_\phi}(s) + 1} \tag{4.7}$$

where π_ϕ is the policy derived from the input policy π after applying the effects of state aliasing as described in Equation 4.1. $V(\pi_\phi, s)$ is the value of this policy after applying the reidentification action likelihoods in Equation 4.4.

With this score, GVPI iteratively proceeds to minimize this score, it does not

maximize like in policy iteration since we have taken the inverse of the value. We chose to score actions this way because we observed that optimizing for the sum of state values can cause the policy iteration search to ignore the value of some states in favor of high value states since the overall sum is greater. In some problems, this maybe acceptable. But for some others, such as in gridworld, it produces more policies where the goal state is not always reachable from all states, which is undesirable. This score helps improve the state values of all states more uniformly; increasing the value of a state with value 0 contributes more to reducing the score (which is to be minimized) than improving a state with a higher value.

As one might intuit, this policy iteration search is not guaranteed to find the optimal policy for the expected value. Rather it is a means to generate a set of good policies that optimize for expected value and account for delays in execution. Repeated random restarts help find better policies, and in each attempt it will stop when the policy can no longer be changed to improve the score. This is as opposed to using standard policy iteration which could end up in infinite loops for SAMDPs. Since SAMDPs are non-markovian and the state policies are coupled, the policy update in one state could reduce the value of another state by changing it's policy. The policy iteration procedure could get stuck updating back and forth between coupled states, which we saw when we tried to solve SAMDPs with policy iteration.

4.5.4 Trading Value for Reduced Complexity

Thus far we have not discussed how to explicitly penalize the search process for policy confusion. By incorporating a reidentification action into the execution when policies are confusing (similar states, different actions), the policy search automatically goes towards simpler policies unless the reward is sufficiently higher to merit additional risk of confusion. If one is interested in searching for even simpler policies,

our methodology also supports pushing the search process to look for policies of lower confusion score, which will often come at the expense of value. The policy score is updated to allow this as in Equation 4.8

$$ps2(\pi, p) = (1 - \omega) * ps1(\pi, p) + \omega * CS(\pi, p) \tag{4.8}$$

where CS is the confusion score from Equation 4.3 and ω is a hyperparameter between $[0, 1]$ which determines the emphasis given to reducing confusion, where 1 means the search will focus purely on reducing confusion. Additional scaling of the two scores was not helpful because the ps1 score is already in the range $[0, 1]$, and so too is the confusion score.

For an example of the kind of simple policies found by GVPI for a gridworld setting, see Figure 4.8. The policy on the right is the simpler policy output by GVPI. The policy on the left is the optimal policy without considering state-aliasing, that is output by standard policy iteration for the original MDP. The left policy has a lower expected value in SAMDP, and a higher policy confusion score.

4.5.5 Obtaining the Classification Likelihood Matrix

The likelihood of two states getting misidentified would be affected by the domain features being used, and a person's perceptual capabilities. In this paper we take the state classification probabilities as input.

In an actual application, the probability of state classification (also misclassification) may need to be empirically determined. For example, in the warehouse worker domain an employee could be tested to see how they classify packages by a supervisor. This can be used to compute the classification likelihoods. When determining these likelihoods, the human should be asked to classify within some desired time cutoff or as quickly as possible. The classification likelihoods would be affected by the time

allowed to identify a state for that domain. Then based on classification likelihoods, a policy maybe tailored specifically to that person. This data may also be averaged over groups of people, and a standard policy could be developed.

Alternatively, the state classification likelihood could be defined by normalized state similarities. This is reasonable if a similarity function that reflects human perception is available for the domain. We take such an approach for our human studies which will be discussed.

4.6 Experiments and Results

We tested our algorithm on two domains; Warehouse Worker and Gridworld. The discount factor was set to $\gamma = 0.9$. We varied the weight for reducing confusion (ω) between $[0, 1]$ in increments of 0.1. For each setting, we ran the GVPI search 10 times.

4.6.1 Warehouse Worker Domain Setup

In the Warehouse Worker domain, a worker stands at the end of a conveyor belt on which customer orders are sent. The customer orders comprises of a group of products. Each order needs to be put into a small, medium, or large box. Additionally, the worker has to decide if bubble wrap is necessary for the products in the order.

The states in this domain is what kind of an order a set of products actually is, and there is an associated correct action for each order type. The state and action sets are defined by the cartesian product of the set of box sizes $\{small, medium, large\}$, and if bubble wrap is needed $\{wrap, no\_wrap\}$. For example a set of glass items could be a small order that requires a small box with bubblewrap, $small \times wrap$. When a worker sees an order, it is not always apparent what box size is needed. For some orders, they may mistake a small order for a medium sized one or vice versa. Additionally, due to the diversity of products, the worker has no idea which products

actually need bubble wrap or not. For example there maybe tempered (hardened) glass products that do not need bubble wrap, but the worker might not know this.

In our conceptualization of this domain, after the worker goes through basic training in the warehouse, the worker is evaluated by the supervisor to evaluate how they classify orders. This becomes the classification likelihoods for ϕ. Based on this, a policy is developed for the worker by considering the company's average estimates for the money made per order when using different types of packaging (reward specification), and the likelihood of order types.

We now detail the domain configuration settings we used in our experiments for the warehouse domain. The classification likelihoods are shown in Table 4.1. The likelihood of a worker confusing any small order with any medium sized order or vice versa is 16.34%, and is the same for misclassifying any medium with any large order. The likelihood of confusing a small order with a large order is less than 1%. The likelihood of the worker correctly determining if bubblewrap is needed is 50% across all order sizes; there are so many products that their accuracy for determining if bubble wrap is needed is random.

	l	l × w	m	m × w	s	s × w
l	32.68%	32.68%	12.50%	12.50%	0.98%	0.98%
l × w	32.68%	32.68%	12.50%	12.50%	0.98%	0.98%
m	16.34%	16.34%	25.00%	25.00%	16.34%	16.34%
m × w	16.34%	16.34%	25.00%	25.00%	16.34%	16.34%
s	0.98%	0.98%	12.50%	12.50%	32.68%	32.68%
s × w	0.98%	0.98%	12.50%	12.50%	32.68%	32.68%

Table 4.1: Classification Likelihood Matrix (ϕ) for Warehouse Worker Domain, Where (S,m,l) Stands for (Small,medium,large) and "w" Means Bubblewrap Needed. Columns Sum to 100%.

The transition likelihoods are defined in Table 4.2, where the rows are states, and

columns are actions. Each entry is a tuple of successor state and probability. The rewards for each state action pair are defined in Table 4.3

	l	l × w	m	m × w	s	s × w
l	S,16.66%	S,16.66%	{l},100%	{l},100%	{l},100%	{l},100%
l × w	S,16.66%	S,16.66%	{l × w},50%	{l × w},50%	{l × w},100%	{l× w},100%
m	S,16.66%	S,16.66%	S,16.66%	S,16.66%	{m},100%	{m},100%
m × w	S,16.66%	S,16.66%	S,16.66%	S,16.66%	{m × w},100%	{m × w},100%
s	S,16.66%	S,16.66%	S,16.66%	S,16.66%	S,16.66%	S,16.66%
s × w	S,16.66%	S,16.66%	S,16.66%	S,16.66%	S,16.66%	S,16.66%

Table 4.2: Transition Likelihood Matrix; Rows Are States and Columns Are Actions. Recall Actions Maps 1:1 to States and So Have Matching Names. Each Entry is a Tuple of a Set of States From State Space S, and the Probability Of Transition to One of the States in That Set. Capitalized S is the Entire Set of States

	l	l × w	m	m × w	s	s × w
l	1.0	1.0	0	0	0	0
l × w	0.9	1.0	0	0	0	0
m	0.9	0.9	1.0	1.0	0	0
m × w	0.8	0.9	0.9	1.0	0	0
s	0.8	0.8	0.9	0.9	1.0	1.0
s × w	0.7	0.8	0.8	0.9	0.9	1.0

Table 4.3: Reward for State,Action Pairs; Rows Are States and Columns Are Actions.

With regards to the domain dynamics, if the worker tries to put a medium sized order in a small box, the action will fail, and they will stay in the same state. Using any box size smaller than necessary will fail, but any larger size box will work. A successful action will transition to the next order (state) based on the probability of different types of orders. For our experiments we used a uniform distribution of customer orders (states). Lastly, the reward for using the exact action for an order is 1. The reduction in reward if a larger box is used is -0.1, and a further reduction

of -0.1 is incurred if bubblewrap was needed but wasn't used. If bubble is used but not needed, there is no reduction in reward; we consider that the cost of bubblewrap in comparison to the monetary reward for a completing an order is negligible.

4.6.2 Gridworld Experimental Setup

For our experiments in Gridworld, we used a 10x10 grid (100 states). The actions include moving up, down, left and right. The transitions are deterministic. The likelihood of confusing a grid position with another grid state is determined by the L1 distance as defined in Equation 4.9. This results in neighboring grid states being much more likely to get confused with the current state than those further away. Taking an invalid action, such as moving up from the top row of the grid results in no motion. The agent would get a reward of 100 for transitioning into the goal state in the bottom-right corner.

$$\phi_{grid}(s, s') = \frac{L1(s, s')^2}{\sum_{s'' \in S} L1(s, s'')^2} \tag{4.9}$$

4.6.3 Results

The Expected Value of policies after accounting for delays and erroneous execution are shown in Figures 4.4 and 4.2. The box plot shows the upper and lower quartiles, and the circles represent outliers. The trendline connects the median values from each setting of ω. The first setting with a "*" represents the optimal policies discovered policy iteration in the original MDP. We do 30 random restarts to get a set of optimal policies for the original MDP. The expected value of these "MDP-optimal" policies are not optimal after accounting for delays and errors in execution in the SAMDP. The policies found by GVPI are often better, especially for lower settings of ω for both domains. In both domains as ω increases the expected value goes down as

expected. The expected value for gridworld may seem low but that is expected since it is averaged over 100 states with only 1 state having all the reward. Add to this the reward discounting, delays and erroneous execution, the values computed are smaller than one might expect. We verified this by hardcoding the known optimal policy which accounts for delays and execution errors, and it's expected value averaged over all states is 0.34.

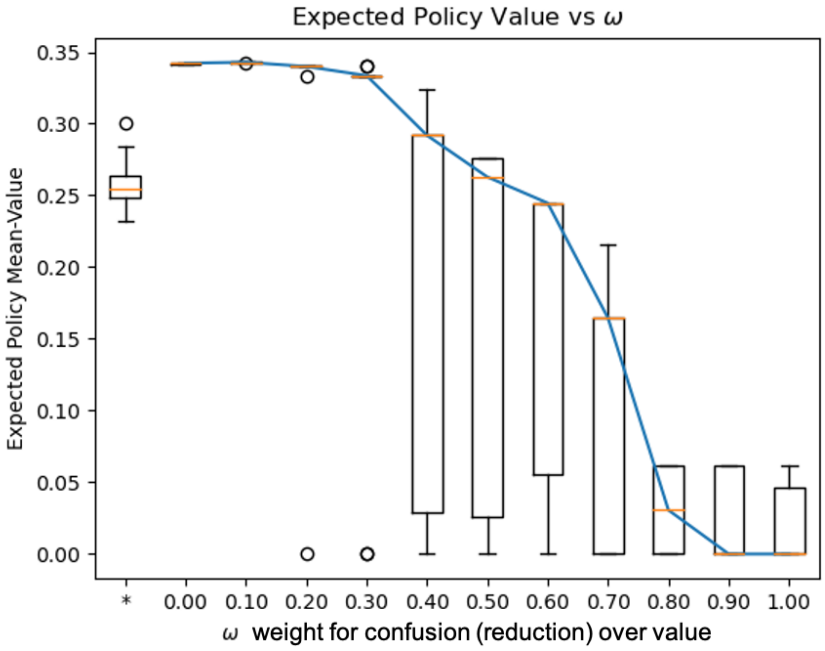

Figure 4.2: Expected Value of Policies Generated With Varying ω in Gridworld

The confusion score of policies generated by varying ω are shown in Figures 4.3 and 4.5 for gridworld and warehouse domain respectively. To interpret these correctly, please keep in mind that the maximum confusion score a policy can have is 1, by our definition in Equation 4.3. In the confusion graphs of both domains, one will notice that the policy confusion is already quite low even with $\omega = 0$. This is because the effects of policy confusion is already folded into the computation of value in GVPI through the delay effect and erroneous execution. A simpler policy would have less

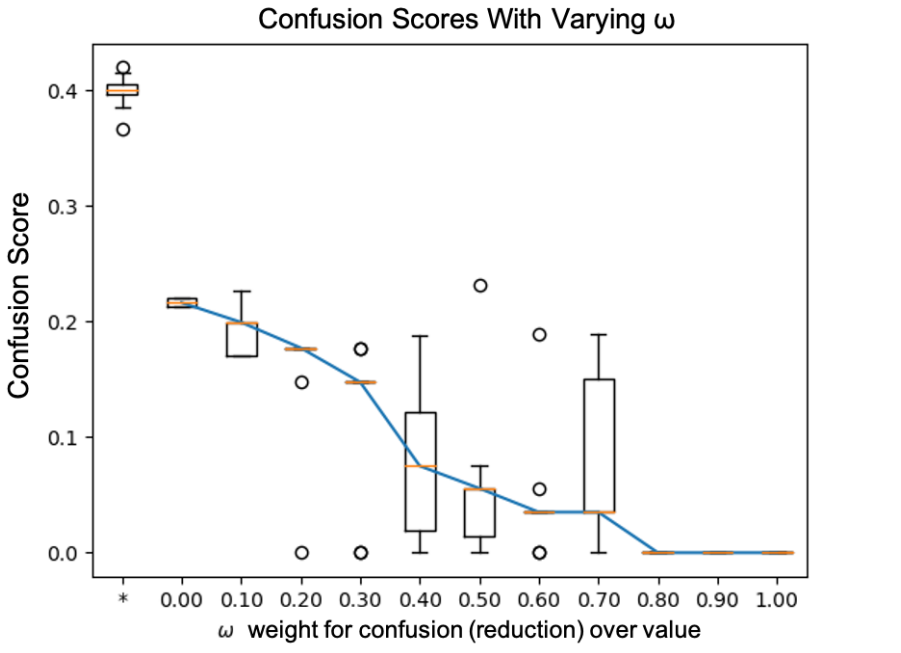

Figure 4.3: Confusion Score of Policies Generated With Varying ω in Gridworld

of both, and so it is naturally preferred during the search. Increasing ω only serves to push the search even more towards simpler policies.

For Gridworld, the median confusion value of the policies decreases more gracefully with increasing ω. We think this is likely due to the spread and availability of policies with different tradeoffs between value and confusion. This is not so in the Warehouse domain. Increasing the weight for confusion only results in it getting stuck at worse local optima in the search. We think this is due to the sparsity of policies. Also, note that the absolute value of the policy confusion is still low; the worst it does is 0.175 and the maximum confusion a policy can achieve in this domain is 1.0. In comparison, the optimal-in-MDP policies have a much higher(worse) confusion score, and their corresponding range of expected values in the SAMDP is low (Figure 4.4.

In both domains, we see high variance for expected value and confusion, especially in the middle range of ω values, which corresponds to the tradeoff difficulty during

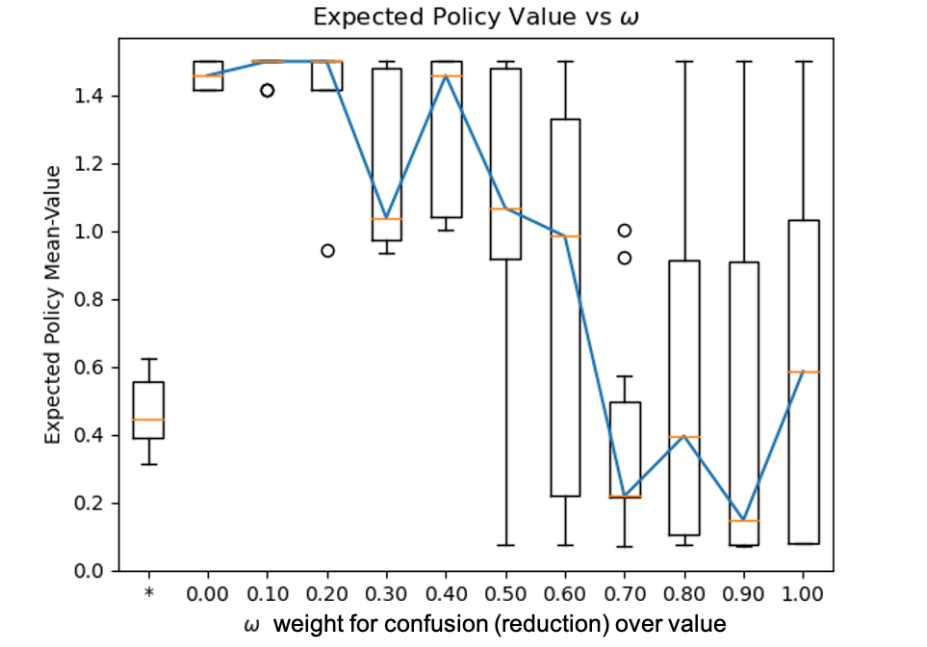

Figure 4.4: Expected Value of Policies Generated With Varying ω in Warehouse Domain

the search. Lastly, we show the expected value and confusion of policies in one plot for each of the domains in Figures 4.6 and 4.7 respectively. The blue dots represent policies discovered by GVPI, and the orange dots are the optimal-MDP policies. This is also to show that GVPI explores the space of tradeoffs between expected value and confusion.

4.6.4 Human Studies

We conducted a human study to test the hypothesis that the execution performance of humans using a simple policy is higher in contrast to when a difficult (higher likelihood of confusion) policy is given. We gamified the study by asking each participant to execute a policy as given in Figure 4.8 by matching a displayed color to the appropriate arrow direction and maximize their score. The same of participants

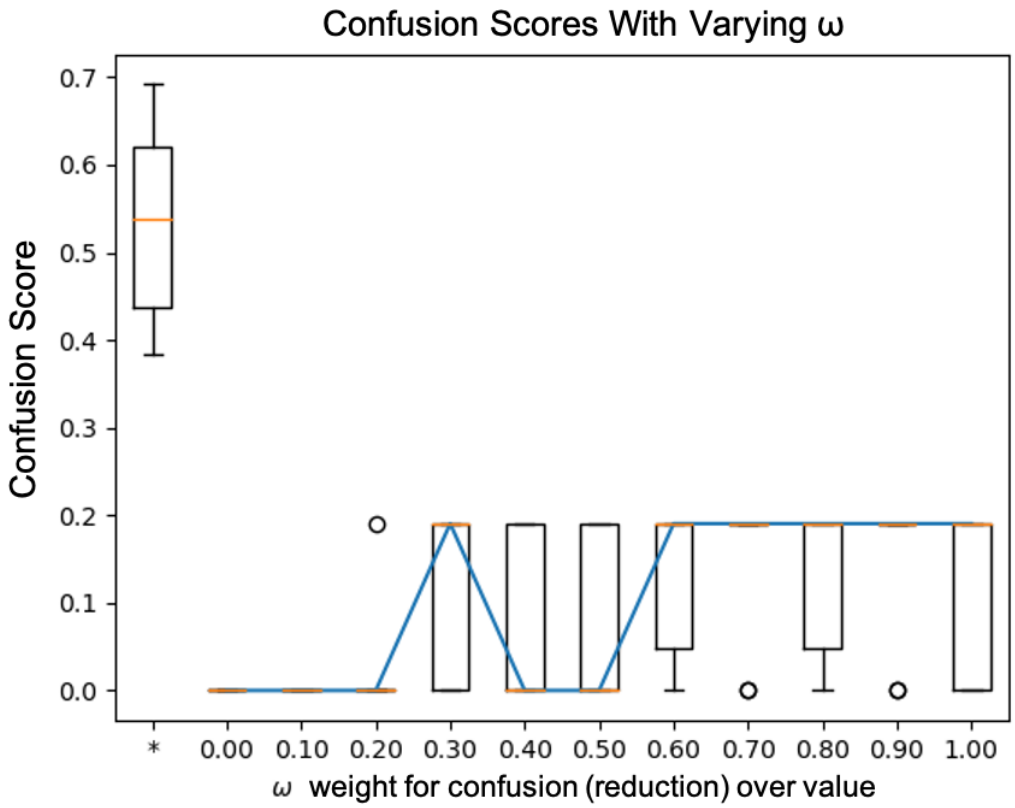

Figure 4.5: Confusion Score of Policies Generated With Varying Ω In Warehouse Domain

repeated the game twice; for a difficult policy and for a simple policy. The correct policy was always displayed on the side, so they did not have to memorize it, only follow it. Our objective was only to measure the number of actions, number of correct actions and error rate. These measures reflect how well the human can follow a given policy.

Note that some color states are intentionally visually similar to other color states to cause state-aliasing. The participants were filtered by their ability to distinguish between different colors so that they could execute both difficult and simple policies. The table 4.4 shows results for the 41 participants in this study.

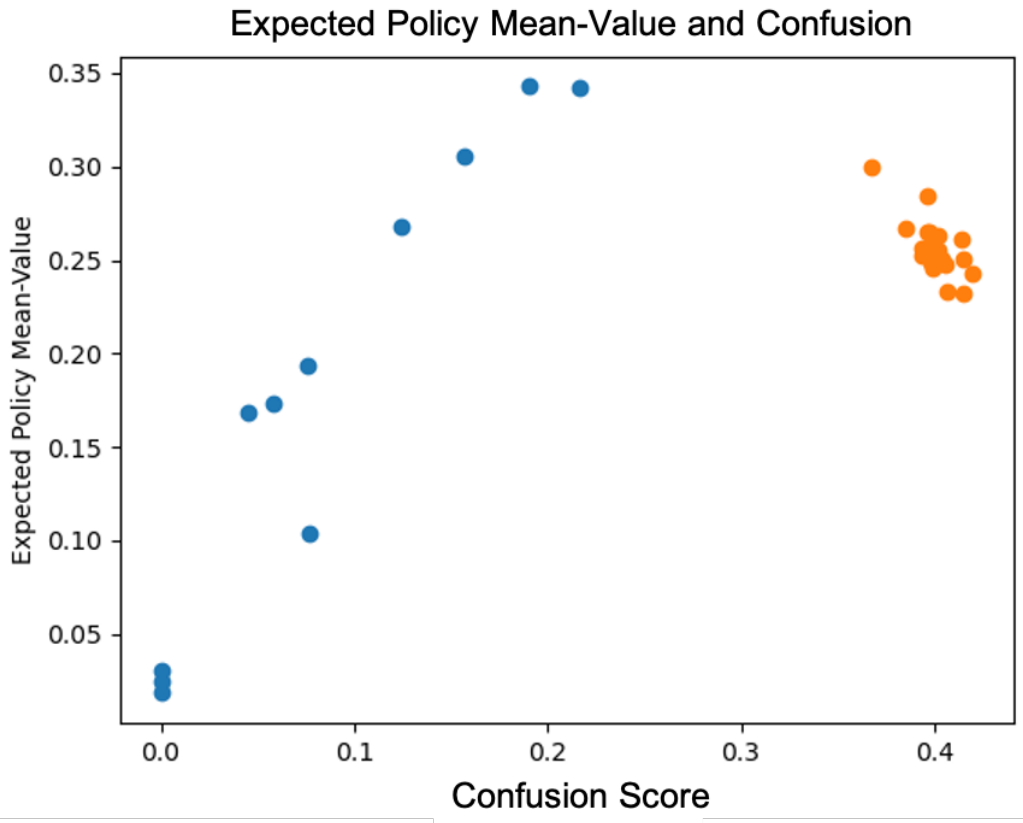

Figure 4.6: Expected Value (Blue) and Confusion Score (Orange) of Policies Generated During Search in Gridworld

We wanted to see if the number of actions executed was greater with the simpler policy; since less confusion likelihood should imply fewer delays. We are especially the number of correct actions (throughput). We also wanted to check if the rate of errors was lower. Since data from the two settings of simple and difficult policies may have unequal variances, we used a Welch's t-test (one-tail) to evaluate the results. We used the implementation in the Scipy python library [104]

For the total number of actions executed by a participant, we can reject the null hypothesis that the number of actions executed is the same or lower with the simpler policy than the difficult policy; the one-tailed T-test gave a p-value of $< .00001$.

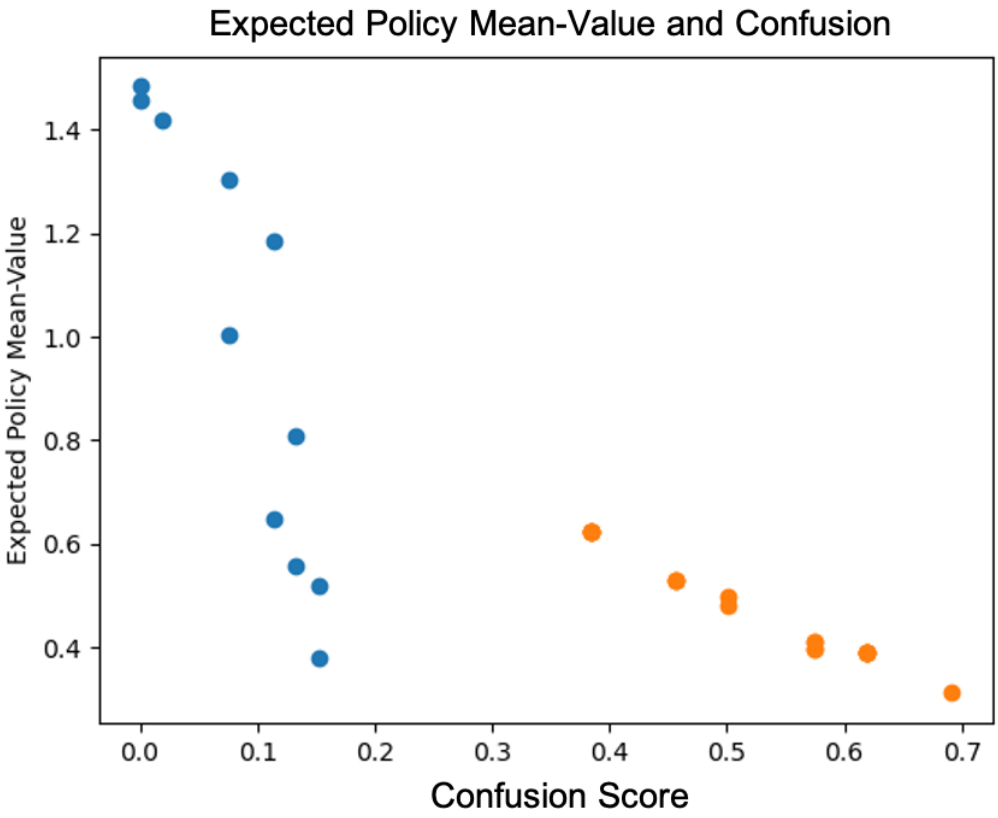

Figure 4.7: Expected Value (Blue) and Confusion Scores (Orange) of Policies Generated During Search in Warehouse Domain

For the second hypothesis, that a simpler policy yields a higher number of correctly executed actions, a one-tailed T-test gave a p-value of $< .00001$. So with a very low likelihood of error, we can say our hypothesis held good in this human study. The results are clearly significant at $p < 0.05$.

We are also interested in the *rate* of errors when executing the simple versus difficult policies. We wanted to show that the rate of errors in a difficult policy is more than that of a simple policy. A one-tail T-test for the rate of errors (number of errors/total attempts) was significant only at a p-value of 0.065. So while we have good reason to believe this to be the case, we cannot say with confidence (p-value $<$

0.05) that the rate of errors is definitely lower. There might have been other factors affecting the rate of errors that we had not considered, such as the speed of execution of the simpler policy. One possible effect is that when the participants were acting very fast with the simpler policy, the likelihood of errors went up.

We also note that our GVPI algorithm consistently outputs the simpler policy shown in Figure 4.8 for a simple MDP corresponding to the human studies. In this MDP, the rewards are 1.0 for the correct action(s) in the simple policy, 1.1 for the actions that are different in the difficult policy (see Figure 4.8), and 0 for all other actions. Transitions to successor states are independent of the action and equally likely; we needed this to test policy execution uniformly across all states. The discount factor γ was 0.9, and we modeled the state classification likelihoods such that similar color states were equally likely (50%) to be confused as each other. The simpler policy output by GVPI is desirable since the additional reward for choosing the optimal action is small compared to the loss that could be incurred due to delays and incorrect execution.

Overall, our human studies show that giving a simpler policy reduces the delays (higher number of actions executed) and increases the throughput (number of correct actions). This translates to more rewards accrued. The rate of incorrect actions was not conclusively shown to be lower, even if the data gives us good reason to think so. There could be more factors that affected the execution, such as the rate of policy execution. Running each trial for longer might give us more conclusive data.

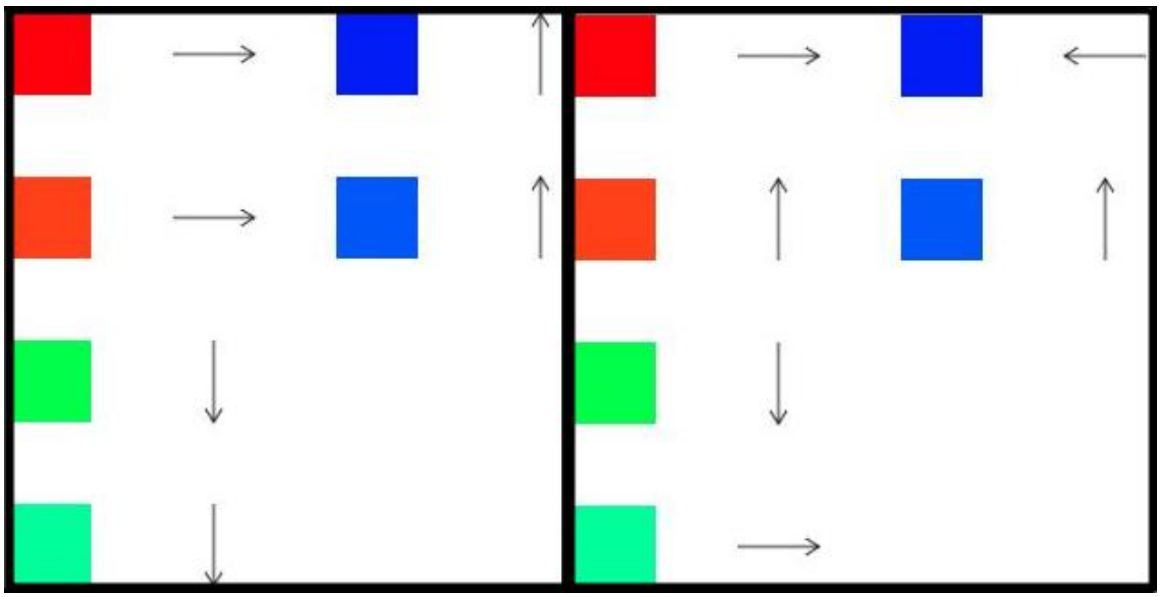

Figure 4.8: The Simple Policy (Left), and Difficult Policy (Right) Given to Users to Execute

	Simple Policy (μ, σ)	Difficult Policy (μ, σ)
Correct Attempts	26.34, 5.62	17.95, 3.77
Total Attempts	28.12, 6.13	19.68, 3.82

Table 4.4: Mean and Standard Deviation for the Number of Correct Actions and Total Actions Executed for the Simple and Difficult Policies

4.7 Summary

In this chapter, we describe the problems that can arise from state aliasing when humans execute a policy; these are execution errors and delays in policy execution. We formally define the problem of computing policies in SAMDPs and define how delays and errors can be computed for a given policy in the SAMDP using the state classification likelihood. We discuss how the state classification likelihood can be empirically evaluated for human agents and also how domain rewards can be ap-

propriately discounted to account for state detection time. Given the description of SAMDP, we present a modified policy iteration algorithm (GVPI) which searches for policies that account for delays and errors, and optimize the expected value. GVPI also allows searching for simpler policies by increasing a hyperparameter that penalizes policy confusion score. Lastly, we conducted human studies to show how our assumptions translate to real-world behavior.

Complexity tradeoff is one possible consideration when computing policies for humans. Another consideration is how human's behave in the face of uncertainty that can arise when state inference is complex or difficult; we will tackle this in the following chapter.

Chapter 5

ACCOUNTING FOR BEHAVIORAL RESPONSE TO UNCERTAINTY IN MDP

In the previous chapter we considered that the human may just delay their action due to uncertainty, and considered this delay while deciding how to tradeoff policy-complexity and policy-value. In this chapter we will focus more on the human response to uncertainty during inference. We extend the human response to uncertainty from being just a delay or inaction-step, to the human taking extra-sensing actions and updating their belief. An everyday example of this would be a person looking around to get a better sense of their location if they are uncertain where they are when following directions (a policy). We apply this response to uncertainty during state identification when following a policy in MDPs. This is the work done in our paper [42].

5.1 Motivation

When human agents have uncertainty in what to do, we may take additional sensing actions to try and resolve the state uncertainty(assuming policy knowledge is not the probelm) or repeat unhelpful previous perceptual actions. We treat these actions as coming from an extra-sensing policy of the human, and not part of the policy given to execute; they are a consequence of the human's uncertainty. We cannot programmatically control these, as we would with a non-human agent. In this work, we model the extra-sensing behavior as a state-specific extra-sensing action.

As a result of the extra-sensing action, there can be cases when the agent's uncertainty might get reduced, as well as cases where the additional sensing does not make a difference (in expectation) in identification errors; a human agent might do them

anyway as they are the only options to resolve uncertainty. In the approach here, we are *not* trying to remove uncertainty during execution; we expect it to persist in certain problems despite efforts to minimize it. Efforts to minimize it are important, but orthogonal to our work (we discuss these in the related work). We instead try to account for the effects of uncertainty when computing a policy to get better policies. An optimal policy (in terms of domain dynamics) which ignores the effects of uncertainty on execution behavior can be markedly suboptimal.

Just as in the previous chapter (Chapter 4), we consider that if the action is the same across the possible states in the human's mind, then the human will act without additional actions or delays. Since we are trying to infer the information in a person's mind, problems with mental modeling come into this problem. However, we do not need a complex mental model for this problem. What we need are just the likelihoods of certain events. Specifically, events like the likelihood of inferring state 2, when the person is in state 1. We also need a model of how they respond to being uncertain. We need a simple state-specific probabilistic model of the human agent's inference which can be built empirically and problem-specific; we do not need or want to make assumptions on the type of inference (optimal, noisy-rational). Given this, we can compute better policies for human execution by accounting for how people make mistakes and respond to uncertainty.

In this chapter, we formally define the problem of computing a policy that accounts for human uncertainty during execution in an MDP, and how that makes it a POMDP. Having framed the problem, we will discuss two methodologies to solve it, and present experimental results on two domains; a gridworld domain and a warehouse worker domain. This will be followed by a discussion on the related work, and avenues for extending this research direction.

5.2 Problem Definition and Human Model Used

In this work, we assume that the underlying MDP problem is fully observable (the ground truth state is knowable). However, when the human enacts a policy, the human's limitations leads to suboptimal execution due to state uncertainty during execution. So we need a policy that accounts for the human's state uncertainty and associated execution behavior. We will build up to the formal definition of the problem by first discussing the human model used. Then we will define how it is incorporated it into a POMDP for computing a better policy for humans.

5.2.1 Human Model

The human model is defined using the probability of inference events and extra-sensing events given a ground-truth state. For a set of domain states S, we define the human model as $H = \langle p_c, p_u, \psi_0, \psi_1 \rangle$, and the terms are defined as follows:

- $p_c : S \times S \rightarrow [0, 1]$ gives the likelihood of classifying(identifying) one state as another; $p_c(\hat{s}|s^*)$ where s^* is the true state, and $\hat{s}$ is the best-guess state that the human agent thinks it is. We will use the $\hat{s}$ symbol above a state to indicate the human's guess of the state.

- $p_u : S \times \{S_i \in 2^s\} \rightarrow [0, 1]$ gives the likelihood of being uncertain between a set of states (S_i) for a given true state. For example, $p_u(\{s_i, s_j\}|s^*)$ is the probability of the human considering $\{s_i, s_j\}$ as the possible states when the true state is s^*. We will refer to such S_i sets as a "possible-set", and reflects the mental-state or belief state of the agent; it represents what states they think are possible.

- $\psi_0 : S \rightarrow [0, 1]$ is a bias term that affects the probability of the extra-sensing

action being taken. So even if the human infers only one state, they may not feel confident and take additional sensing actions anyway; this is what the bias term (ψ_0) captures.

- $\psi_1 : S \rightarrow [0, 1]$ is a scaling-factor that is the additional probability of extra-sensing action being taken when the agent is uncertain about the right policy action. If there is more pressure to act, or the human-agent tends to be impulsive, this number would be lower to make the probability of extra-sensing action lower.

5.2.2 POMDP With Human Execution Under Uncertainty

Given this human model, the problem of a POMDP with Human Execution under Uncertainty (POMDP-HUE) is defined by the tuple $\langle S, A, T, r, \gamma, p_i, H, S_2 \rangle$. Each of the terms are defined as follows:

- S is the set of states in the problem.
- A is the set of actions in the domain, and an additional a^+ which is the extra-sensing action.
- S_2 contains one successor state to every state in S, and is reached only when a^+ is taken. This captures the change in human's mental state (belief state) after the extra-sensing action. This is illustrated for a single state in Figure 5.1. This is effectively folding the belief state into the state space.
- $T : S \bigcup S_2 \times A \times S \bigcup S_2 \rightarrow [0, 1]$ is the transition function that outputs the likelihood of transition from one state to a successor state after an action. This includes the extra-sensing action (a^+) dynamics.

- $r : S \bigcup S_2 \times A \rightarrow \mathbb{R}$ is the reward function. This includes the cost/reward associated to extra-sensing actions.
- γ is the discount factor
- $p_i : S \rightarrow [0, 1]$ is the probability of a state being the initial state
- H refers to the human model as defined by $\langle p_c, p_u, \psi_1, \psi_0 \rangle$. These terms are unique to each state, even between the state in S and it's corresponding state in S_2, as the uncertainty can change after extra-sensing action. For details on how to compute the human model terms, please see [42].

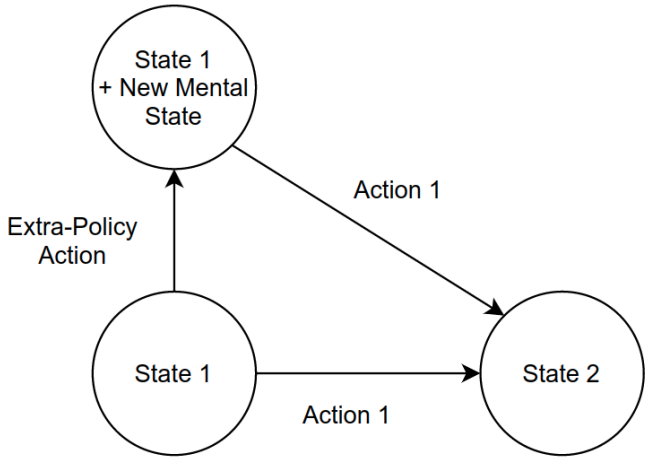

Figure 5.1: Additional State Added to MDP for State 1 to Account For Different Inference Likelihoods by the Human Agent

The objective in the POMDP-HUE problem is to output a deterministic policy ($\pi_d : S \rightarrow A$) that optimizes for policy value (equation 5.3) after accounting for effects of uncertainty –that we will shortly quantify– determined by the human model H. The deterministic policy is a mapping from a problem state to an action. The state as inferred by the human can be seen as a noisy observation emitted by the real state.

This makes it a POMDP, and why we refer to the objective as computing a reactive controller for a POMDP. Reactive controllers were defined in influential prior works [69] [73]. Before we define the objective function, we state the reasoning that was used to define it.

In this problem we make the assumption that the human's inference and uncertainty is predominantly influenced by the current state only either because of the domain, or because the human agent was instructed to do so. This makes sense especially in settings when the policy is set as an OOTL (out of the loop) default policy for when automation fails as we discussed at the start of Chapter 4.

Furthermore, we consider that if a person is uncertain over a set of possible states in their mind, and if the policy conflicts between these states, then the human is more likely to take extra-sensing actions to try to resolve uncertainty (which we show in our human subject studies). Put another way, the uncertainty that matters when executing a policy, is about what the right action is; it is not to perfectly detect the state. If the action is the same across states then the human is not likely to care about resolving state uncertainty. Building on this, we define the likelihood of the extra-sensing action (which is denoted by a^{+}) being taken as a function over the policy.

Since extra-sensing actions and state uncertainty affect execution of a policy, any deterministic policy given to the human (π_d), when actually executed by the human can become a stochastic policy; $\pi_p : (S \times A|\pi_d) \rightarrow [0, 1]$. The first effect from uncertainty, is the likelihood of taking an extra-sensing action in a state ($s* \in S \bigcup S_2$) is defined as follows:

$$\pi_p(s*, a^+|\pi_d) = \psi_0(s*) + \psi_1(s*)\times$$
$$\sum_{S_i \in 2^S} p_u(S_i|s*) \times 1[0 < \sum_{s_1, s_2 \in S_i} 1[\pi_d(s_1) \neq \pi_d(s_2)]] \tag{5.1}$$

where 1[.] is the indicator function. In short, the equation says is that if one of the states in the possible-set of states(S_i) has a policy action that doesn't match with another, then the likelihood of extra-sensing action (a^+) can increase by $p_u(S_i|s*)$. For any given state, the number of possible states of uncertainty are likely to be few, not the full powerset (2^S). For example, in a gridworld setting, the current state's possible-sets may involve neighboring states (positions), but not ones further away.

The extra-sensing action could translate to many types of actions; these could include calling a supervisor or colleague for help, or re-checking state features. The specific dynamics of the extra-sensing actions are domain dependent. For this paper's presentation, we limit the effect of extra-sensing actions in that it can improve the inference likelihood of the human agent, but doesn't change the ground truth state. In terms of the problem definition, this means the state transitions from a state in S to the corresponding state in $S2$ are like in Figure 5.1.

In addition to taking extra-sensing actions, the other effect of uncertainty is on the likelihood of choosing a policy action from the given policy π_d, and is as follows:

$$\pi_p(s*, a|\pi_d) = (1 - \pi_p(s*, a^+, \pi_d))\times$$
$$\sum_{s_i \in S} p_c(\hat{s_i}|s) * 1[\pi_d(s_i) = a] \tag{5.2}$$

This says that the likelihood of an action in a state, depends on what states the human infers. If states with action a_1 are more likely inferred in the current state, then a_1 is more likely to be taken. This probability is multiplied by the probability

of *not taking* the extra-sensing action in the state; this ensures the probabilities sum to 1. Finally, the overall value of the *determinisitic* policy π_d given to the human is defined as:

$$V(\pi_d) = \sum_{s \in S} p_i(s) * V_{\pi_p}(s) \tag{5.3}$$

where $V_{\pi_p}(s)$ is the state value in the underlying MDP with the stochastic policy π_p which included the effects of uncertainty. The value is a weighted sum of state value, where the weights are the initial state likelihood given in $p_i(.)$.

5.3 Computing Human Model Parameters

Our work focuses on the computing the policy for a human model defined in terms of $\langle p_c, p_u, \psi_0, \psi_1 \rangle$. Part of the appeal of this approach for us in modeling the human agent, is that we only need probabilities of events. We do not have to make any assumptions on inference-limitations such as bounded or noisy-rational assumption on human inference [93], [112]. The probabilities can be estimated from empirical data. We present an empirical approach to collect the data needed to compute the probabilities; we use this approach in our human subject studies as well.

Since the model parameters are dependent on the state, we can collect data by testing the human agent on just the task of state detection. The human agent is presented multiple instances (trials) of each state –state samples are ordered randomly– and asked to look at the state following a predefined perception-policy. The perception policy could be as simple as a time limit on looking at state information before acting; for example, in manufacturing, a time limit is important as it translates to cost. Alternatively, the perception policy could be a predefined series of perception actions. After the perception-policy, we ask what they think the possible states are (can be more than one) and what they think the most-likely state is. Most impor-

tantly, we ask if they would like to confirm their answer of they think the correct state is, or look at the state again (extra-sensing) before confirming their answer. For this process, we would count the following for each state:

- $C_i(\hat{s}_i|s^*)$:The number of times a state (s_i) was inferred (most-likely state to the human) for a given state (s^*).

- $C_u(S_i|s^*)$: The number of times the person inferred a set of possible states ($S_i \subset 2^S$) for a given state. This includes the empty set, and singleton sets with one state. The count $C_u(\{s_i\}|s^*)$ of a singleton set $\{s_i\}$ will be atleast as much as $C_i(\hat{s}_i|s^*)$; it can be greater if (for one of the trials) the human only considers one state as possible but is also uncertain, and takes an extra-sensing action.

- $C_{e0}(s^*)$: How often a person took an extra-sensing action when uncertain, *and* their set of possible states was either 1 or none.

- $C_{e1}(s^*)$: How often a person took the extra-sensing action *when* they reported they were uncertain over two or more possible states.

Note than when a person takes one or more extra-sensing actions, we consider all subsequent counts separately; these counts are used is to compute the human model parameters for states in the set S_2, separate from S.

Using the data collected we compute the human model parameters as:

$$p_c(\hat{s}_i|s^*) = \frac{C_i(\hat{s}_i|s^*)}{\sum_{s_j \in S} C_i(s_j|s^*)} \tag{5.4}$$

$$p_u(S_i|s^*) = \frac{C_u(S_i|s^*)}{\sum_{S_j \in 2^S} C_u(S_j|s^*)} \tag{5.5}$$

$$\psi_0(s^*) = \frac{C_{e0}(s^*)}{\sum_{S_j \in \{S:S \in 2^S, |S| \leq 1\}} C_u(S_j|s^*)} \tag{5.6}$$

$$\psi_1(s^*) = \frac{C_{e1}(s^*)}{\sum_{S_j \in \{S:S\in 2^S, |S|>1\}} C_u(S_j|s^*)} \tag{5.7}$$

These definitions are so that ψ_0 captures the likelihood of taking extra-sensing action even without any inference conflict (but the human was not confident in their inference), or the human was unable to infer any state. On the other hand, ψ_1 is the likelihood of taking extra-sensing actions when uncertain; this covers the cases when the human's inference results in 2 or more states being possible (conflicting inference). If ψ_1 is much less than 1, it would mean that the human decides to act more often than resolve uncertainty even if uncertain. One can think of ψ_1 as a reflection of the pressure to act on the human agent, or a reflection of their patience to resolve uncertainty.

5.4 Policy Computation for POMDP-HUE

Finding an optimal solution to the POMDP-HUE problem is at least as difficult as computing a reactive (memoryless) controller for a POMDP, which is what our problem reduces to if one ignores the extra-sensing action; this can be done by setting $\psi_0 = 0, \psi_1 = 0$ for all states. Computing a reactive controller has been shown to be NP-hard ([69]). To handle this computational complexity, we present two algorithms. One is a hill-climbing algorithm for computing good albeit suboptimal policies quickly, and to handle larger state spaces. The other is a branch-and-bound algorithm for computing the optimal policy at higher computational cost, which is suitable for smaller state spaces and also for bounding the suboptimality of the hill-climbing approach for larger state spaces.

5.4.1 Human Agent Policy Iteration(HAPI)

We call our hill climbing approach Human-Agent Policy Iteration (HAPI) which takes the greedy best step to change the policy while accounting for human agent's uncertainty effects. In HAPI we start with a random deterministic policy (π_d), and compute the corresponding stochastic policy after state aliasing (π_p as defined by equations 5.1 and 5.2). We then determine the value of this stochastic policy by equation 4.2. Then (in the hill climbing step) for each possible policy change we compute the new policy value, and select the action to change the policy. This is repeated until no better changes can be made. Each step's computational complexity is $O(|S|^4|A|)$; this is because each step tests a number of changes no more than $|S||A|$, and the value of a fixed policy can be computed in $O(|S|^3)$ by computing the state transition likelihoods for that policy and using the following closed form computation in Equation 5.8(standard equation for value computation in a Markov Reward Process (MRP) (See [54] for more details on Markov processes):

$$\vec{v_s} = (I - \gamma * P_{ss'})^{-1} * \vec{r_s} \tag{5.8}$$

Where $\vec{v_s}$ is the vector of state values, $P_{ss'}$ is the transition probability matrix for a given policy, and $\vec{r_s}$ is the vector of expected rewards at each state (which can be computed for a fixed policy).

The total time taken for HAPI will naturally be problem specific; the number of improvement steps will depend on the initial point and the possible improvements in the domain. Additional random restarts can improve the outcome, as is common in hill-climbing approaches.

5.4.2 HUE Branch And Bound Policy Search (H-B&B)

HAPI is helpful to quickly find a good policy. However, if one wanted the optimal policy, then the following branch and bound approach –which we will refer to as H-B&B can be used for smaller state spaces. It can also be used to bound the suboptimality of the policy found by HAPI, which can be used to decide if further iterations of HAPI would be worthwhile or not.

This branch-and-bound searches in policy space by choosing an action for a state at each level in the search tree. We assume the reader is familiar with the basics of branch and bound [14]. At any given point in the policy search, only a partial policy is defined. We need a lowerbound, and an upperbound to determine if the node in the search tree should be expanded. We set the initial lowerbound as the value of the policy output by HAPI search.

We still need a helpful upperbound that accounts for the extra-sensing action. To compute this, we use an MDP relaxation of the POMDP-HUE for a given partial policy. This is done by assuming perfect state observability *only* for the remainder of the undefined states (policy not yet assigned), and using a lower-bound for the likelihood of errors and extra-sensing actions for the other states. We call this a "Partially-Controlled MDP" (PC-MDP). We compute the optimal policy (including extra-sensing actions) for this PC-MDP using value iteration and that is the upperbound. This idea of using an easier MDP to bound the state-value in branch-and-bound is similar to the bound employed in [73] except theirs does not consider or allow any notion of extra-actions. The gist of it is as follows: If one can set a lower-bound for the probability of state-misidentification and extra-sensing actions in all states, then by optimizing for the remainder of the policy action probability in each state, the policy-value obtained will be equal to or greater than any other

possible policy completion. A trivial lower-bound would be to assign zero probability to errors, i.e. $s_1 \neq s_2 \rightarrow p_c(s_1|s_2) = 0$, and extra-sensing actions ($\pi_p(s, a^+|\pi_d) = 0$) for the states whose policy is not yet defined. Then optimizing the PC-MDP policy would give the upperbound for state-value. Our bound considers the effect of prior decisions in H-B&B to give a tighter, more helpful upperbound for the search process. This is done by using the human model H parameters and lower-bounding the likelihood of extra-sensing action by removing undefined states from the probability computation in Equation 5.1. The pruning effects of our upperbound will be shown in the results. The proof for our upperbound will follow shortly.

In each step in the branch and bound, we need to run value iteration on the PC-MDP. In our algorithm, we stop value iteration after a certain number of iterations; we set number of iterations(k) to 1000 in our experiments. We then take an upperbound for each state's value computed as $v_k(s) + \frac{\epsilon * \gamma}{1-\gamma}$(Chapter 17 [87]) where ϵ here is $||v_k - v_{k-1}||$. This error is added to the policy value to set the upperbound.

The size of the policy search tree is unfortunately large; it is $|A|^{|S|}$ if we assume the same number of actions ($|A|$) in each state. However, a good upperbound and ordering the states intelligently can greatly prune the tree. We order the nodes in the search tree using the following score:

$$score(s) = (\frac{1}{|S|} + p_i(s)) \times \sum_{s' in S} p_c(s|s') \times \max_{a \in A(s')} r(s', a) \tag{5.9}$$

This function increases the score of a state based on how likely a state is to be the initial state ($p_i(s)$) since those state values determine the overall policy value (Equation 4.2). It also considers the likelihood that other states are confused with it, because the policy decision for those states will affect the state value for others too (due to state confusion). This likelihood is scaled by the max reward possible in the other states. The scaling is because we want to order policy decisions for states

based on how much they influence the policy-value; so we prioritize decisions affecting higher reward/lower cost states. This score (Equation 5.9) can help us make pivotal policy decisions sooner in the search process, and work with the upperbound to prune the search tree faster. In the next section we will discuss the upperbound used in H-B&B and how it is computed

5.4.3 Upperbound for Partial Policy Completions

Before we go into the details of how the upperbound is computed, let us review the equations that define how the stochastic policy –which is actually executed– is a function of the deterministic policy, the classification probabilities, and uncertainty probabilities.

The first pertinent equation is the likelihood of extra-sensing action is a function of the state as well as the policy defined over other states; this was defined in Equation 5.1. The likelihood of normal policy actions (not extra-sensing) was defined in Equation 5.2, and the value of a policy was defined in Equation 5.3.

The key idea used is that we can get an upperbound by relaxing the POMDP –partial observability of the state due to human uncertainty– into a perfect observability setting, like the assumption made in [73]. We call this relaxed MDP, a partially-controlled MDP (PC-MDP) for reasons that will become clear. During policy search in H-B&B each node in the search tree is a partial-policy; by partial-policy we mean that only some of the states have an assigned action in π_d , which is the deterministic policy to be given to the human. This partial policy needs to be kept in the PC-MDP for it to be a useful bound. Some key differences with the Branch and Bound approach in prior work [73] is that we do not use a cross product between the policy graph and MDP (which doubles the state space), *and* we need to account for the extra-sensing action. In order to do so we leverage the following

theorem:

Theorem: A partial stochastic policy whose every state-action probability is the same or lower than those of another partial policy, can be completed and optimized to get the same or better value as any completion of the other partial policy

This is best understood starting with an illustration in Figure 5.2.

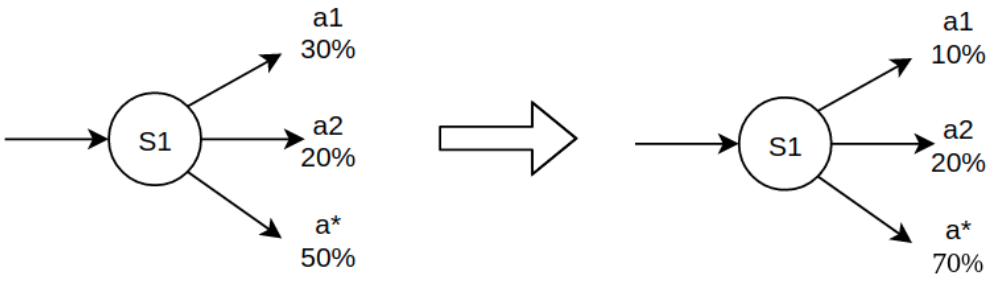

Figure 5.2: Example of a Partial Stochastic Policy (Left) Whose Probabilities for Every Action (A1,A2) Are the Same or Lower Than the Right Policy. "A^*" Is the Remaining Likelihood That an Action Can Be Assigned to (Optimizable).

Note that it is not enough that the total probability that is optimizable is lesser for a completion to have a better or equal policy-value than another. Even if the overall sum of probabilities for the partial policy is lower, if any one action has more probability than the other partial-policy, then this statement is not guaranteed to hold. A very simple example of this case is shown in Figure 5.3 in a single-state problem with two actions that loop back (a multi-arm bandit with discounted future rewards). In this even if we reduce the overall fixed action-probability from the policy on the left, the policy on the right will not have a greater possible policy-completion (see Figure for details).

Let us now see how the upperbound value is computed for H-B&B. In the PC-

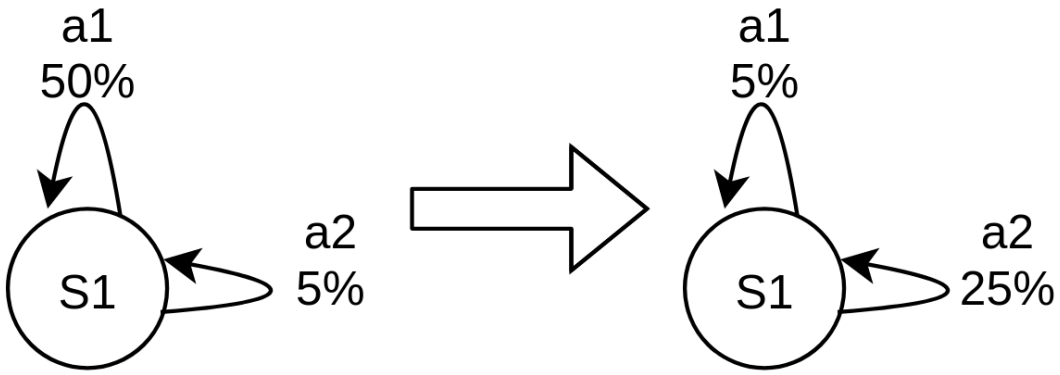

Figure 5.3: The Policy on the Right–While Having Lower Total Probability Used Up–Will Not Have a Higher Value for Any Policy Completion Than the Optimal Completion for the Policy on the Left, if A1 Has a Higher Reward Than A2 Since the Probability of A2 Is Larger in the Right-Policy

MDP at a search node in H-B&B, each state might have some of their actions fixed with a certain probability (hence Partially-Controllable MDP). This partial-policy comes from the prior decisions made during policy search in H-B&B. This PC-MDP can be converted into an equivalent MDP by updating the action transitions and rewards to account for the partially-defined policy actions (combining them); this is done by taking the expected sum for transitions and rewards. The probabilities used for this expected sum are determined by the partially defined policy at each node in the policy-search tree. In this derived MDP, any deterministic policy will have a policy-value greater than or equal to any stochastic-policy in the derived action space. Note that the statement "a deterministic policy's value will be greater or equal to that of a stochastic policy" is true for all MDPs (well established).

Now let's connect this to the example in Figure 5.2; in it the partial-policy on the right has every defined action's probability as the same or lower to the case on the left. So any policy-completion –assuming policy in other states are the same– for the

case on the right will have greater or equal value than any possible policy-completion for the left-case. So if we can *lower* bound the defined action's probabilities based on the partial-policy at each search node, the optimal policy-value in the derived MDP would be an upperbound for any policy-completion of that partial-policy. Having defined how the upperbound is determined for H-B&B, let's apply it to the search process.

5.4.4 Upperbound as Applied In H-B&B

As we have seen in equation 5.2, a determinisitic policy translates to a stochastic policy after accounting for uncertainty effects. Using the same functions, a partial deterministic policy at each node of the H-B&B search space can be mapped to a partial stochastic policy. If we could do so, then we could relax the setting to a fully observable MDP and compute the optimal value of the PC-MDP as an upperbound using the logic stated previously.

The problem with what we have said so far is that we *cannot* know the partial probabilities of the actions (including $a_\emptyset$) until the entire policy is complete. So instead, we compute a lower bound on final action probabilities in a state. The lowerbound for delay probability ($\pi_{sa}(s, a_\emptyset, \pi_d)$) is computed by considering that all undefined state policies will *not* conflict with the state-policies defined thus far, i.e. $\pi_d(s_i) == \pi_d(s_j)$ where s_i represents the state's whose policy is defined, and s_j is an undefined state. Then the probability is computed as per equation 5.1. So now we have lower-bounded the probability of a^+. We still need to lower-bound the probabilities for the normal policy actions.

For the policy action probabilities, the lower-bound likelihood changed from Equation 5.2 to Equation 5.10. In this, we use the *maximum* probability of extra-sensing actions possible from any policy completion. The maximum possible delay is com-

puted by considering the worst case of policy conflicts; every state that can be misclassified and not yet assigned a policy action will have a different action to the current state. This results in the maximum possible likelihood for extra-sensing action.

$$\pi_{sa}(s, a, \pi_d) = (1 - max\_delay(s, \pi_d)) \times \sum_{s_i \in S} p_c(\hat{s_i}|s) * 1[\pi_d(s_i) = a] \tag{5.10}$$

Now we have probabilities for actions in the current partial-policy that are less than or equal to the probabilities for the same actions in any possible policy completion. With these probabilities we can get the optimal policy-value of the associated "PC-MDP" (discussed in the previous section) by value iteration. As previously discussed, the policy value –when all the fixed action probabilities are lower– will be greater than or equal to any policy completion on the original partial policy.

This upperbound was verified in our experiments, that the bound converges from above(monotonic and non-increasing) to the true policy value for any completion of the partial policy. To ensure that it is an upperbound during actual computation, we add MDP value-iteration error bounds to the policy value computed. The policy value used after k-iterations of value-iteration becomes $v_k(s) + \frac{\epsilon * \gamma}{1-\gamma}$(Chapter 17 [87]) where ϵ here is $||v_k - v_{k-1}||$. Alternatively, one could set a target error ϵ and determine the number of iterations needed as $\lceil \frac{log(2R_{max}/(\epsilon(1-\gamma))}{log(1/\gamma} \rceil$ [87].

With the error bound added, we have a true upperbound for H-B&B at every node in the search. We will present empirical results to show that the number of nodes pruned in the search tree is appreciable, and that this is not a trivial bound.

5.5 Experiments and Results

We tested our algorithms on two sizes of gridworld domains, 5x5 and 10x10 to see how the performance changes with state space size. We also tested our approach on the warehouse worker domain introduced in the previous chapter.

5.5.1 Gridworld Experimental Setup

The dynamics in this gridworld are as described in the previous chapter. For testing our algorithms, we repeated HAPI ten times (10 random restarts) for each experimental setting and consider the best value as the output from HAPI. As for H-B&B search, it ofcourse need only be run once.

First we present the Gridworld experiments on 5×5 grids where H-B&B was allowed to run to completion. We varied the properties of the MDP to see how well HAPI performs compared to H-B&B. The actions for each state (defined by agent position) are the standard ones; these are move up, down, left, and right. The goal is the bottom right square like in Figure 5.6, albeit *without* the colors (colors are used in the human subject studies). The goal state is an absorbing state, and the reward is 100 upon transitioning into it. When the agent takes the extra-sensing action (a^+), it reduces by half both the error likelihood ($p_c()$ of incorrect state detection) and probability of incorrect possible-sets p_u (possible-sets that have states other than the ground truth). Taking a^+ results in a state transition to a parallel state in S_2 (as in problem definition and Figure 5.1). S_2 states have the same action effects but with different p_c and p_u functions. Subsequent extra-sensing actions from this state, returns to the same state. This means additional extra-sensing actions from S_2 does not change the inference outcomes (p_c and p_u) of the agent in our experiments.

All action transitions are stochastic with a 5% chance of transitioning to a random

neighboring grid position or stay in place. All actions will have a random cost for each experimental setting. An invalid action (like moving up from the top of the grid) results in the agent staying in the same state and incur the cost assigned to that state and action. The extra-sensing action has a cost of 1.

As for the likelihoods of confusing states (p_c), we define the likelihood of confusing a grid state (position) with another based on the L1 distance between positions. p_c is defined in Equation 5.11. The equation is simply saying that neighboring states are much more likely to be confused with each other than with those further away.

$$p_c(s|s') = \frac{1/(L1(s,s') + \mathbb{1}[s = s'])^m}{\sum_{s'' \in S} 1/(L1(s,s'') + \mathbb{1}[s = s''])^m} \tag{5.11}$$

where m is a scalar. We set it to 5 for our experiments. This makes the likelihood of confusing one state with another that is more than 1 step away to be very small. Lastly, we add the +1 to avoid dividing by zero. As for the possible-sets in p_u, we limit ourselves to sets of size 1 and 2. The probability of each set ($p_u(.)$) are computed using equation 5.12.

$$\begin{aligned} p_u(\{s_2, s_1\}|s*) = p_c(\hat{s_2}|s_1) * p_c(\hat{s_1}|s*) + \\ p_c(\hat{s_1}|s_2) * p_c(\hat{s_2}|s*) \end{aligned} \tag{5.12}$$

Note s_1 can be the same as s_2 in Equation 5.12; those cases correspond to the $p_u(.)$ probability for possible-sets of size 1. Lastly, in the agent model, ψ_0 was set to 0.05 –which means when there is no inference conflict the agent may still take a^+ 5% of the time– and ψ_1 to 0.9.

With respect to the experimental settings, we first present results of a 5x5 grid, with one additional mental-state (S_2 state) per state. We used a smaller grid first because while HAPI (hill-climbing) can handle larger sizes of grids, branch and bound

(H-B&B) solving speed drops very quickly. This is because the policy space grows as $|A|^{|S|}$; even a 5x5 grid has $4^{25} \approx 1.1 \times 10^{15}$ policies. However, the search process eliminates most policies quickly, and the bound helps immensely with pruning the policy space (which we will show in the results). Engineering improvements to speed up H-B&B through parallelization and memory management is left as future improvements. We posit that for a single task's policy (for a human agent) even state-sizes around 25 can be sufficient for some problems. For example, a basic car-maintenance policy for owners would have a few states and associated actions to deal with issues such as oil-change, car battery-health and such. We also present results for a 10x10 gridworld setup, where the branch and bound approach serves to bound the suboptimality of the policy found by HAPI.

To evaluate the algorithms, we focused on varying three parameters: (1) The discount factor γ, whose default value is 0.7; (2) the likelihood of random actions whose default value is $\rho = 0.05$; (3) A "reward noise range" parameter (RNR) to add random rewards to each of the actions in the grid and whose default value is 2; an $RNR = 2$ would result in random rewards for each action in the range $[-1, 1]$, i.e., uniformly distributed about 0.

We chose to vary the discount factor since a larger discount factor couples the policy decisions more strongly (since the state value is affected by states further away). We also chose to add random rewards to each of the actions other than the goal actions to make the search more challenging. Lastly, increased random action likelihood meant that states which had both high reward and high cost actions are less attractive than if there was no random action likelihood.

Our tcode was implemented in python using "pybnb" library for branch and bound, and PyTorch and NumPy for matrix operations. The experiments were run on a PC with Intel® Core™ i7-6700 CPU, running at 3.40GHz on Ubuntu 20.04 with 32 GB

of memory.

5.5.2 Gridworld Results

We first present the values of the policies discovered for the 5x5 grid experiments in Table 5.1. The best policy discovered by HAPI approach was either very close to optimal, or optimal in all cases. The takeaway is that for this experimental setting, HAPI with 10 iterations found either a very competitive value or the optimal policy value in all cases. We found this to be the case in the 10x10 grid setting as well (with the random costs and stochastic transitions) in Table 5.2. For those experiments, we only used H-B&B to find an upperbound and use it to define the suboptimality of the policy value found by HAPI; the last number in each table entry is the ratio with the upperbound. In the 10x10 grid setting as well, HAPI was able to perform well. A related point is that H-B&B was able to find a good upperbound to the policy value within 30 minutes, which can be helpful in deciding if further iterations of HAPI might be worthwhile or not.

Discount Factor (RNR=2, ρ=0.05)		**Reward Noise Range (ρ=0.05, γ=0.7)**		**Random Action Probability (RNR=2, γ=0.7)**	
γ	Values	RNR	Values	ρ	Values
0.3	11.87, 11.87, 1.0	0	33.67, 33.67, 1.0	0.05	34.16, 34.17, 1.0
0.5	19.05, 19.05, 1.0	1	33.89, 33.89, 1.0	0.1	33.65, 33.65, 1.0
0.7	34.16, 34.17, 1.0	2	34.16, 34.17, 1.0	0.15	33.08, 33.08, 1.0
0.9	69.45, 69.45, 1.0	4	34.75, 34.75, 1.0	0.2	32.46, 32.46, 1.0

Table 5.1: Policy Value Results for a 5x5 Grid. Each of the Primary Columns Changes One Experiment-Parameter, and Holds the Other Two Constant. Each Entry Has the Best Policy Value From Hapi, the Optimal Value Found by H-B&B, and the Ratio of the Two. All Values Rounded Down to Two Decimal Places.

Discount Factor (RNR=2, ρ=0.05)		**Reward Noise Range (ρ=0.05, γ=0.7)**		**Random Action Probability (RNR=2, γ=0.7)**	
γ	Values	RNR	Values	ρ	Values
0.3	3.17, 3.48, 0.91	0	11.14, 11.47, 0.97	0.05	11.56, 12.29, 0.94
0.5	5.26, 5.65, 0.93	1	11.27, 11.83, 0.95	0.1	11.17, 11.86, 0.94
0.7	11.56, 12.29, 0.94	2	11.56, 12.29, 0.94	0.15	10.77, 11.4, 0.94
0.9	43.9, 45.91, 0.96	4	12.28, 13.48, 0.91	0.2	10.36, 10.97, 0.94

Table 5.2: Policy Value Results for a 10x10 Grid. Each of the Primary Columns Changes One Experiment-Parameter, and Holds the Other Two Constant. Each Entry Has the Best Policy Value From Hapi, the Best Upperbound Found by H-B&B After 30 Minutes, and the Ratio of the Two. All Values Rounded Down to Two Decimal Places.

To show the effectiveness of the bound used in H-B&B for pruning the policy search space, we present in Table 5.3 the number of nodes opened by our branch and bound search before finding the optimal solution for 5x5 grid. The number of nodes is significantly less than the policy space of 5^{16}, which is helpful because the cost of each node is high since we run value iteration on each node for the associated PC-MDP. This also shows that our upper bound was helpful in pruning the policy search tree.

Discount Factor (RNR=2, ρ=0.05)		**Reward Noise Range (ρ=0.05, γ=0.7)**		**Random Action Probability (RNR=2, γ=0.7)**	
γ	Nodes opened	RNR	Nodes opened	ρ	Nodes opened
0.3	41713	0	20649	0.05	7981
0.5	5525	1	12573	0.1	7297
0.7	7981	2	7981	0.15	6785
0.9	11933	4	27589	0.2	6401

Table 5.3: Number of Nodes Opened by H-B&B Before Finding Optimal Solution, for a Policy Search Tree of Size 4^{25} In 5x5 Grid Experiments

Additionally the time taken by both algorithms are presented in Tables 5.4 and 5.5 for the two grid sizes.

Discount Factor (RNR=2, ρ=0.05)		**Reward Noise Range (ρ=0.05, γ=0.7)**		**Random Action Probability (RNR=2, γ=0.7)**	
γ	Time(sec)	RNR	Time(sec)	ρ	Time(sec)
0.3	30.09,1788.25,	0	31.51,1175.24,	0.05	31.3,521.97,
0.5	38.73,293.87,	1	33.53,769.04,	0.1	30.08,497.26,
0.7	31.3,521.97,	2	31.3,521.97,	0.15	31.7,498.22,
0.9	34.37,2148.27,	4	29.87,2285.57,	0.2	33.98,482.82,

Table 5.4: The Total Time Taken (In Seconds) By HAPI, Followed by the Time Taken for H-B&B in 5x5 Grid Experiment Settings

Discount Factor (RNR=2, ρ=0.05)		**Reward Noise Range (ρ=0.05, γ=0.7)**		**Random Action Probability (RNR=2, γ=0.7)**	
γ	Time(sec)	RNR	Time(sec)	ρ	Time(sec)
0.3	13887.44,1800.19,	0	11439.52,1800.36,	0.05	13978.69,1800.52,
0.5	15079.82,1800.31,	1	10434.4,1800.84,	0.1	12784.0,1800.6,
0.7	13978.69,1800.52,	2	13978.69,1800.52,	0.15	10832.57,1801.0,
0.9	11442.28,1802.08,	4	12596.2,1800.67,	0.2	9614.67,1800.86,

Table 5.5: The Total Time Taken (In Seconds) By HAPI, Followed by the Time Taken for H-B&B in 10x10 Grid Experiment Settings. H-B&B Was Terminated in 30 Minutes and Only the Tightest Upperbound Was Taken.

5.5.3 Warehouse Worker Experimental Setup

This domain is as introduced in the previous chapter; the state and action sets are defined by the cartesian product of the set of box sizes $\{small, medium, large\}$ ($\{s, m, l\}$), and if bubble-wrap (soft padding) is needed or not $\{\emptyset, w\}$. For example a group of items with glass items could be a small order that requires a small box with bubble-wrap, $small \times wrap$ (represented as $s \times w$). When a worker sees an order, it is not always apparent what box size is needed. Additionally, due to the diversity of products, the worker has no idea which products actually need bubble wrap or not. For example there maybe tempered (hardened) glass products that do not need bubble-wrap, but the worker might not know this and use bubble-wrap anyway.

In this work, we changed the transitions and rewards for the domain. The transition likelihoods are defined in Table 5.6, where the rows are states, and columns are actions. Each entry is a tuple of successor state and probability. The rewards for each state action pair are defined in Table 5.7

Thus far we have only talked of package sizes. The other dimension is whether

	l	l × w	m	m × w	s	s × w
l	S,16.66%	S,16.66%	{m},100%	{m},100%	{l},100%	{l},100%
l × w	S,16.66%	S,16.66%	{m,m × w},50%	{m,m × w},50%	{l × w},100%	{l× w},100%
m	S,16.66%	S,16.66%	S,16.66%	S,16.66%	{s,s × w},50%	{s,s × w},50%
m × w	S,16.66%	S,16.66%	S,16.66%	S,16.66%	{s,s × w},50%	{s,s × w},50%
s	S,16.66%	S,16.66%	S,16.66%	S,16.66%	S,16.66%	S,16.66%
s × w	S,16.66%	S,16.66%	S,16.66%	S,16.66%	S,16.66%	S,16.66%

Table 5.6: Transition Likelihood Matrix; Rows Are States and Columns Are Actions. Recall Actions Maps 1:1 to States and So Have Matching Names. Each Entry Is a Tuple of a Set of States From State Space S, and the Probability of Transition to One of the States in That Set. Note: Capitalized S Is the Entire Set of States

	l	l × w	m	m × w	s	s × w
l	1.0	1.0	0	0	0	0
l × w	0.9	1.0	0	0	0	0
m	0.9	0.9	1.0	1.0	0	0
m × w	0.8	0.9	0.9	1.0	0	0
s	0.8	0.8	0.9	0.9	1.0	1.0
s × w	0.7	0.8	0.8	0.9	0.9	1.0

Table 5.7: Reward for State,Action Pairs; Rows Are States and Columns Are Actions.

an order needs bubblewrap (soft packaging material) or not. If an order is large and needs bubble-wrap, then packaging it as medium without bubble wrap gives a lower reward and transitions to “medium-size with-bubble wrap" state 50% of the time, or medium-size (no bubble wrap) state for the other 50% of the time. This is to reflect that the user might have packaged the items that required bubble wrap already. The analogous transition happens with medium-sized orders that require bubble wrap.

As for the reward. The right action–right size and addition of bubble wrap if needed– gives a reward of 1.0. All other suboptimal actions gives a lower reward. How much lower is randomly determined by the Reward Noise Range (RNR) parameter.

We vary this parameter between $0.1, 0.2, 0.3, 0.5$. Based on this parameter the reward is randomly reduced by upto this amount for each action.

The other part of the experimental setup is the classification likelihoods which are shown in Table 5.8. The likelihood of a worker confusing any small order with any medium sized order or vice versa is about 16%, and is the same for misclassifying any medium with any large order. The likelihood of confusing a small order with a large order is less than 1%. The likelihood of the worker correctly determining if bubblewrap is needed is 50% across all order sizes; there are so many products that their accuracy for determining if bubble wrap is needed is random.

	l	l × w	m	m × w	s	s × w
l	32.68%	32.68%	12.50%	12.50%	0.98%	0.98%
l × w	32.68%	32.68%	12.50%	12.50%	0.98%	0.98%
m	16.34%	16.34%	25.00%	25.00%	16.34%	16.34%
m × w	16.34%	16.34%	25.00%	25.00%	16.34%	16.34%
s	0.98%	0.98%	12.50%	12.50%	32.68%	32.68%
s × w	0.98%	0.98%	12.50%	12.50%	32.68%	32.68%

Table 5.8: Classification Likelihood Matrix (p_c) For Warehouse-Worker Domain, Where (S,M,L) Stands for (Small,Medium,Large) And "W" Means Bubblewrap Needed. Columns Sum to 1000%.

The probability of uncertainty $p_u(.)$ is computed using the same method as for the gridworld experiments. The $p_u(.)$ probabilities were set by taking the average of the classification probabilities from each pair of states, computed as :

$$\begin{aligned} p_u(\{s_2, s_1\}|s*) = (\frac{p_c(\hat{s2}|s*) + p_c(\hat{s_2}|s1)}{2}) * p_c(\hat{s1}|s^*) \\ +(\frac{p_c(\hat{s1}|s*) + p_c(\hat{s_1}|s2)}{2}) * p_c(\hat{s2}|s^*) \end{aligned} \tag{5.13}$$

We ensured that the probability of the cases for any given state sum to 1. This equation

This would mean that if two states had a high probability in p_c of being inferred in the state $s*$, then the likelihood that the agent would be uncertain between those two states is higher too. Lastly, we set $\psi_1 = 1.0, \psi_0 = 0$ for our experiments. The extra-sensing action cost is -0.1. The extra-sensing action in our experiments does not improve the inference likelihoods. This represents the case when the human repeats unhelpful sensing actions, or is just delayed due to uncertainty.

In our experiments we vary the reward noise range (RNR) parameter as { 0.1, 0.2, 0.3, 0.5} for one set of experiments, these are presented in Figure 5.4. We also vary the discount factor as $0.3, 0.5, 0.7, 0.9$. These results are presented in 5.5. The default RNR is 0.0 when it is not being varied, and the default discount factor is 0.7. We also set the probability of taking a random action is 5% to add more stochasticity into the domain. For each setting we run HAPI 30 times and present the distribution of the policy value (Equation 4.2) normalized by the value of the policy returned by branch and bound. Since the branch and bound policy value is optimal , we expect the range to be [0,1].

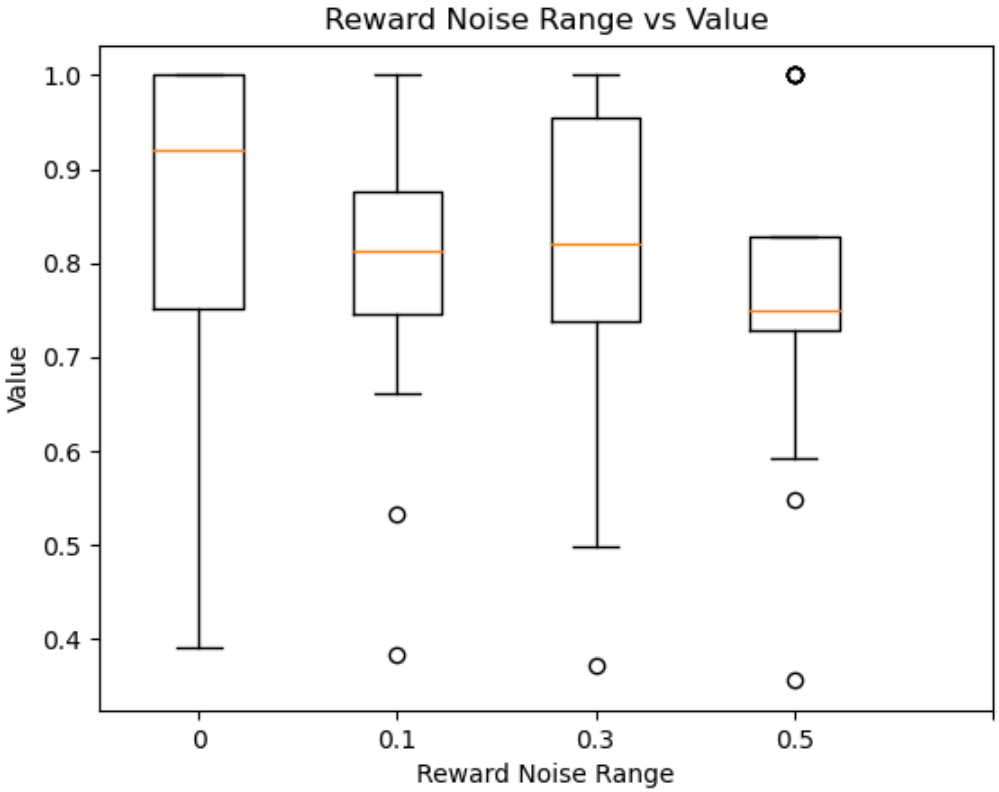

Figure 5.4: Box Plot of HAPI Policy Values for Warehouse-Worker Domain With Varying Reward Noise Range; Values Normalized by the Policy Value From Branch and Bound Search.

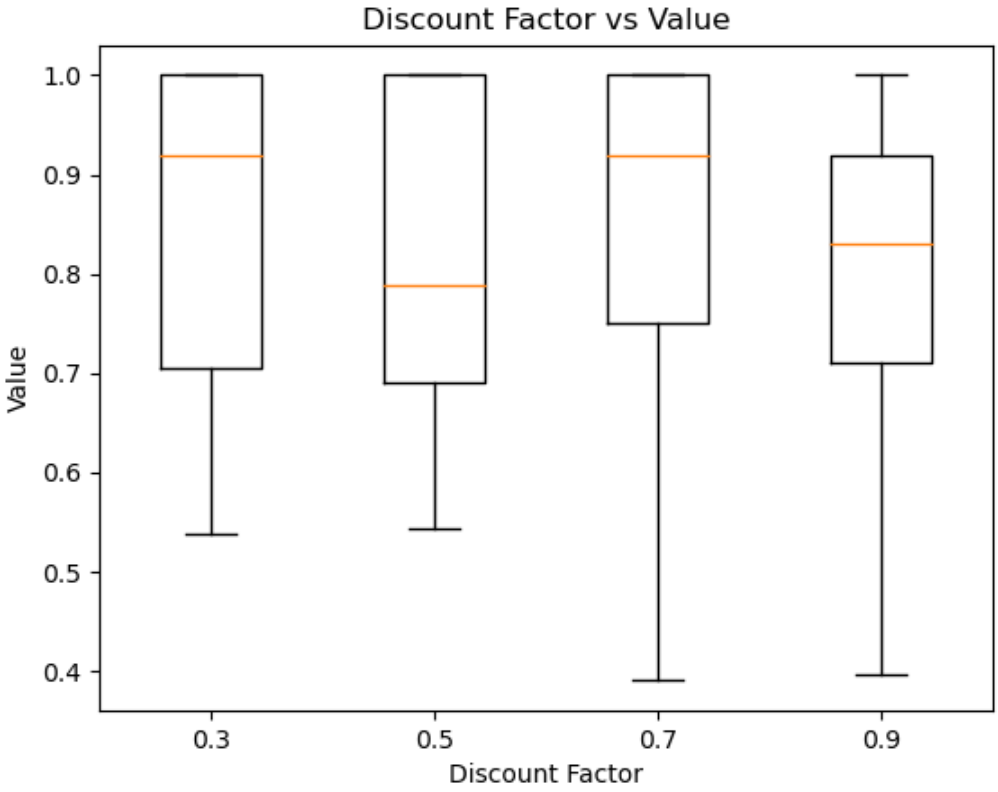

Figure 5.5: Box Plot of HAPI Policy Values for Warehouse-Worker Domain With Varying Discount Factor (Γ); Values Normalized by the Policy Value From Branch and Bound Search

5.5.4 Warehouse Worker Results

In all the experiments, as we wanted, the branch and bound policy has the highest value. The HAPI policy search found the optimal policy value atleast once out of 30 iterations, which is what we hoped. The variance in policy value found by HAPI is a lot more than in the grid world experiments. The trend of incresing variance in policy value for HAPI as discount factor increases is not as evident in this domain as it was in gridworld. This maybe because of fewer states.

The surprising trend is that as the RNR parameter was increased (less reward for actions) HAPI seemed to find the optimal policy a lot more consistently. It is easy to see why this holds in the limit when all the rewards are zero and the only cost is from inaction (extra-sensing action). Then the optimal policy is any policy that has the same action across all states, of which there are many and easy to find. We were surprised that the effect of the extra-sensing action was pronounced even with smaller values of RNR. Indeed the optimal policies found were just to put all state orders in the large-box with bubblewrap, which fits out intuition; if the difference in rewards is negligible, then use the action that applies to all states and avoids extra-sensing actions.

5.6 Human Subject Experiments

For our human subject experiments we wanted to see if our assumptions hold, with respect to human policy execution under state uncertainty. We use a small grid world setting, and use colors to define each state in order to introduce state uncertainty from perception. The full grid is as illustrated in Figure 5.6. In the underlying MDP, each action has costs as illustrated in Figure 5.6, and all undisplayed actions have a cost of -10. There is a reward for reaching the goal at the bottom right position (+10). After reaching the goal position, the state then changes to a random new position from a set of initial states.

Our objective in this study was to first build an averaged human model using the procedure described in Section 5.3, and then use that model to compute a policy that accounts for the uncertainty effects. The performance using this policy is then compared to the human agents' performance using the optimal policy for the underlying MDP. The state space was designed so that some sets of states (the color associated to them) are visually similar, like "Green 1" and "Green 2" in Figure 5.6, and cause uncertainty.

All participants were recruited using the "Prolific" service for online studies, and prescreened using using their service for vision (can see colors clearly). We also asked 3 questions at the start of our study to test if participants could distinguish between lighter and darker shades that look similar. All prolific participants are above 18 years of age, and equal division of male and female participants (as they identify) was requested for the study (Prolific handles this part). No other demographic information was collected. Gender or age based comparisons are out of the scope of this study. Our human-subject studies had IRB approval, and we gave clear information about the purpose of the experiments to our participants in the consent form before the

Figure 5.6: A Colored Gridworld Domain in Which an Agent Determines the States by the Color; Initial States Are Annotated as Well.

experiments. We also debriefed the participants, and gave the option to contact us for more information.

In the first phase, we collect data to build an (averaged) model of a human agent for the task. This means computing $< p_c, p_u, \psi_0, \psi_1 >$ for each state (including S_2 states). We do this using a preliminary study that displays the colors used in the main study, and asks the human to match a color displayed (colored square) on the left of the screen to a list of numbered colors on the right of the screen. The color was only shown for 0.5 seconds, but the table on the right was permanently displayed. They were given the option to see-again if they were uncertain by pressing the back arrow key; doing so gives us a clear indication of the extra-sensing action.

In this study, before the human can submit their answer or ask to see the color again, we ask them to enter their guesses as to which colors they think it could have

been. Example, if a color $Green1$ was displayed the user might enter $(1, 2)$ which are the indices for the two green shades, or just enter one color if they were confident in their decision.

The participants were paid a flat amount, as well as an additional 0.1 dollars for every correct answer as an incentive to get it correct. We used data from 16 participants for this phase, each participants was asked 15 questions; colors were randomly sampled during testing. We use the data collected to compute the parameters of the human model as per the equations in Section 5.3.

As one might expect, the two shades of green, and two shades of red causes participants to request to see the color again, as well as make the most mistakes. All the unique (non-similar) colors that we tested the participants with were easily identified with almost no errors and no extra-sensing actions. We saw this for many unique colors during our initial testing, and so felt confident that we were not showing the colors for too short a duration. One interesting note was that people seemed more likely to confuse the darker shade of red with the lighter shade than vice versa; this was not the case for the shades of green which was much less confusing.

After computing the human model parameters, we use it to compute the optimal policy using H-B&B for the grid MDP in the study. We also computed the optimal policy by value iteration which ignores state-uncertainty. These were the two policies given to the humans in phase 2 of the study, and are displayed in Figure 5.7; the policy on the left is the optimal policy of the MDP (ignores) uncertainty, and the policy on the right is the one that accounts for uncertainty. The discount factor γ was set to 0.9 to compute the policies.

Just as in phase 1, we display a color on the left of the screen for 0.5 seconds. This color corresponds to their current position in the grid as in Figure 5.6. We ask the participant to press the arrow key corresponding to the color seen using the policy

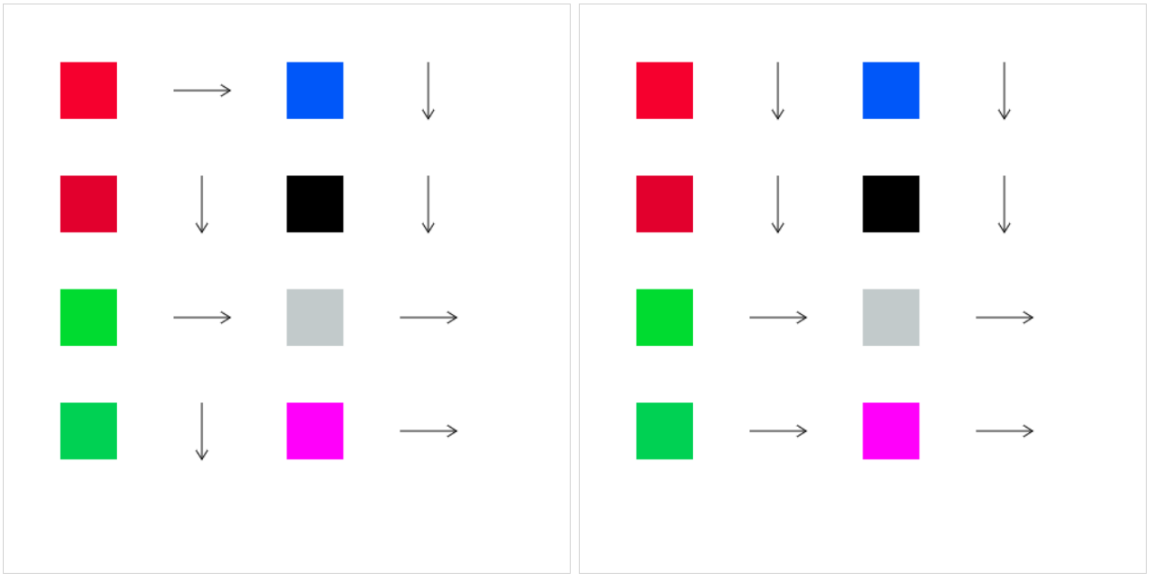

Figure 5.7: The Two Policies for the Second Phase of Human Subject Experiments. The Left Policy Is the Optimal Policy *Without* Considering the Effects of Uncertainty, and the Right Is the Optimal Policy After Accounting for Uncertainty.

displayed on the right, and so navigate through the underlying grid. After each step the color of the new position is displayed. The participant can see a color again by pressing the "Control" button (extra-sensing a^+ action). We track every button press and their progress through the grid. When they reach the goal state, the state is reset to another of the initial states, for a total of 3 times. The initial states are shown in Figure 5.6. We used 20 participants for this phase, and had to drop one data point as the data suggested they were randomly guessing for both policies.

Using the data collected, we wanted to see if using the policy that accounted for uncertainty translated to more reward accrued. We took the difference between the reward accrued in each run (initial to goal state) with it's corresponding run in the other policy. On average the cost incurred by the policy that accounts for uncertainty is less by 1.45. We ran a dependent t-test for paired samples; the same participant

did both corresponding runs, and so we treat the data as paired samples. We set the significance level for t-test at $\alpha = 0.05$, and ran the paired t-test using the scipy-stats library in python [104] to see if the policy with uncertainty was better (one-sided test). We got a t-test statistic value of 2.240 which corresponds to a p-value of 0.0147. Thus we can reject the null hypothesis that the policy *with* uncertainty gives the same or worse value than the optimal policy without uncertainty.

For our experiments, given the simplicity of the problem we found it sufficient and easier to build an averaged human model and use it for all participants. Ideally one would build a human model unique to a person. Such an averaged model could be tuned with fewer additional data points per person, using a Bayesian approach to computing the parameters (using a dirichlet distribution for tracking the priors). Overall, our human-subject studies give support to the idea that policies which consider state uncertainty are executed faster and more reliably.

5.7 Related Work

If the effect of uncertainty was limited to only erroneous state detection (as captured by $p_c(.)$ in the human model), one can frame the problem in this chapter as computing the reactive controller for a POMDP and use prior methods in [69] and [73]. However, none of the prior methods handle the case where additional (extra-sensing) actions are taken by the agent due to state uncertainty or any function of policy decisions. This can result in different policies between our approach and prior reactive controller approaches if such actions are indeed taken; We verified this by setting ψ_1, ψ_0 to be zero (so no extra-policy actions are taken). This reduces our H-B&B to computing the standard POMDP reactive controller based on $p_c(.)$ as the observation likelihood. We found that the policy and values were suboptimal when there were higher costs of extra-sensing actions; this fits our intuition as the reactive

controller is explicitly ignoring those.

An extension of reactive memoryless controllers is history-based controllers [63] which considers mapping history of observations to policies as opposed to just the current observation (state history of 1). These policies might work well for human agent execution as one need only consider the history of states and not ask the human to explicitly track prior probability distributions, which is a much higher cognitive load. However, such policies are still susceptible to state uncertainty in the human's mind for the same perceptual and cognitive difficulties as in the single observation (single history) case. For this work we limited ourselves to history of 1 state. We will consider extending this to longer histories in future work, and we do not think it a trivial extension as the human model needs to consider sequences of possible-states, and how state-action decisions affect the uncertainty.

On a different but related note, there is work that considers "blindspots" in an agent's representation [81]. These blindspots can arise due to a mismatch in the state space during training versus execution, or limitations in representational capabilities. There is a follow-up work that focuses specifically on a human agent's blindspots [83] and reducing errors. The approach here attacks the problem from a different angle, and do not try to minimize the errors for a given policy; we instead try to compute a policy that accounts for human errors and behavior in response to uncertainty.

Lastly, there is work that considers how human decisions can change based on when the outcome is sampled or when they have to make their decision, and that work is aptly called Decision Field Theory. It considers that the outcome of human choice between different prospects (of different risks) will vary based on the time of sampling the decision. If we apply this to our problem in this chapter, we might consider the effects of time restrictions on human policy behavior. We could analyze how the errors and likelihood of extra-sensing actions change if the human is forced to

act within a fixed time period. This is left as a direction to be explored in extensions to the approach herein.

5.8 Summary and Extensions

In this chapter, we define the problem of computing a reactive policy that accounts for human execution behavior under state uncertainty. This work tackled the cognitive limitations for inference in state identification when executing a policy, and we also considered the human response to uncertainty during that inference. We formalized a probabilistic model of the human agent's inference and behavior, as well as how to compute the parameters in it. We then presented two algorithms (HAPI and H-B&B) to compute policies for our problem, and show experimental results in a gridworld setting.

Lastly, we conducted human subject studies to show an example of how the human model can be empirically derived, and use it with our H-B&B algorithm to compute the optimal policy for our problem. We show that this policy resulted in statistically more reward accrued than the optimal MDP policy that ignores the effects of uncertainty. Our human-subject studies supports the considerations we make for our human model such as expecting identification errors between similar states and extra-policy actions.

Thus far in the dissertation, we have discussed how to compute robot policy to make human decisions easier (Chapter 3), how to compute policies that are easier for humans (Chapter 4), and policies that consider human behavior under uncertainty (Chapter 5). Next we look at a different problem in SDM, that of the human as a planner, and present an approach and interface for co-planning with an automated planner.

Chapter 6

CO-PLANNING IN FACTORED STATE SPACES

In Chapter 3 we discussed how reasoning about robot motion plans can be made easier, and the human can navigate the space with less effort. However, humans reasoning about, and planning in factored (and abstract) state-spaces is a different challenge. When human agents are co-planning with automated planners, it would be unreasonable to assume that humans have the same memory and recall of pertinent information as a machine, and so we sought to tackle this issue. In this chapter, we incorporate human-factors design principles from Ecological Interface Design [102] to develop an approach and interface for co-planning in factored spaces. This includes visualization of state features and plans, showing pertinent action information, and co-planning through sub-goal specification.

6.1 Motivation

One of the barriers to the adoption of automated planners is their usability. This is due to the amount of time and knowledge needed to interpret any output and interact with the planner. An area in which we can improve usability is plan trace and domain visualization. Current plan trace visualizations often represent plans in a pipeline or linear fashion (as we will see). If there is no complete ordering of actions, adjacent actions may have no immediate relationship or dependence. So the user would have to keep the effects of actions in mind, and connect it with a future action to realize the need for the prior action. Consequently, the user may have to parse the entire plan, before beginning to conceptualize about other possible plans. This is because the user needs to know about the dependencies across the plan. This high cognitive load

often leads to mental fatigue in the user, which reduces the quality of plan criticism in mixed-initiative planning. Thus, it is important to present information in a visual and easy-to-parse (and recall) format. This would allow users to quickly generate alternate plans, or modify existing ones and compare them.

To address such problems, we can use the Ecological Interface Design (EID) Principles [102] –which helped set the standards for design in complex human-machine systems– specifically the EID principle that require that the correct affordances (actions) are easily inferable to the operator. To this end, we introduce TGE-viz (Transition Graph Embedding visualization), a visualization approach that uses ideas from graph embeddings to display the transition graph of a domain in 2 dimensions. Graph embeddings are a popular method to reduce the vast amount of information in graphs by embedding their vertices to a continuous space to speed up analytics [16]. Lower dimensional embeddings allow humans to be involved in the analysis. We can intuitively see structures (clusters and shapes) and relative distances, which can be used to augment any automated analysis. It is this intuitive understanding and human insight that we hope to bring into mixed-initiative planning.

We will first discuss the related work for this problem, followed by relevant concepts of graph theory which will be used in this chapter. This is followed by the discussion of the user interface for plan trace visualization, and interacting with the automated planner. The work in this chapter was previously communicated in [39].

6.2 Related Work

Existing plan trace and domain visualizations present information in a sequential manner with no notion of the rest of the domain. Examples of such representations are Conductor [15], MAPGEN [5], SPIFe [26], Fresco [19] and Webplanner[71]. These can be seen in Figure 6.1. Fresco [19] tries to remedy the issue of single-plan only/linear

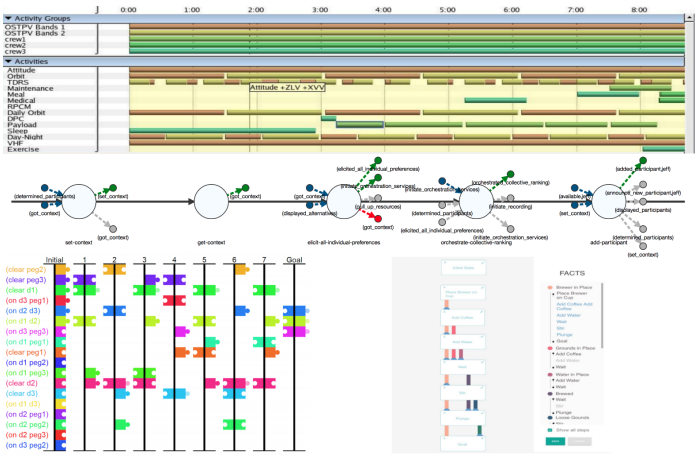

Figure 6.1: Existing Visualization For Planners. Clockwise: SPIFe, Fresco, Conductor, WEBPLANNER

representations by displaying top-k plans. However, it is still left to the user to painstakingly parse the actions of each plan trace of the top-k plans to determine the differences and alternatives. In comparison, our plan trace visualization allows the user to visually parse differences and then go into details as needed. We will now go into some details of our approach to highlight the benefits of our approach.

6.3 Background

We assume the reader is familiar with STRIPS style planning, for more details on it, we refer the reader to [36]. This section will focus on a few graph theory concepts utilized for TGE-viz.

In graph theory, Hopcount is one of the metrics to measure distance between nodes. Hopcount between two nodes is number of hops or links on the shortest path between the two nodes.

$$H_{A \to B} = \min_{K}(P_{A \to B}(k)) \tag{6.1}$$

where $H_{A \to B}$ is the hopcount between A and B, and $P_{A \to B}(k)$ is the number of

intermediate nodes on the k^{th} path from A to B. This concept comes from networks research, where one considers the number of devices a message must pass through. The shortest path may have a larger hop-count than the minimum possible hop-count between two nodes; this can happen if edge weights are different. For our case with deterministic transitions, all edge weights are the same (equals 1), and so hop-count corresponds to shortest-path cost. If one uses probabilistic transitions with probabilities as edge-weights, this would change.

Another useful measure for our problem is "Closeness" of a node n_i; it is the average hopcount from a node (n_i) to all other nodes. Typically the reciprocal of the total hopcount from n_i is used [49] as the closeness measure (as in Equation 6.2).

$$C_{n_i} = \frac{1}{\Sigma_{n_j \in N/\{n_i\}} H_{n_i \to n_j}} \tag{6.2}$$

Closeness indicates how tightly coupled a node is with other nodes. So the average closeness C_g over all nodes in the graph is used as a measure of the closeness of the overall graph. The radius is another helpful metric. First we define the eccentricity of a node as the longest hopcount to any other node. The radius of a graph is the minimum node eccentricity over all nodes. We will shortly discuss the relationship of these graph metrics to the quality of TGE-viz embeddings after first defining the problem of embedding a factored space.

6.4 Problem Formulation

The objective of TGE-viz is that given a domain specification, find embeddings for all grounded propositions and grounded actions such that the nearest neighbors of the graph over grounded actions and propositions are preserved. This optimization measure is used to capture the structure and relationships of the domain. Henceforth, we shall refer to the undirected graph with the actions and propositions of the domain

as just the transition graph. Actions are connected to propositions if the propositions are in the preconditions or effects of the actions. For this work, we only use positive propositions to minimize the information load on the human operator.

The input to the problem is the set of all grounded actions G_a. Each g_a is the triple (P, a, E), where P is the set of precondition propositions (each of which is a string), a is a string that represents the grounded action, and E is the set of effects. All the nodes in the domain are represented by $\tau = V \bigcup A$, where A is the set of all action strings from G_a, and V is the set of all propositions in the domain.

The graph is built by adding edges between every action $a \in A$ and proposition $v \in V$ that is a precondition or effect (includes the delete list) in G_a. The output is the set of tuples (t_i, e_i) where t_i is the i^{th} term of τ, and e_i is its embedding. The embeddings are optimized such that neighboring nodes in the graph are closer together, and non-neighbors are further away. Having defined the problem, we will now look at an approach to embedding the domain graph using force-directed graph embeddings.

6.5 Embedding a Planning Domain

Graph embedding is a rich field with many algorithms that have their relative merits. For this work we experimented with Multi-Dimensional Scaling (metric and non-metric MDS), Locally Linear Embedding, Isomaps, and Spectral clustering. The aforementioned algorithms were run with the implementations in Sci-kit python library [80]. We found that what worked best was a *variation* of Force-based graph embedding [37], which is one of the original techniques of graph embedding, and was intended to provide helpful visualizations of graphs (often called a graph *drawing* algorithm). There are many variants within this class of graph drawing algorithms as compared in [96]. We chose to code a *variant* of the Fruchterman-Reingold algorithm

for its simplicity, speed and scalability for large transition graphs.

6.5.1 TGE-viz Graph Embedding Algorithm

The high-level algorithm for TGE-viz is in Figure 6.2, which builds the transition graph and then updates the embeddings. The crux of the algorithm is in how the

GRAPHEMBEDDER($A, \tau, iterations = 1500$)

1 G $\leftarrow$ BuildGraph(A)
2 \\ Initialize the embeddings between 0,100 in 2d
3 E $\leftarrow$ InitEmbeddings(τ,0,100,2)
4 For $i \in range(0, iterations)$ do
5 E $\leftarrow$ UpdateEmbeddings(E,G,τ, α=1.0)
6 *return* E

Figure 6.2: High-level algorithm for embedding a transition graph

embeddings are updated from their initial random positions. This is in Figure 6.4.

Each node is pulled towards it's neighbors with a force proportional to the distance between them. Each node is also repelled from non-neighbors R by a force inversely proportional to the distance between them. For each iteration, we randomly select *only* $log(|\tau|)$ nodes for repulsion (into the set R) to speed up computation. At each iteration, the node can only move up to a set max-distance α. More even distribution of embeddings can be achieved by repelling from all non-neighbors in every iteration, rather than a random set of size $log(|\tau|)$. However, this will not scale well with very large number of nodes, and it is not necessary to produce helpful visualizations of the domain.

This embedding algorithm works well for a lot of graph configurations, and the

BuildGraph(G_a)

1 G $\leftarrow \emptyset$

2 For $g_a \in G_a$ do

3 \\ each action is comprised of a set of preconditions,

4 \\ the actionID, and set of effects

5 (P,a,E) $\leftarrow g_a$

6 For each $v \in P \bigcup E$ do

7 G = G $\bigcup$ {(a,v)}

Figure 6.3: Building the graph for embedding

degree of usefulness of the visualization depends on the graph measures of "closeness" and "radius" (that was explained in the previously in this chapter). We tested our approach to analyze the embedding performance with the following experiments.

6.6 Experiments and Results

We ran TGE-viz on various configurations of two qualitatively different domains from the automated planning literature, viz. Logistics and Barman domains. In Logistics the objective is to transport boxes from their starting location in cities to their destination using a mix of trucks and airplanes. The standard logistics domain allows airplanes to fly to all airport locations. This allows for very little variability in the underlying graph structure. So, we added constraints to let certain airplanes only access specific airport locations (specific cities), and not all airport locations. This is done with a static property assigned to airplane objects. We added this modification to give the graph more structure and separation between different parts of the domain as evidenced by the low average-closeness score and larger radius measures in Table

UpdateEmbeddings($E, G, \tau, \alpha = 1.0$)

\\ E is the current set of embeddings
\\ create a copy into B (base set)
B = copy(E)
E $\leftarrow \emptyset$
\\ randomly select log($|\tau|$) number of terms for repulsion
\\ more cost-efficient that repelling from all non-neighbors
R $\leftarrow$ RandomSelect($\tau, log(|\tau|)$)
For $w \in \tau$ do
    \\ For loop described in Figure 6.5
return E

Figure 6.4: Process to update the embeddings

6.1.

On the other hand, the Barman domain involves making cocktails using a variety of ingredients and recipes defined in the domain description, using shot-glasses and a shaker to mix ingredients. This domain has an underlying graph that has high average closeness score. This is because there are significantly more actions that connect a proposition to another, and each proposition is connected to a much larger ratio of the total set of propositions by very short hopcounts. As a result, the closeness is higher and radius of the graph is much smaller as seen in Table 6.1.

Barman serves to highlight the kind of domains where the embeddings using TGE-viz will be less separated. As we will see in the results, even in the Barman domain, we can see the structure in the domain and glean information.

Recall that we modified the domains to see how the embedding performs with

1 attrForce $\leftarrow \vec{0}$

2 repelForce $\leftarrow \vec{0}$

3 $\vec{w} \leftarrow$ GetCurrentEmbedding(w,B)

4 For each $n \in$ neighbors(G,w) do

5 $\quad \vec{n} \leftarrow$ GetCurrentEmbedding(n,B)

6 $\quad$ attrForce = attrForce + $(\vec{n} - \vec{w})$

7 For each $r \in$ R do

8 $\quad \vec{r} \leftarrow$ GetCurrentEmbedding(r,B)

9 $\quad \backslash\backslash$ repulsion is inv proportional to distance

10 $\quad$ repulsion = $\frac{|\tau|}{log(|\tau|)} * \frac{1}{\vec{r}-\vec{w}}$

11 $\quad$ repelForce = repelForce + repulsion

12 $\vec{m} \leftarrow$ attrForce - repelForce

13 $\vec{d} = \min((\vec{m}, \vec{m}/|\vec{m}|) * \alpha) \backslash\backslash$ limit by learning rate

14 $E \leftarrow E \bigcup \{(w, \vec{w} + \vec{d})\} \backslash\backslash$ store updated embedding

Figure 6.5: Loop process for Procedure in Figure 6.4

different domain structures. For Logistics domain we increased the connectivity between cities by adding airplanes that connect them. In Barman, we increased the number of cocktails and shot-glasses (cups).

6.6.1 Results and Analysis for Graph Embeddings

The embeddings for various configurations of Logistics and Barman are presented in Figures 6.6 and 6.7, their graph measures in Table 6.1

For the Logistics embeddings in Figure6.6, $C1, C2, ...C7$ encirle the propositions

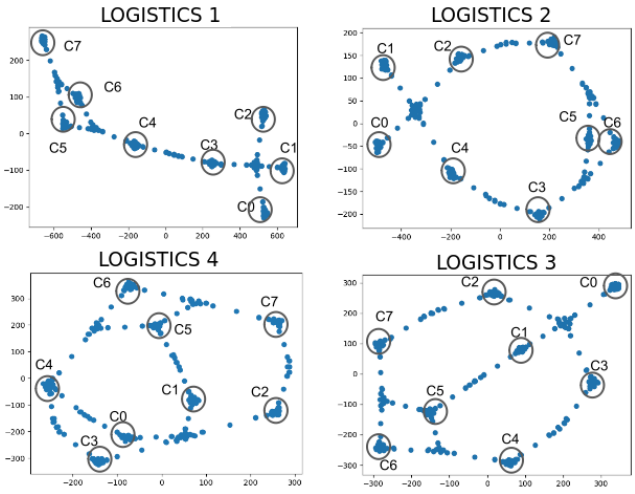

Figure 6.6: Logistics Domain Embeddings

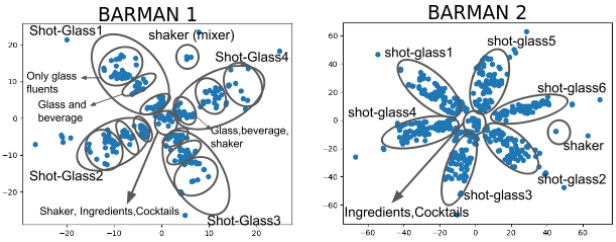

Figure 6.7: Barman Domain Embeddings

Table 6.1: Graph Metrics

Name	**Average Closeness**	**Radius**
Logistics 1	0.067	18
Logistics 2	0.079	20
Logistics 3	0.086	16
Logistics 4	0.090	16
Barman 1	0.429	2
Barman 2	0.41	2

and actions related to transportation within each of those eponymous cities. The embeddings between cities represent the transportation of a package between them. As we add more airplanes and possible actions to transport packages between cities, the graph embeddings update to reflect this in their structure. The clean separation and structure is because the Logistics graphs have a low closeness score (think degree of connectivity), and the radii are larger. This will result in the TGE-viz algorithm spreading the propositions and actions wider in the embedding space. This makes it easier to extract information from the visualization.

On the other hand, Barman domain has a very high closeness score and small radius. So the embeddings are not as spread out as in Logistics and thus makes it less clean (separated) for visualization and ascribing meaning to clusters and paths in the embedding space. Even in the densely connected Barman domain we can clearly see separation in the embeddings as highlighted in figure 6.7. The propositions and actions related to each shot-glass (cup) form protruding prongs from the center. The propositions and actions related to mixing ingredients in the shaker and making cocktails are in the center. Very weakly connected nodes that have few edges like $clean\_shot1$ and $clean\_shaker$ are pushed to the periphery.

Hence, one way to determine how well a domain's transition graph can be visualized with TGE-viz is to use the closeness and radius scores of the graph. Lower closeness and higher radius scores are better. Using such graph embeddings, we now present the first version of our user interface that lets a user visualize plan traces with respect to the entire domain structure, and interact with the planner through it.

6.6.2 Mixed Initiative User Interface With TGE-viz

After computing the 2-d embeddings, we will now discuss how one can display them in an interactive interface to allow a human operator to be involved in the

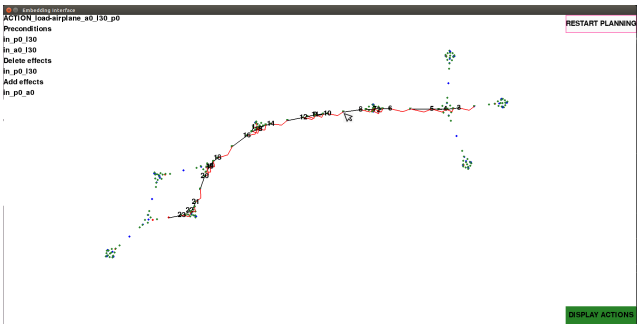

Figure 6.8: Plan Trace in Modified Logistics With Tge-Viz for the Goal of Delivering the Package to City 6 Location 3.

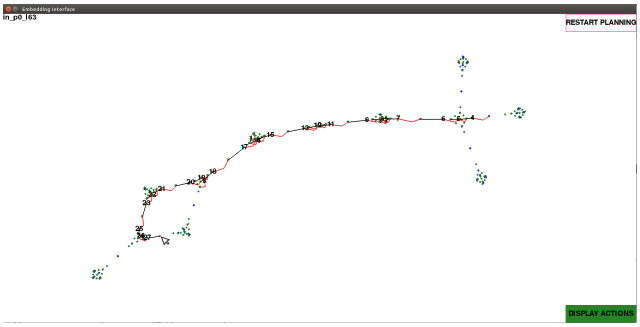

Figure 6.9: Alternate Plan Trace in Modified Logistics With Tge-Viz for the Goal of Delivering the Package to City 6 Location 3.

planning process. We will also discuss the features added to help ease the cognitive load of the user.

Given a set of embeddings for propositions and grounded actions, we developed a proof-of-concept user interface in PyGame [92] to represent/visualize those embeddings and co-plan with an automated planner. Figure 6.8 is a screenshot capture of our interface. The interface first displays all the propositions and actions in the domain. The actions are green, the current state propositions are red, and other propositions are displayed in blue. The background can be switched between white and black depending on the user's preferences. The initial state is fed in through a *problem.pddl* file in the standard pddl format [46].

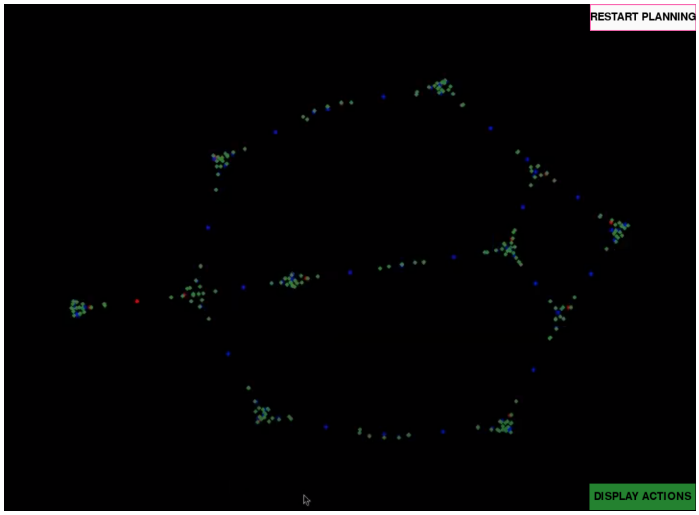

Figure 6.10: Interface For Collaborative Planning.

The two dimensional embeddings are displayed in an interface with actions and propositions are colored with two different colors to help the human parse the points as in Figure 6.10. Our interface allows the user to turn off the display of actions, and only see the propositions, as in Figure 6.11. Additionally, the current state propositions are displayed in a different color (red) to help with human parsing the visual information. Our interface also allows zooming in and out of certain regions, as well as turning off actions in the display. Such display of information is coherent with the guidelines from EID for Knowledge Based Behaviors (KBB); for KBB the recommendation is that (paraphrased) "*represent the work domain in the form of an abstraction hierarchy to serve as an externalized mental model that will support knowledge-based problem solving*" as stated in [101]. The abstractions support we have is through filtering actions and propositions, as well as through the zoom functionality. Additionally, when the user hovers their mouse over a node in the display, the pertinent information appears on the top-left as in Figure 6.12; this could be the action information such as precondition and effects, or the proposition name.

Thus far we have spoken of how humans can glean information, but we also need to

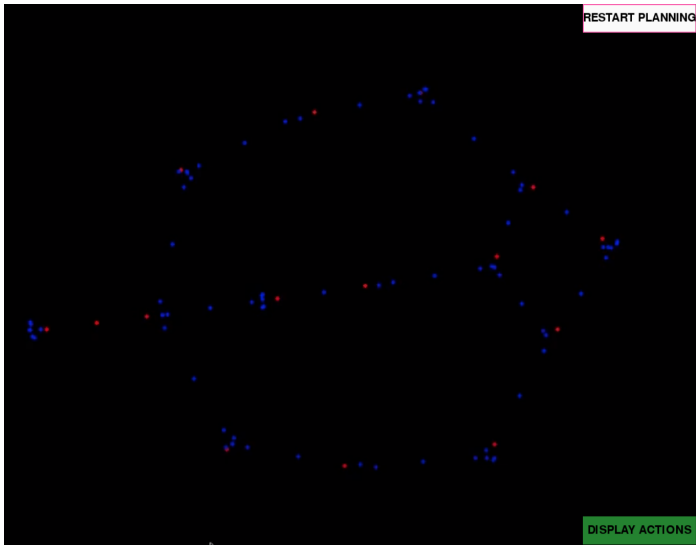

Figure 6.11: Filtering the Display to Only Propositions; Current State Propositions in Red, and All Others in Blue

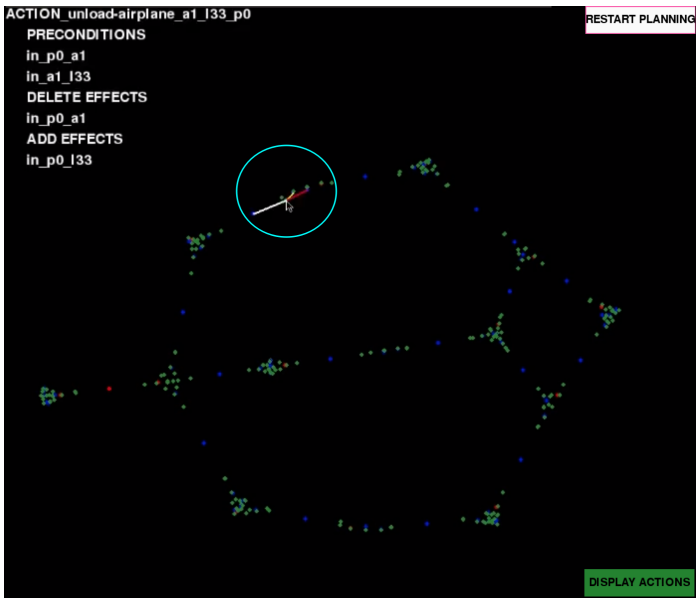

Figure 6.12: Display of Action Information When an Action Node Is Highlighted

consider how they can collaborate in the planning. Our approach allows the human to set subgoals and ask a back-end planner –fast downward [48] for our implementation– to generate a plan to that sub-goal as in Figure 6.13. If that plan is satisfactory, then successive subgoals can be assigned and the plan is extended from the previous state as in Figure 6.14. If a sub-plan is not satisfactory, then the user can reset the planning to the initial state using the “Reset Planning" button. Adding support for undoing and redoing are additional improvements that we left out our proof-of-concept. The plan is displayed with action numbers on the visual interface, and correspond to the action order in the text/terminal output of the same plan; this is as returned by the backend planner which we also display to the user

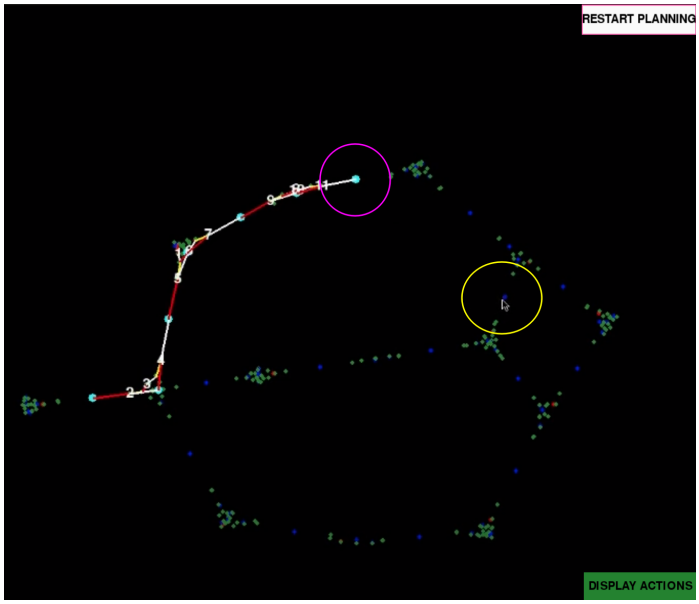

Figure 6.13: Subplan Generated by Planner After User Clicks a Subgoal

6.7 Summary and Extensions

One limitation of the current approach is that we only embed the propositions and not their negations. This is helpful in limiting the information displayed to the

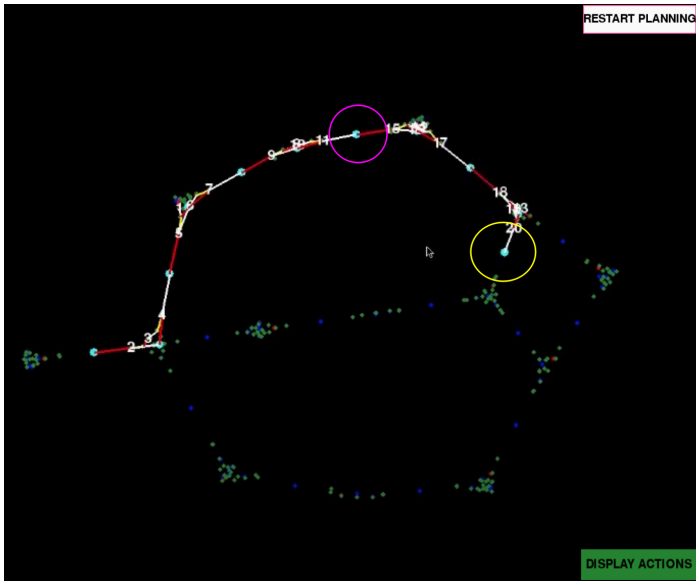

Figure 6.14: Extending the Plan to the Next Goal From the Plan in Figure 6.13

human. If each proposition corresponds to a value in a multi-valued fluent, then only embedding positive propositions maybe sufficient and cover the state space. For example, "package1 in location 3" or "package1 in location 5" are both values associated to the fluent "package1 location". Adding "not package1 in location 3" when it is in location 5 is redundant. However, this may not always be sufficient. For some domains we may have to embed the negations of propositions too depending on the variables and actions definitions in the domain. This can be incorporated into the existing approach by adding the negations to the embedding set.

With respect to the interface, there are some extensions that could help. One could display time and other costs associated with the actions in the plan traces displayed on the embeddings. The user should be able to toggle these on and off, to minimize information load.

In summary, our TGE-viz approach allows visualization of plan traces in the context of the transition graph of domain, embedded in 2d space using force-directed embeddings. This enables the human in mixed-initiative planning to intuitively and

quickly analyze plan traces overlayed on it. Alternate plans can be formulated easier, and our interface provides an easy way to interact with the automated planner. The primary human limitation we tackle here is memory; it would be unreasonable to expect the human to remember all domain information and think of all possible alternatives for actions easily. We also allow the human to limit the information presented, to help with information load. We think that interfaces such as ours which reduce the information and cognitive load of the user will help the adoption of automated planners.

For acting and planning with humans thus far, our problems have considered an individual human in the loop. In the next chapter, we tackle a problem involving a team of humans, and how task switching costs can limit team performance in completing a set of tasks.

Chapter 7

TEAM TASK ASSIGNMENT

In this chapter we focus on human teams and the problem of assigning tasks linked by sequencing constraints (partial-ordering) for completing a common set of goals. Team members maybe required to switch between tasks to satisfy a deadline (makespan). Existing scheduling approaches either ignore or treat the cost of task switching improperly as we will discuss. For people, task switching is a non-trivial performance affector, and this cost has been well documented in the human factors literature [74][109],[72]. In this chapter we will explicitly consider the time cost of task switching when assigning tasks for a team of humans. We also develop an approach that allows one (such as a team manager) to limit the amount of task-switching to satisfy deadlines associated to goals. Note that in this chapter, we use the word task to mean an action in a plan for completing a set of goals. This work involves the human both during planning and execution. In planning, the team members provides their estimates for the time taken for each task, and their availability. In execution, it is the humans that actually execute the tasks, and we consider their task-switching costs during execution.

7.1 Motivation

Organizations need to manage human teams to accomplish their goals by planning to complete a set of tasks within time and resource constraints. As an example, in the Agile methodology for software development –which was our initial motivating setting– in which tasks are assigned to team members typically every two weeks [2]. In most cases, there are dependencies among these tasks, as well as priorities. Other

key characteristics of these type of planning&scheduling (P&S) tasks are: (1) uncertainty in the task duration; (2) availability of team members (affected by holidays, time-off); (3) task-specific qualifications (e.g., only a developer trained on Javascript programming can develop the front-end interface); and (4) there may be limited or no central-control during the execution, as people's tasks are assigned at the start of the planned period, and then control is local i.e. each person decides what to work on. In relation to this last characteristic, if a team member can work on a blocking task that is a prerequisite for other tasks, then they might preempt their current task to work on it. Once that is done, then they can resume working on the preempted task. In this way, that team member would unblock others who might be idly waiting because they are not qualified to do it.

The objective of a P&S solution in our setting is to compute a task-to-human assignment that considers the following: tasks dependencies, goal prioritization, makespan, resource constraints (agent availability and qualifications), uncertain tasks duration, distributed control, and preemption capabilities of agents. Note that in this paper, tasks are low-level activities like actions in a plan. A good solution to the type of P&S problems herein, would increase the likelihood of task completion within the planning period.

The contributions of our approach include the formalization of a real-world P&S problem, and the adaptation of two search algorithms to solve it, viz. Tabu search and Monte Carlo Tree Search (MCTS). We also evaluated these approaches on diverse and randomly generated problem instances. Our experimental results show the ability of the algorithms to effectively compute task assignments and their priorities that leverage preemption to improve outcomes in this problem setting.

7.2 Related Work

In SDM, an approach to execution control under uncertainty is Dynamic Controlability (DC) [75] for Simple Temporal Networks with Uncertainty(STNU) [103; 89]. However, such DC approaches require a central controller for dispatching, and continuous monitoring of the execution state which is not available in our problem setting. Distributed Multi-agent STNU (MaSTNU [17]) sought to address the problem of distributed control when communication with a central agent is absent or intermittent. It computes multiple STNUs, one for each agent.

However, without enabling preemption of tasks (context switching), these approaches cannot satisfy the makespans in some problems. Figure 7.1 shows an example, where there is no solution even without uncertainty (contingent links). In the example, we use a disjunctive condition using the representation used in [25] which is there to indicate an agent can only do one task at a time and no-preemption. We also have an external edge that enforces synchronization between agents (using the representation from [17]). In this example, there is no assignment to D2 or E2 that allows satisfying all constraints. However, if we allow preemption, it lets us ignore the disjunctive constraint. Then E2 can be started at t=0, preempted at t=6 to finish D2, and then return to E2. This would satisfy all makespans (displayed in red).

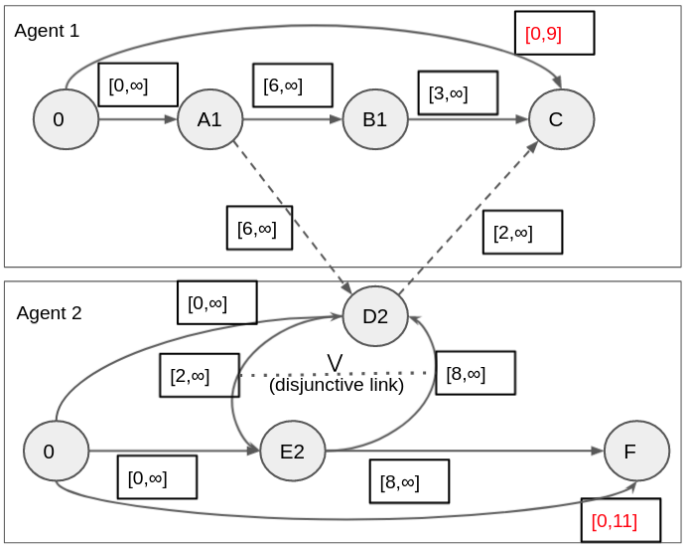

Figure 7.1: Example of a Multi-Agent STN That Cannot Be Solved Without Preemption.

Another framework is probabilistic STN (PSTN), where the uncertain durations follow a probability distribution, which is an assumption we make in our work as well. There is recent work on computing a dynamic control schedule for PSTNs [38], and another one for using Monte Carlo Tree Search [88]. These methods still require a central controller.

From operations research (OR) literature, our problem maps to Resource Constrained Project Scheduling Problems (RCPSP). There are different dimensions to RCPSP problems. In a recent survey on RCPSP problems [44], the dimensions were categorized into (1)resource type, (2) activity concepts, (3)objective function, (4) availability of information. With respect to these dimensions, our problem is defined as (1)renewable resources (human agents)(2) preemptable activities (3) time-based objectives (4) Stochastic durations. The RCPSP dimensions and where our problem is situated is displayed in Figure 7.2. We have not found any RCPSP work that considers both stochastic durations and *preemption* with a makespan objective. The

approaches we found in RCPSP literature that consider preemption have deterministic durations. These approaches either have no penalty for interruption [76] or impose a hard limitation on the number of interruptions [113] to account for preemption cost without explicitly considering it in the constraint satisfaction problem.

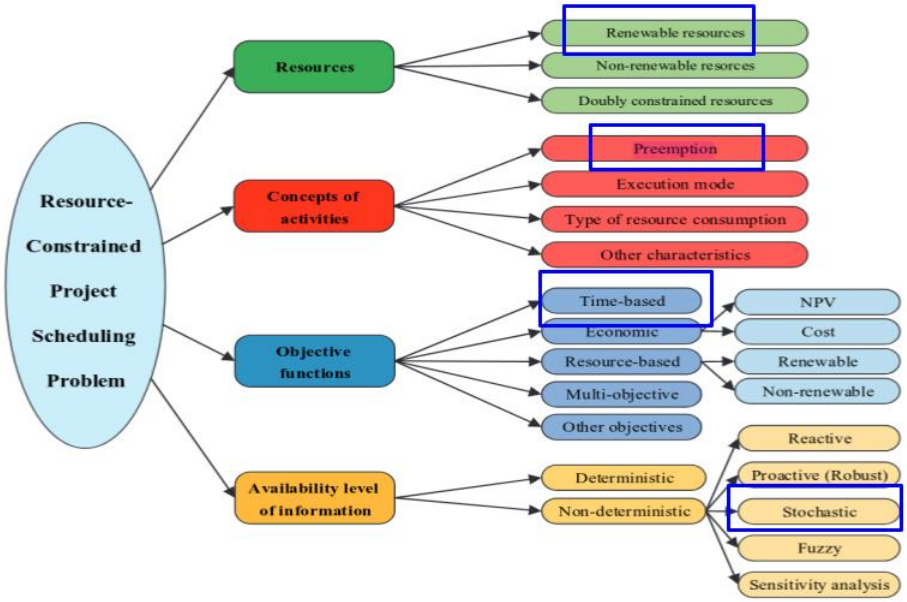

Figure 7.2: Categories of Rcpsp Problem From Operations Research Literature From [44]

The last branch of relevant literature is from Operating Systems (OS), specifically from Real Time Operating Systems (RTOS) [95]. In such systems, process priorities are used to interrupt/preempt the current activity on a processing unit, and switch control. Priorities become very useful when most of the tasks have uncertain durations, or are affected by exogenous events (interrupts), and the execution is distributed across different processes. The closest analogy to our problem we found in RTOS literature is Worst Case Execution Time (WCET)[90]. WCET is especially used to certify safety critical systems like automotive controllers; the RTOS response

to various test-loads is evaluated to make sure the processing time for critical tasks is within a safety cutoff. This is different from our problem in that we do not tune our approach to satisfy a fixed makespan for certain tasks. Rather, ours is a general purpose (assignment) algorithm to handle variable makespans, and constraints over agent availability, and qualifications.

7.3 Problem Formulation

Our problem of Assignment and Prioritization of Tasks (APT) for Distributed Execution is given by the tuple $< \pi, T, A, P, G, \gamma, w, m, \delta, q, a, \beta >$ where:

- π: Partial Order Plan represented by a directed acyclic graph (DAG $= < T, E >$) where T is the set of vertices of the DAG, and correspond to the tasks in the plan, and the directed edges capture the ordering dependencies.
- T: set of tasks
- A: set of agents.
- P: fixed set of priority levels assignable to the actions. P $\subset \mathbb{Z}^+$
- G: set of goals.
- $\gamma : G \rightarrow 2^T$: tasks associated to each goal. A goal is considered complete when it's associated tasks are done.
- $w : G \rightarrow \mathbb{R}^+$: importance weight (priority) of each goal. Different from assignable task priority.
- $m \in \mathbb{R}^+$ is the maximum makespan within which the goals ought to be completed.

- $\delta : T \times \mathbb{R}^+ \rightarrow [0,1]$ defines the probability of completing a task in a given amount of time.

- $q : A \times T \rightarrow \{0,1\}$: qualification function; $q(a_i, t_j) = 1$ if agent a_i can perform task t_j.

- $a : A \times \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \{0,1\}$: availability function; $a(a_i, t_1, t_2) = 1$ if a_i is available between t_1 and t_2.

- $\beta : A \times T \rightarrow \mathbb{R}^+$ is the preemption cost function that tells how much additional time a task will take because it has been preempted.

A solution (output) to the APT problem consists of finding an assignment function $\alpha : T \rightarrow A$ and prioritization function $\rho : T \rightarrow P$ that fulfill the temporal and resource constraints (given by π, q and a). In order to compare different solutions, we define a scoring function for goal completion. The score of a plan is the weighted probability of goal completion within the maximum makespan (m). This is defined in Equation 7.1.

$$\sum_{g \in G} \frac{w(g)}{\sum_{g' \in G} w(g')} \times prob(c(g, \alpha, \rho) < m) \tag{7.1}$$

$prob(.)$ returns the probability, and $c(.)$ is a stochastic function that returns the completion time of goal g for a given assignment and prioritization. $c(.)$ is determined using a simulator of the execution process that we will describe next in the following section.

7.4 Solving APT Problems

We use a search-based approach to solving APT. To estimate the goodness of solutions –given the stochasticity of task durations– we developed a simulator to sample and rollout execution trajectories for a given assignment and prioritization

of tasks. In the following sections we will describe the simulator and the search algorithms that use it to solve APT.

7.4.1 Simulator

The simulator takes in the problem components and a potential solution (given by α and ρ) and returns a sampled value of the $c(.)$ function. The simulator samples durations for each task, and simulates execution. We use a discrete-time simulator. A continuous-time simulator would be faster, but a discrete simulator implicitly considers the slack from human behavior like reading emails. Thus, it better fits the problem setting we are interested in. At each step, the simulator determines which task an agent works on based on their backlog; backlog is the set of tasks ready and incomplete at that step. Tasks assigned to an agent as per $\alpha(.)$ will appear on their backlog when task dependencies are completed. The simulator includes certain key dynamics:

- Completion time: when a task is assigned to an agent's backlog, it comes with its priority level (given by ρ) and the time required to complete the task (sampled using δ).

- FIFO: when two tasks arrive to an agent's backlog with equal task priority (set by ρ), the agent works on the first task that arrived. If they arrive at the same time, then the order is arbitrary.

- Agent availability: when an agent is working on a task, progress is made on the task only during the time when the agent is available (given by $a(.)$). When the agent is available, the time required to complete the task is decremented by one time-step of the simulator.

- Preemption: when a higher-priority task enters the backlog, it will preempt the current task, which will return to the backlog. The time needed for completion of the preempted task will increase (as per β).

The simulator returns the completion time of each goal in G. Completion times are used by the search algorithms to find good assignments and prioritizations of tasks. In this work, we consider all tasks as preemptible. Non-preemptible tasks can be easily supported by checks in the simulator. The rest of the methodology remains unchanged.

7.4.2 Search Algorithms

We present two stochastic search approaches to solve APT problems: Tabu search and MCTS. Both use simulations to evaluate the value of a solution using Equation 7.1. In Tabu search, we start with a random initial solution, and take local improvement steps until no more improvements greater than a specified threshold are possible. We iterate the process (random restart) for as many times as possible within a time limit. The tabu list in tabu search only keeps the solutions obtained at the end of each iteration.

For MCTS, the nodes at each level in the tree correspond to a partial assignment of tasks with priorities (a partial solution). Each successor node adds an assignment and a priority for a task. During MCTS search, the process steps through each node and recursively selects the best-child node based on the average reward of the node, (initialized to 0); this is only *if* the node is already fully expanded. If the node is not yet fully expanded, i.e. only a subset of children have been evaluated, then a random node from the remaining children is selected to be opened. Each node is evaluated by random rollout, i.e. random assignments and prioritizations for the remaining tasks (subject to the qualification constraints). The reward at the leaf node (end of

a rollout) is the average score returned by 30 simulations of the complete solution obtained at the leaf node. This reward is backpropagated to each of the parent nodes, and the average reward is updated. When the allowed time for search has expired, MCTS may *not* have opened a terminal node of the search tree (which would have all tasks assigned). In that case, we take the best leaf-node from amongst the nodes opened so far; this would only have a partial assignment of tasks. We then complete the assignment using a hill-climbing search for the remainder with no random restarts or tabu list. This is needed because unlike MCTS for the games of Chess or Go, we need a complete solution and not just the next decision in the search tree. In our experiments, we did *not* utilize exploration with upper confidence bounds [58]. We found that exploration resulted in worse solutions when limited by the cutoff time; we attribute this to MCTS taking more time to explore than exploit/discern between good partial solutions that were found early.

One of the algorithmic decisions that helped improve results in MCTS was to order the task assignment decisions intelligently. We run a topological sort on the partial-order plans DAG and use the topology levels to order tasks in the search tree. The rationale is that decisions made for tasks (nodes) at lower topological levels would impact downstream decisions, so it makes more sense to decide these first. Additionally, since goals have importance weights associated to them, we further order task assignments in MCTS based on the highest weight of the goals that require them. Lastly, our MCTS code was built upon the standard implementation of the algorithm [27] as implemented in the python `mcts` library.

7.5 Experiments

We evaluate the Tabu and MCTS algorithms on 6 randomly generated APT problems. We give each algorithm a time limit of 30 minutes. The number of tasks in

				Goals						
H	a	p	m	g_0	g_1	g_2	g_3	g_4	MCTS	Tabu
3	4	1	163	1	1	2	1	1	0.86	0.83
3	4	3	163	1	1	2	1	1	0.99	0.93
3	6	1	163	2	2	1	3	3	0.99	1.0
3	6	3	163	2	2	1	3	3	1.0	1.0
3	4	1	156	2	3	1			0.04	0.0
3	4	3	156	2	3	1			0.48	0.4
3	6	1	156	1	2	1	2	2	1.0	1.0
3	6	3	156	1	2	1	2	2	1.0	1.0
3	4	1	165	3	3	3	1	3	1.0	1.0
3	4	3	165	3	3	3	1	3	0.93	0.99
3	6	1	165	3	3	2			1.0	1.0
3	6	3	165	3	3	2			1.0	1.0

Table 7.1: Likelihood of Finishing Execution by Maximum Makespan for McTs and Tabu Search Given Different Configurations of Topological Depth (H), Number of Agents ($A = |A|$), Priority Levels ($P = |P|$)) And Maximum Makespan (M).

each plan is fixed to 30. So, a plan DAG with fewer levels/depth will have more tasks per topology level resulting in a wider graph. The number of goals was randomly set between [3, 5], and each goal weight was randomly set between [1,3]. Each task from the last topological level was randomly assigned to one of the goals. The simulation time-step was 1 hour. The following problem features were randomly generated:

Task Time: The time required per task is stochastic, and follows a uniform distribution between two limits. The limits are sampled from a normal distribution

				Goals						
H	a	p	m	g_0	g_1	g_2	g_3	g_4	MCTS	Tabu
6	4	1	169	2	1	3			0.32	0.33
6	4	3	169	2	1	3			0.34	0.33
6	6	1	169	1	3	3			1.0	1.0
6	6	3	169	1	3	3			1.0	1.0
6	4	1	174	2	1	1			1.0	1.0
6	4	3	174	2	1	1			1.0	0.99
6	6	1	174	3	1	2			1.0	1.0
6	6	3	174	3	1	2			1.0	1.0
6	4	1	154	1	2	3	3		0.51	0.59
6	4	3	154	1	2	3	3		0.7	0.76
6	6	1	154	3	2	3	3		1.0	0.84
6	6	3	154	3	2	3	3		1.0	1.0

Table 7.2: Second Set of Results For Likelihood of Finishing Execution by Maximum Makespan for McTs and Tabu Search Given Different Configurations of Topological Depth (H), Number of Agents ($A = |A|$), Priority Levels ($P = |P|$)) And Maximum Makespan (M).

with a mean of 8 (hours), and standard deviation of 3; the smaller sampled value is the lower bound.

Agent Availability: Each agent is available during random intervals of time to make the search more challenging. We start with the agent being available for the max duration of a problem. Then, for each hour, the likelihood of an agent taking time off starting from any given hour is 0.05. If an agent takes time off, the duration

is sampled from a Gaussian distribution with mean 8 and standard deviation 4.

Likelihood of Task Dependency: Each task has a 5% chance of being connected to any other task. This is on-top of the single edge needed to enforce the depth of the DAG underlying the partial plan.

Qualifications: Each task requires a qualification to perform it. Each agent is assigned a subset of the possible qualifications, with at least one agent having each qualification. The likelihood of an agent having an additional qualification is 0.25. We fixed the number of qualifications to 4.

The makespan m was set to 60% of the sum of all the tasks' duration upper bound. We set this value empirically based on results from using a team of 3 agents and 1 priority level; the score using Equation 7.1 was often below 50% with either search algorithm, so we chose it as a challenging makespan. Given these variations in problem parameters, we posit that our problem generation is sufficiently parameterized to produce diverse, and challenging problems.

During simulation of an assignment on a plan, when an agent's task is preempted by a higher priority task, we set the penalty to a fixed amount; 0.5 hours additional time to complete the preempted task. For our experiments, we vary the following: topology of the underlying DAG - we set the depth to 3 or 6, which affects the longest sequence of dependencies; number of agents - we used teams of 4 and 6 people; priority levels - we used 1 and 3 priority levels to evaluate the effects of preemption. All code was written in python and experiments were run on a PC with Intel® Core™ i7-6700 CPU, running at 3.40GHz on Ubuntu 20.04 with 32 GB of memory. All random elements are controlled by a seed.

7.6 Results

In Table 7.1 and 7.2, we present the results on 6 randomly generated problems; each problem's data is separated into a sub-table. The score under the MCTS and Tabu columns is computed as per Equation 7.1, and is the averaged result of 100 simulation runs. For 4 agents, when the number of priorities increases from 1 to 3, the success rate increases appreciably for both algorithms, except in one anomalous case which we attribute to the stochastic nature of the search. Having more priority levels alleviates the agent resource constraint. An example of this is when only one agent has a necessary qualification for many tasks, that agent becomes the bottleneck. By allowing preemption, that agent can switch tasks and improve outcomes. For example, in the second graph we see an increase in the score from 4 agents with 1 priority level, to 4 agents and 3 priority levels. The reason for that is only one agent had the qualification required for 8 longer tasks. When the same partial plan was run with 6 agents, there were two more agents who had that qualification, and that helped the success rate jump to 1.0. Both MCTS and Tabu search performed comparably well, and so we cannot say one is better. It is unsurprising that with 6 agents (even with just 1 priority) both methods tend to find seemingly optimal score solutions, except for the last graph when Tabu search only finds a 1.0 score solution with 3 priority levels and not with 1 priority level.

7.7 Summary

In this work, we presented a new type of P&S problem (APT) that resembles a set of real world problems. The main differences with prior work is the combination of distributed control during execution and the agent's ability to preempt tasks. We presented the adaptation of two search algorithms, MCTS and Tabu search, to solve

these problems. The experimental results show that both algorithms provide promising results for APT problems. The results also show that when agents are able to preempt their current tasks, the goal completion score improves appreciably.

Chapter 8

CONCLUSION

In this final chapter, we will summarize the work done herein, followed by recommendations on how to incorporate human limitations into other SDM settings based on what was learned during the research. We end with open problems for future research that require incorporating human limitations.

8.1 Summary of Research

We looked at the problem of incorporating human cognitive limitations when computing solutions for a set of sequential decision making problems. The primary thrust of this work was that using problem-specific human models (Chapters 4, 5) and task-proxies (Chapters 3, 7), is a sensible approach when computing solutions for humans. We studied how cognitive limitations –such as limited attention, and information load– could be factored into the problem constraints and assumptions, so as to improve the quality of interaction with a human-in-the-loop. By using task-specific proxies, assumptions, and behavioral models, we avoid the pitfalls of using a general cognitive model or assumptions on human intelligence –like bounded-rationality or finite-lookahead for inference– that we discussed in Chapter 2. We also studied good design principles for human-planner interactions and built an interface based on those principles.

The specific problems we looked at included the following: computing policies for humans when acting in a setting that can be modeled as an MDP; computing robot path trajectories when humans and robots are moving in the same space; computing task assignments and their priorities for distributed execution in human teams; and

how to design an interface for human and automated-planner interaction.

With respect to computing policies for humans, we looked at two settings. In the first setting, we considered how to tradeoff policy-complexity with policy-value to find policies that are easier to execute. We defined an easier policy as one that had the same actions across similar states. Similarity of states was input, and could be determined by how likely a human was to confuse two states, or based on the number of feature similarities. We computed policies using a modified policy iteration method that used a weighted sum of policy-value and complexity where both were scaled to be in the range of $[0, 1]$.

In the the second setting for computing human policies, we considered the human's response to uncertainty, and how by modeling the costs and dynamics of that response, one could come up with better policies for humans to execute. We presented two methods to compute policies for this setting: a faster, heuristic-method using policy iteration that can handle larger state spaces; and a method to compute the optimal policy for the problem setting using a branch-and-bound method that is suitable for smaller state spaces. In both these settings for computing the human policy, we used a problem specific human model of how human inference and behavior is affected by the domain and policy given to the human. We use statistical models of errors and responses, and incorporate them into the computation process.

With respect to human robot interaction, we consider the problem of determining how to layout paths for an AGV (autonomous guided vehicle). In this problem, we consider that the robot(s) have a specific set of tasks to perform and points of interest that need to remain connected in the final graph comprising of all the paths that the robot can move on. We consider that the human pays limited attention to the robot and cannot invest significant time considering all possible combination of paths that the different robots moving around that person might take. So when

computing the robot paths to layout on the floor, we emphasize predictability of the robot motion from just the current position alone. We do this by minimizing the number of branching vertices in the overall graph. In our literature review we found this to be a novel graph problem, with the closest analogues being the Strongly Connected Steiner Subgraph (SCSS) (which does not consider branching/diffusing vertices), and Directed Steiner Tree with Limited Diffusing vertices (DSTLD) (which has a single source node and doesn't consider path costs). To tackle this problem, we presented a hill-climbing approach to search the space of combination of paths that are within an acceptable path-cost for each task and minimizing the number of branching vertices.

With respect to planning for distributed human teams, we presented the problem of Assignment and Prioritization of Tasks (APT), a novel and real-world inspired problem setting to help human teams complete projects with limited communication and distributed execution (applicable for offshore/multi-national teams or remote work). The requirement of the human team members in this problem is that they will switch to another task when they know it is of a higher priority level. In order to reduce the cognitive and execution cost of task switching, we allow the number of priority levels (and thus maximum number of switches) usable by our search method to be limited by the user. The objective of the policy is to complete all the tasks within the deadline(makespan) with a high probability. This is done using MCTS and Tabu search. Both search approaches performed comparitively well in our experiments involving randomly generated partial-order plans with varying topological depths and dependencies.

The preceding problems we described involved the human directly following a policy, or being an active participant, while at most giving information for the planning/computing process but not actively involved in computing the solution (such as

proposing or correcting parts of the solution). In our work on defining the planning interface for humans and automated planners we also considered cognitive costs for humans as co-planners. With respect to human-planner interfaces, our work [39] focused on developing an interface that reduced the cognitive burden of remembering and holding all the domain information –such as all possible state-feature values or grounded actions– by embedding it in a 2-d representation and displaying it. The display of information and features of the interface was guided by principles from cognitive ergonomics, specifically Ecological Interface Design (EID) principles. Our interface allowed one to manage information load, and allowed the human to specify subgoals and guide the automated planner through different plans , while helping the human think of alternative plans.

8.2 Recommendations for Computing Human Friendly Solutions

Based on the work done herein, we gleaned a few principles that can be useful when developing solutions for human-in-the-loop SDM problems.

A general observation that applies to a lot of problems that require human cognitive effort is that one has to trade-off value with cognitive effort; this effort could be inferential effort, demands on memory, attention costs, and such. Identifying the key cognitive factors (like memory) in each problem is pivotal, and ought to influence the selection of the problem-specific human-model or task-proxy, as well as the methodology.

Using a task-proxy, instead of a human cognitive model, when possible, can make the problem easier to work with –like what was done in Chapters 3 and 7– as it doesn't require parameter tuning/learning on a problem-specific human model. However, the solutions do still need to be validated with human-subject experiments.

With respect to building a human-model of behavior for a specific problem, the

more specific the model is to the problem and expected behavior, the better. More general models that are fit to problems can have a greater risk of incorrectly fitting as was shown in the work by [8].

On the note of problem-specific human models, it can be important (for outcome measures) to consider problem-specific behaviors like extra-sensing actions in Chapter 5, or risk-aversion behavior. These problem-specific assumptions, or normative assumptions, can make an appreciable difference to the quality of outcomes. Considering how the human would behave in different problem situations ought to influence the computational model. For example, it can be helpful to consider under what situations in a problem might cause a human agent to be stressed, and make poorer decisions and have less attention.

Lastly, for running human subject studies in SDM problems, it is especially important (and often difficult) to isolate the particular cognitive facet(s) as much as possible –such as working memory, inferential ability, attention, and such– and ensure that variation in other facets are not influencing the results. For example, if inferential performance is being studied, memory should be eliminated as an issue by clearly displaying or highlighting all the pertinent information. If not carefully controlled, one runs the risk of confounding effects due to the interplay of attention, memory, and inference in human cognitive performance.

8.3 Avenues for Future Research

In addition to the individual problem extensions discussed in each chapter, there are quite a few open problems that can be worth studying.

For computing human policies to execute, one of the issues is the number of features that the human has to pay attention to, which can influence state-inference outcomes. Reducing the features that need to be attended to would ease the attention-

cost to the human and can reduce the errors in policy-execution as well. So exploring this trade-off can be an interesting avenue for research. Another direction for human-policy computation is to determine what sets of options (partial state-action mappings) to teach a human user, such that their outcomes improve across one or more MDPs. This is akin to determining the playbook/strategies to use in a sport; one cannot compute a complete policy or expect the human to remember it, so we compute partial policies and model how the human might put them together when executing. The assumptions would of course have to be tested with human-subject experiments.

In summary, this dissertation highlights the importance and value of incorporating human cognitive limitations when coming up with solutions for human-in-the-loop SDM problems. The fields of cognitive ergonomics, and psychology have a wealth of information on cognitive limitations and good principles for human-friendly design. Some of this knowledge was adapted and applied to the SDM problems presented herein. We use task proxies (constraints or optimization objectives) and problem-specific models of human behavior to translate cognitive limitations into problem components that we can incorporate when computing solutions. We hope this dissertation inspires continued work in this interesting and useful direction.

REFERENCES

[1] Abel, D., W. Dabney, A. Harutyunyan, M. K. Ho, M. Littman, D. Precup and S. Singh, "On the expressivity of markov reward", Advances in Neural Information Processing Systems **34**, 7799–7812 (2021).

[2] Abrahamsson, P., O. Salo, J. Ronkainen and J. Warsta, "Agile software development methods: Review and analysis", arXiv preprint arXiv:1709.08439 (2017).

[3] Abras, C., D. Maloney-Krichmar, J. Preece *et al.*, "User-centered design", Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications **37**, 4, 445–456 (2004).

[4] Acemoglu, D. and P. Restrepo, "Artificial intelligence, automation, and work", in "The economics of artificial intelligence: An agenda", pp. 197–236 (University of Chicago Press, 2018).

[5] Ai-Chang, M., J. Bresina, L. Charest, A. Chase, J.-J. Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. Chafin, W. Dias and P. Maldague, "MAPGEN: Mixed-initiative planning and scheduling for the Mars Exploration Rover mission", IEEE Intelligent Systems **19**, 1, 8–12 (2004).

[6] Althoff, D., D. Wollherr and M. Buss, "Safety assessment of trajectories for navigation in uncertain and dynamic environments", in "2011 IEEE International Conference on Robotics and Automation", pp. 5407–5412 (IEEE, 2011).

[7] AmericanAcademyOfPediatrics, "The apgar score.", Advances in neonatal care: official journal of the National Association of Neonatal Nurses **6**, 4, 220 (2006).

[8] Armstrong, S. and S. Mindermann, "Occam's razor is insufficient to infer the preferences of irrational agents", Advances in Neural Information Processing Systems **31** (2018).

[9] Arnold, T. and M. Scheutz, "The tactile ethics of soft robotics: Designing wisely for human–robot interaction", Soft robotics **4**, 2, 81–87 (2017).

[10] Ashton, K. *et al.*, "That 'internet of things' thing", RFID journal **22**, 7, 97–114 (2009).

[11] Bainbridge, L., "Ironies of automation", in "Analysis, design and evaluation of man–machine systems", pp. 129–135 (Elsevier, 1983).

[12] Boucherie, R. J. and N. M. Van Dijk, *Markov decision processes in practice*, vol. 248 (Springer, 2017).

[13] Broz, F., I. Nourbakhsh and R. Simmons, "Planning for human–robot interaction in socially situated tasks", International Journal of Social Robotics **5**, 2, 193–214 (2013).

[14] Brusco, M. J., S. Stahl *et al.*, *Branch-and-bound applications in combinatorial data analysis*, vol. 2 (Springer, 2005).

[15] Bryce, D., P. Bonasso, K. Adil, S. Bell and D. Kortenkamp, "In-situ domain modeling with fact routes", in "Proceedings of the Workshop on User Interfaces and Scheduling and Planning, UISP", pp. 15–22 (2017).

[16] Cai, H., V. W. Zheng and K. Chang, "A comprehensive survey of graph embedding: problems, techniques and applications", IEEE Transactions on Knowledge and Data Engineering (2018).

[17] Casanova, G., C. Pralet, C. Lesire and T. Vidal, "Solving dynamic controllability problem of multi-agent plans with uncertainty using mixed integer linear programming.", (2016).

[18] Casey, B. M., D. D. McIntire and K. J. Leveno, "The continuing value of the apgar score for the assessment of newborn infants", New England Journal of Medicine **344**, 7, 467–471 (2001).

[19] Chakraborti, T., K. P. Fadnis, K. Talamadupula, M. Dholakia, B. Srivastava, J. O. Kephart and R. K. Bellamy, "Visualizations for an explainable planning agent", arXiv preprint arXiv:1709.04517 (2017).

[20] Chakraborti, T., S. Sreedharan, Y. Zhang and S. Kambhampati, "Plan explanations as model reconciliation: Moving beyond explanation as soliloquy", in "26th International Joint Conference on Artificial Intelligence, IJCAI 2017", pp. 156–163 (International Joint Conferences on Artificial Intelligence, 2017).

[21] Chen, C., O. Li, D. Tao, A. Barnett, C. Rudin and J. K. Su, "This looks like that: deep learning for interpretable image recognition", Advances in neural information processing systems **32** (2019).

[22] Chen, C., Y. Liu, S. Kreiss and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning", in "2019 International Conference on Robotics and Automation (ICRA)", pp. 6015–6022 (IEEE, 2019).

[23] Chen, Y., C. Liu, B. E. Shi and M. Liu, "Robot navigation in crowds by graph convolutional networks with attention learned from human gaze", IEEE Robotics and Automation Letters **5**, 2, 2754–2761 (2020).

[24] Chitnis, R. H., H. Esfandiari, M. Hajiaghayi, R. Khandekar, G. Kortsarz and S. Seddighin, "A tight algorithm for strongly connected steiner subgraph on two terminals with demands", in "International Symposium on Parameterized and Exact Computation", pp. 159–171 (Springer, 2014).

[25] Cimatti, A., A. Micheli and M. Roveri, "Dynamic controllability of disjunctive temporal networks: Validation and synthesis of executable strategies", in "Thirtieth AAAI Conference on Artificial Intelligence", (2016).

[26] Clement, B. J., J. Barreiro, M. J. Iatauro, R. L. Knight and J. D. Frank, "Spatial planning for international space station crew operations", in "Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space", (Citeseer, 2010).

[27] Coulom, R., "Efficient selectivity and backup operators in monte-carlo tree search", in "International conference on computers and games", pp. 72–83 (Springer, 2006).

[28] Crowston, K., "Amazon mechanical turk: A research tool for organizations and information systems scholars", in "Shaping the future of ict research. methods and approaches", pp. 210–221 (Springer, 2012).

[29] Dragan, A. and S. Srinivasa, "Generating legible motion", (2013).

[30] Dragan, A. D., *Legible robot motion planning*, Ph.D. thesis, Carnegie Mellon University (2015).

[31] Du, M., N. Liu and X. Hu, "Techniques for interpretable machine learning", Communications of the ACM **63**, 1, 68–77 (2019).

[32] Endsley, M. R., "From here to autonomy: lessons learned from human–automation research", Human factors **59**, 1, 5–27 (2017).

[33] Evans, O., A. Stuhlmüller and N. Goodman, "Learning the preferences of ignorant, inconsistent agents", in "Thirtieth AAAI Conference on Artificial Intelligence", (2016).

[34] Fazlollahtabar, H. and M. Saidi-Mehrabad, *Autonomous guided vehicles*, vol. 20 (Springer, 2015).

[35] Feldman, J. and M. Ruhl, "The directed steiner network problem is tractable for a constant number of terminals", SIAM Journal on Computing **36**, 2, 543–561 (2006).

[36] Fikes, R. E. and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving", Artificial intelligence **2**, 3-4, 189–208 (1971).

[37] Fruchterman, T. M. and E. M. Reingold, "Graph drawing by force-directed placement", Software: Practice and experience **21**, 11, 1129–1164 (1991).

[38] Gao, M., L. Popowski and J. Boerkoel, "Dynamic control of probabilistic simple temporal networks", in "Proceedings of the AAAI Conference on Artificial Intelligence", vol. 34, pp. 9851–9858 (2020).

[39] Gopalakrishnan, S. and S. Kambhampati, "Tge-viz: Transition graph embedding for visualization of plan traces and domains", arXiv preprint arXiv:1811.09900 URL `https://arxiv.org/pdf/1811.09900.pdf` (2018).

[40] Gopalakrishnan, S. and S. Kambhampati, "Minimizing robot navigation-graph for position-based predictability by humans", arXiv preprint arXiv:2010.15255 URL https://arxiv.org/pdf/2010.15255.pdf (2020).

[41] Gopalakrishnan, S., V. Mudit and S. Kambhampati, "Synthesizing policies that account for human execution errors caused by state aliasing in markov decision processes", ICAPS 2021 Workshop on Explainable AI Planning URL https://openreview.net/pdf?id=aqRrDbKxOel (2021).

[42] Gopalakrishnan, S., V. Mudit and S. Kambhampati, "Computing policies that account for the effects of human agent uncertaintyduring execution in markov decision processes", arXiv preprint arXiv:2109.07436 URL https://arxiv.org/pdf/2109.07436.pdf (2022).

[43] Griffiths, T. L., F. Lieder and N. D. Goodman, "Rational use of cognitive resources: Levels of analysis between the computational and the algorithmic", Topics in cognitive science **7**, 2, 217–229 (2015).

[44] Habibi, F., F. Barzinpour and S. Sadjadi, "Resource-constrained project scheduling problem: review of past and recent developments", Journal of project management **3**, 2, 55–88 (2018).

[45] Hagberg, A. A., D. A. Schult and P. J. Swart, "Exploring network structure, dynamics, and function using networkx", in "Proceedings of the 7th Python in Science Conference", edited by G. Varoquaux, T. Vaught and J. Millman, pp. 11 – 15 (Pasadena, CA USA, 2008).

[46] Haslum, P., N. Lipovetzky, D. Magazzeni and C. Muise, "An introduction to the planning domain definition language", Synthesis Lectures on Artificial Intelligence and Machine Learning **13**, 2, 1–187 (2019).

[47] Hathaliya, J. J. and S. Tanwar, "An exhaustive survey on security and privacy issues in healthcare 4.0", Computer Communications **153**, 311–335 (2020).

[48] Helmert, M., "The Fast Downward planning system", JAIR **26**, 191–246 (2006).

[49] Hernández, J. M. and P. Van Mieghem, "Classification of graph metrics", Delft University of Technology, Tech. Rep pp. 1–8 (2011).

[50] Hiroi, Y. and A. Ito, "Are bigger robots scary?—the relationship between robot size and psychological threat—", in "2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics", pp. 546–551 (IEEE, 2008).

[51] Hollnagel, E., "Cognitive ergonomics: it's all in the mind", Ergonomics **40**, 10, 1170–1182 (1997).

[52] Hsu, H. and P. A. Lachenbruch, "Paired t test", Wiley StatsRef: statistics reference online (2014).

[53] Hu, Q. and W. Yue, *Markov decision processes with their applications*, vol. 14 (Springer Science & Business Media, 2007).

[54] Ibe, O., *Markov processes for stochastic modeling* (Newnes, 2013).

[55] Kahneman, D., S. P. Slovic, P. Slovic and A. Tversky, *Judgment under uncertainty: Heuristics and biases* (Cambridge university press, 1982).

[56] Kistan, T., A. Gardi and R. Sabatini, "Machine learning and cognitive ergonomics in air traffic management: Recent developments and considerations for certification", Aerospace **5**, 4, 103 (2018).

[57] Knight, H., "Expressive motion for low degree-of-freedom robots", (2016).

[58] Kocsis, L. and C. Szepesvari, "Bandit-based monte-carlo planning", in "Proceedings of ECML'06", (2006).

[59] Kretzschmar, H., M. Spies, C. Sprunk and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning", The International Journal of Robotics Research **35**, 11, 1289–1307 (2016).

[60] Kruse, T., A. Kirsch, E. A. Sisbot and R. Alami, "Exploiting human cooperation in human-centered robot navigation", in "19th International Symposium in Robot and Human Interactive Communication", pp. 192–197 (IEEE, 2010).

[61] Kruse, T., A. K. Pandey, R. Alami and A. Kirsch, "Human-aware robot navigation: A survey", Robotics and Autonomous Systems **61**, 12, 1726–1743 (2013).

[62] Kulkarni, A., S. Sreedharan, S. Keren, T. Chakraborti, D. E. Smith and S. Kambhampati, "Design for interpretability", in "ICAPS Workshop on Explainable AI Planning (XAIP)", (2019).

[63] Kumar, A. and S. Zilberstein, "History-based controller design and optimization for partially observable mdps", in "Proceedings of the International Conference on Automated Planning and Scheduling", vol. 25 (2015).

[64] Kwon, M., E. Biyik, A. Talati, K. Bhasin, D. P. Losey and D. Sadigh, "When humans aren't optimal: Robots that collaborate with risk-aware humans", in "2020 15th ACM/IEEE International Conference on Human-Robot Interaction (HRI)", pp. 43–52 (IEEE, 2020).

[65] Lage, I., D. Lifschitz, F. Doshi-Velez and O. Amir, "Exploring computational user models for agent policy summarization", arXiv preprint arXiv:1905.13271 (2019).

[66] Lasi, H., P. Fettke, H.-G. Kemper, T. Feld and M. Hoffmann, "Industry 4.0", Business & information systems engineering **6**, 4, 239–242 (2014).

[67] Lee, C. and C. Lim, "From technological development to social advance: A review of industry 4.0 through machine learning", Technological Forecasting and Social Change **167**, 120653, URL `https://www.sciencedirect.com/science/article/pii/S0040162521000858` (2021).

[68] Lin, R., S. Kraus, J. Wilkenfeld and J. Barry, "Negotiating with bounded rational agents in environments with incomplete information using an automated agent", Artificial Intelligence **172**, 6-7, 823–851 (2008).

[69] Littman, M. L., "Memoryless policies: Theoretical limitations and practical results", From animals to animats **3**, 238–245 (1994).

[70] Liu, B., M. Ding, S. Shaham, W. Rahayu, F. Farokhi and Z. Lin, "When machine learning meets privacy: A survey and outlook", ACM Computing Surveys (CSUR) **54**, 2, 1–36 (2021).

[71] Magnaguagno, M. C., R. F. Pereira, M. D. Móre and F. Meneguzzi, "Web planner: A tool to develop classical planning domains and visualize heuristic state-space search", in "Proceedings of the Workshop on User Interfaces and Scheduling and Planning, UISP", pp. 32–38 (2017).

[72] McDonald, J. D., L. A. DeChurch, R. Asencio, D. R. Carter, J. R. Mesmer-Magnus and N. S. Contractor, "Team task switching: A conceptual framework for understanding functional work shifts", in "Proceedings of the Human Factors and Ergonomics Society Annual Meeting", vol. 59, pp. 1157–1161 (SAGE Publications Sage CA: Los Angeles, CA, 2015).

[73] Meuleau, N., K.-E. Kim, L. P. Kaelbling and A. R. Cassandra, "Solving pomdps by searching the space of finite policies", arXiv preprint arXiv:1301.6720 (2013).

[74] Monsell, S., "Task switching", Trends in cognitive sciences **7**, 3, 134–140 (2003).

[75] Morris, P., N. Muscettola, T. Vidal *et al.*, "Dynamic control of plans with temporal uncertainty", in "IJCAI", vol. 1, pp. 494–502 (2001).

[76] Moukrim, A., A. Quilliot and H. Toussaint, "An effective branch-and-price algorithm for the preemptive resource constrained project scheduling problem based on minimal interval order enumeration", European Journal of Operational Research **244**, 2, 360–368 (2015).

[77] Müller, J., C. Stachniss, K. O. Arras and W. Burgard, "Socially inspired motion planning for mobile robots in populated environments", in "Proc. of International Conference on Cognitive Systems", (2008).

[78] Nikolaidis, S., A. Kuznetsov, D. Hsu and S. Srinivasa, "Formalizing human-robot mutual adaptation: A bounded memory model", in "2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)", pp. 75–82 (IEEE, 2016).

[79] Nikolaidis, S., Y. X. Zhu, D. Hsu and S. Srinivasa, "Human-robot mutual adaptation in shared autonomy", in "2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI", pp. 294–302 (IEEE, 2017).

[80] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine learning in Python", Journal of Machine Learning Research **12**, 2825–2830 (2011).

[81] Ramakrishnan, R., E. Kamar, D. Dey, E. Horvitz and J. Shah, "Blind spot detection for safe sim-to-real transfer", Journal of Artificial Intelligence Research **67**, 191–234 (2020).

[82] Ramakrishnan, R., E. Kamar, B. Nushi, D. Dey, J. Shah and E. Horvitz, "Overcoming blind spots in the real world: Leveraging complementary abilities for joint execution", in "Proceedings of the AAAI Conference on Artificial Intelligence", vol. 33, pp. 6137–6145 (2019).

[83] Ramakrishnan, R., V. Unhelkar, E. Kamar and J. Shah, "A bayesian approach to identifying representational errors", arXiv preprint arXiv:2103.15171 (2021).

[84] Rao, S. K., P. Sadayappan, F. K. Hwang and P. W. Shor, "The rectilinear steiner arborescence problem", Algorithmica **7**, 1, 277–288 (1992).

[85] robotics, C., "One way automated guided vehicle", http://www.agvrobotor.com/sale-10982400-one-way-automated-guided-vehicle-hospital-smart-cart-agv-with-magnetic-drive-sensor.html, accessed: 2020-10-06 (2021).

[86] Rousseeuw, P. J. and M. Hubert, "Robust statistics for outlier detection", Wiley interdisciplinary reviews: Data mining and knowledge discovery **1**, 1, 73–79 (2011).

[87] Russell, S. and P. Norvig, "Artificial intelligence: A modern approach, global edition 4th", Foundations **19**, 23 (2021).

[88] Saint-Guillain, M., T. S. Vaquero and S. A. Chien, "Lila: Optimal dispatching in probabilistic temporal networks using monte carlo tree search", in "31st International Conference on Automated Planning and Scheduling", p. 38 (2021).

[89] Shah, J. A., J. Stedl, B. C. Williams and P. Robertson, "A fast incremental algorithm for maintaining dispatchability of partially controllable plans.", in "ICAPS", pp. 296–303 (2007).

[90] Sharma, M., H. Elmiligi and F. Gebali, "Performance evaluation of real-time systems", International Journal of Computing and Digital Systems **4**, 01 (2015).

[91] Shi, W. and C. Su, "The rectilinear steiner arborescence problem is np-complete.", in "SODA", pp. 780–787 (Citeseer, 2000).

[92] Shinners, P., "Pygame", `http://pygame.org/` (2011).

[93] Simon, H. A., "Bounded rationality", in "Utility and probability", pp. 15–18 (Springer, 1990).

[94] Staal, M. A., "Stress, cognition, and human performance: A literature review and conceptual framework", (2004).

[95] Stankovic, J. A. and R. Rajkumar, "Real-time operating systems", Real-Time Systems **28**, 2-3, 237–253 (2004).

[96] Sund, D., "Comparison of visualization algorithms for graphs and implementation of visualization algorithm for multi-touch table using javafx", (2016).

[97] Trautman, P. and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds", in "2010 IEEE/RSJ International Conference on Intelligent Robots and Systems", pp. 797–803 (IEEE, 2010).

[98] Trautman, P., J. Ma, R. M. Murray and A. Krause, "Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation", The International Journal of Robotics Research **34**, 3, 335–356 (2015).

[99] Ustun, B. and C. Rudin, "Supersparse linear integer models for optimized medical scoring systems", Machine Learning **102**, 3, 349–391 (2016).

[100] Vasquez, D., B. Okal and K. O. Arras, "Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison", in "2014 IEEE/RSJ International Conference on Intelligent Robots and Systems", pp. 1341–1346 (IEEE, 2014).

[101] Vicente, K. J., "Ecological interface design: A research overview", Analysis, Design and Evaluation of Man–Machine Systems 1995 pp. 623–628 (1995).

[102] Vicente, K. J. and J. Rasmussen, "Ecological interface design: Theoretical foundations", IEEE Transactions on systems, man, and cybernetics **22**, 4, 589–606 (1992).

[103] Vidal, T. and M. Ghallab, "Dealing with uncertain durations in temporal constraint networks dedicated to planning"', in "ECAI", pp. 48–54 (PITMAN, 1996).

[104] Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", Nature Methods **17**, 261–272 (2020).

[105] Watel, D., M.-A. Weisser, C. Bentz and D. Barth, "Directed steiner tree with branching constraint", in "International Computing and Combinatorics Conference", pp. 263–275 (Springer, 2014).

[106] White, D. J., "A survey of applications of markov decision processes", Journal of the operational research society **44**, 11, 1073–1096 (1993).

[107] Whitehead, S. D. and L.-J. Lin, "Reinforcement learning of non-markov decision processes", Artificial Intelligence **73**, 1-2, 271–306 (1995).

[108] Whittlestone, J., R. Nyrup, A. Alexandrova, K. Dihal and S. Cave, "Ethical and societal implications of algorithms, data, and artificial intelligence: a roadmap for research", London: Nuffield Foundation (2019).

[109] Wickens, C. D., A. Santamaria and A. Sebok, "A computational model of task overload management and task switching", in "Proceedings of the human factors and ergonomics society annual meeting", vol. 57, pp. 763–767 (SAGE Publications Sage CA: Los Angeles, CA, 2013).

[110] Woods, D. D., "Coping with complexity: the psychology of human behaviour in complex systems", in "Tasks, errors, and mental models", pp. 128–148 (1988).

[111] Zeng, J., B. Ustun and C. Rudin, "Interpretable classification models for recidivism prediction", Journal of the Royal Statistical Society: Series A (Statistics in Society) **180**, 3, 689–722 (2017).

[112] Zhi-Xuan, T., J. Mann, T. Silver, J. Tenenbaum and V. Mansinghka, "Online bayesian goal inference for boundedly rational planning agents", Advances in Neural Information Processing Systems **33**, 19238–19250 (2020).

[113] Zhu, J., X. Li and W. Shen, "Effective genetic algorithm for resource-constrained project scheduling with limited preemptions", International Journal of Machine Learning and Cybernetics **2**, 2, 55–65 (2011).

[114] Zhu, X., J. Lafferty and Z. Ghahramani, "Combining active learning and semi-supervised learning using gaussian fields and harmonic functions", in "ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining", vol. 3 (2003).