

MINIMIZING COMMUNICATION COST IN AN N-AGENT
DISTRIBUTED BAYESIAN NETWORK BY USING A
DECENTRALIZED MDP

by

Jonathan Koren

B.S., Southern Illinois University, 1998

A Thesis
Submitted in Partial Fulfillment of the Requirements for the
Master of Science Degree

Department of Computer Science
in the Graduate School
Southern Illinois University Carbondale
July, 2005

Copyright Jonathan Koren, 2005

All Rights Reserved.

AN ABSTRACT OF THE THESIS OF

Jonathan Koren, for the Masters of Science degree in Computer Science, presented on July 1, 2005, at Southern Illinois University at Carbondale.

TITLE: Minimizing Communication Cost in an N-Agent Distributed Bayesian Network by Using a Decentralized MDP

MAJOR PROFESSOR: Dr. Norman F. Carver III

Distributed sensor interpretation (DSI) and *distributed diagnosis* (DD) is an application of multiagent systems. DSI/DD systems consist of a network of sensors distributed over a large geographic area. A system of cooperative agents are tasked with reading the sensors in order to determining which event(s) from a given set have occurred. DSI/DD systems enable graceful degradation and an decreased time to solution when compared to centralized SI systems.

In order to properly determine the event(s) the agents are tasked with interpreting, the agents must exchange either data or local solutions among themselves. Without such exchanges, it would be difficult, if not impossible, for the agents to maintain a globally consistent solution to the interpretation problem. Since the agents in a DSI/DD system are assumed to have limited resources, communication among the agents must be kept to a minimum. Previous approaches to this problem involved “satisficing” algorithms that trade a lower solution quality in exchange for less communication.

Previous work combined a satisficing algorithm with a *decentralized Markov decision process* (DEC-MDP) in order to discover the optimal communication

strategy. While the previous work presented a basic strategy, it was lacking in certain respects. These included the fact that the system only included two agents and the lack of a sufficiently detailed action resolution and cost model.

This thesis extends that work to systems of an arbitrary number of agents. This work developed a more realistic model of DSI systems through an enhanced cost model and parallel action resolution. The previous model was also extended to allow agents to take advantage of partial solutions discovered by the other agents in the system. Lastly, a method analyzing the complexity of applying a DEC-MDP to a specific DSI/DD system was developed.

ACKNOWLEDGMENTS

Dr. Carver for his patience and guidance. My father for taking me to the Carbondale K-Mart to buy my first computer, and my mother for not only tirelessly typing in 1,000 line BASIC programs in to that Commodore 64, but also for her suggestion to get a disk drive.

This material is based upon work supported by the National Science Foundation under Grant Nos. IIS-0084135 and IIS-0414945. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

ABSTRACT	i
ACKNOWLEDGMENTS	iii
LIST OF TABLES	vii
LIST OF FIGURES	viii
LIST OF ALGORITHMS	ix
1 INTRODUCTION	1
2 REVIEW OF DISTRIBUTED SENSOR INTERPRETATION	4
2.1 Cooperative Distributed Sensor Interpretation	4
2.2 Belief Networks	5
3 REVIEW OF MARKOV DECISION PROCESSES	7
3.1 Markov Decision Processes	7
3.1.1 Description of MDPs	7
3.1.2 Solving MDPs	8
3.2 Multiagent Markov Decision Processes	11
3.3 Decentralized Markov Decision Processes	15
4 APPLYING DEC-MDPS TO DSI	17
4.1 Description of Problem	17
4.2 Previous Research	19
5 EXTENDING TO N AGENTS	21
5.1 Local State to Global State Mapping	21

5.2	Extending the Bellman Equation	22
5.3	Network Topology	23
5.4	Parallel Action Resolution	24
5.5	Addition of Likelihood-Vector Actions	24
5.6	Discussion of the Cost Model	26
6	IMPLEMENTATION	31
6.1	Why Implementation is Hard	31
6.2	Code Metrics	31
6.3	Evolution of State Representation	32
6.3.1	Storing Available Data Only	32
6.3.2	Explicit Storing of Communication History	33
6.3.3	Addition of Confidence-Flags	33
6.4	Performance	35
6.4.1	Näive Lists Versus Generating Functions	35
6.4.2	precalculation of Confidence-Flags	37
6.4.3	Precalculation of Normalization Constants	38
6.5	Determining the Number of Local States	40
6.6	Use of Generators	43
6.7	Checkpointing	45
6.8	Network Simulator	46
7	EXAMINATION OF THE SYSTEM	48
7.1	Formal Description of System	48
7.2	Experimental Procedure	51
7.3	Examination of Policies	51
7.4	Effect of Likelihood-Vectors	52

8	CONCLUSIONS	53
8.1	Conclusions	53
8.2	Future Work	53
	REFERENCES	55
	APPENDICES	56
	VITA	72

LIST OF TABLES

3.1	Mountain Climbing MDP Transition Function Definitions	10
3.2	Mountain Climbing MDP Values $k = 1$	12
3.3	Mountain Climbing MDP Values $k = 2$	13
3.4	Mountain Climbing MDP Values $k = 8$	14
6.1	An Example $\hat{\mathcal{G}}^s$	34
6.2	An Example $\hat{\mathcal{G}}^{s'}$	35
6.3	Values for $f(a, i)$	42

LIST OF FIGURES

3.1 Mountain Climbing MDP	9
-------------------------------------	---

LIST OF ALGORITHMS

6.1	Näive State Listing	36
6.2	Determine Confidence-Flags Patterns	38
6.3	Determine Normalization Constants	40
6.4	Basic DEC-MDP Solving	47

CHAPTER 1

INTRODUCTION

With the advent of inexpensive computing power and large high speed networks, interest in the application of intelligent multiagent systems has grown. One area of research is that of *distributed sensor interpretation* (DSI) and *distributed diagnosis* (DD). DSI/DD systems involve large numbers of sensors distributed over a wide geographic area. Intelligent agents are distributed among the sensors in order to observe and control the sensors locally.

By processing the data from the sensors, the agents must determine which event(s) from a given set occurred. To accomplish this, the agents must cooperate by exchanging either data or partial solutions among themselves in order to achieve a high-quality globally consistent solution. Since each agent in a DSI system has limited resources, the communication among the agents must be minimized. Applications of DSI/DD include situation assessment and decision support systems[1].

DSI was one of the first applications of multiagent systems to be studied[2]. While it has been extensively investigated, previously developed methods for limiting computation and communication among the agents were primarily based on ad hoc heuristics. Only recently have formal methods been applied to this domain. One of the first formal examinations of DSI[3] attempted to minimize the communication among the agents by modeling a DSI system as a *decentralized*

Markov decision process (DEC-MDP). A DEC-MDP can discover the optimal course of action for the agents in a multiagent system. A DEC-MDP accomplishes this by representing a system as a series of states, with actions enabling transitions between states. When an agent performs an action, it receives a reward based on the state that the agent was in when the action was performed and what state the agent entered as a result of that action. By maximizing the long term reward, the agents develop the optimal course of action.

While the previous work provided the basic approach to utilizing DEC-MDPs within the DSI problem domain, it suffered from several deficiencies. First, the previous work examined only two agent systems. Second, the cost model used to determine the agent rewards was too simplistic. Finally, the previous work did not address the issue of complexity of specific DSI systems.

This thesis attempts to address some of these issues. This work primarily is concerned with extending DEC-MDPs to DSI systems of an arbitrary number of agents. In order to accomplish this, weaknesses of the previous work's state representation were discovered and addressed. The difficulties of an agent extrapolating the internal states of the other agents based on its own internal state were resolved. A method for analyzing the complexity of applying a DEC-MDP to specific DSI problems was as developed. Finally, implementation insights into exploiting DEC-MDPs for this problem domain were developed.

This thesis begins with a review of cooperative distributed problem solving and how it relates to distributed sensor interpretation and distributed diagnosis. The advantages of a distributed interpretation system over a centralized system are outlined, along with some of the complexities a distributed system introduces. Chapter 2 closes with a description of how a sensor network can be modeled as a

Bayesian belief network.

In Chapter 3, *Markov decision processes* (MDPs) are discussed. An MDP is a model that describes how an agent interacts with an environment in order to solve a problem. Through techniques, such as *value iteration*, MDPs allow an agent to learn the optimal course of action to solve the problem. Multiagent extensions to MDPs, such as multiagent and decentralized MDPs are also discussed in this chapter.

Chapter 4 illustrates how DEC-MDPs have previously been applied to DSI problems. The shortcomings of the previous systems will be outlined. Primarily these shortcomings are the lack of a detailed cost model, insufficient analysis of the complexity of the problem domain, and that previous systems consisted of only two agents.

Chapters 5 and 6 address these shortcomings. In these chapters, the issues regarding systems of an arbitrary number of agents are discussed, and solutions proposed. Chapter 5 emphasizes the theoretical issues, such as local state to global state mapping, the development of a more detailed cost model, and the analysis of the approximate complexity of specific DSI systems. On the other hand, Chapter 6 emphasizes implementation issues such as state representation and search performance.

Finally, a series of experiments to determine the efficacy of the proposed extensions are presented. The conclusions that can be drawn from these experiments are offered along with some potential future directions for this research.

CHAPTER 2

REVIEW OF DISTRIBUTED SENSOR INTERPRETATION

2.1 COOPERATIVE DISTRIBUTED SENSOR INTERPRETATION

Research in *cooperative distributed problem solving* (CDPS) revolves around using systems of agents that work together to solve a large-scale problems. These problems include *distributed sensor interpretation* (DSI) and *distributed diagnosis* (DD). DSI/DD problems involve large numbers of sensors dispersed over a wide geographic area. Agents examine the sensor data and use it to determine which (if any) events from a set of possible events occurred.

DSI/DD systems have several advantages over similar centralized systems. First, DSI/DD systems provide redundancy and graceful degradation since there is no longer a single point of failure in the interpretation system. More importantly is that DSI/DD systems should decrease the time to solution. DSI/DD systems allow for parallel operation with multiple sensors and events being examined simultaneously. DSI/DD systems allow agents to be placed closer to the sensors, thus reducing latency when querying, and possibly controlling, sensors in a certain area. Furthermore, overall network bandwidth can be reduced by allowing a great deal of computation to occur closer to the individual sensors; thus requiring only partial results to be shared among the agents.

In a DSI/DD approach, the problem of interpretation is decomposed into a set of subproblems, such as interpreting a subset of the events, are distributed among the agents in the system. Implicit in this approach is the assumption that the agents' solutions to their individual local subproblems can be combined to create an overall global solution. However, these subproblems rarely can be solved in isolation. The subproblems frequently interrelate, and therefore require the agents to communicate local data or solutions to the other agents in order to produce a globally consistent solution. Since a significant amount of communication may be required to guarantee a globally consistent solution, several "satisficing" approaches have been developed[1].

A satisficing approach trades a lower solution quality (i.e. the likelihood that the solution arrived at by the distributed system is the same as the solution arrived at by a centralized system) for less communication or computation among the agents. These approaches are often described as being able to reach a required solution quality with a certain average amount of communication or computation. One such approach is described in Chapter 4.

2.2 BELIEF NETWORKS

Sensor interpretation and diagnosis problems are often expressed as a *belief network* (BN). A belief network consists of a series of levels made up of a set of nodes. Each node represents a random variable, and has the probability of each of its values associated with it. For example, a Boolean variable would have two probabilities associated with it, one for when the variable is true and another for when it is false. The nodes in each level are connected to nodes located in the lower levels by a set of directed arcs. Each arc represents a causal relationship

between the nodes. Nodes with incoming arcs have a set of conditional probabilities associated with them. Each conditional probability indicates the value of the node given the values of the directly connected higher level nodes. An example belief network is shown in Figure C.1, and the associated probabilities are shown in Table C.1.

Generally speaking, nodes at the top level of a belief network represent the events that are to be diagnosed, and the lowest levels represent data from the individual sensors that are used in the diagnosis.

CHAPTER 3

REVIEW OF MARKOV DECISION PROCESSES

3.1 MARKOV DECISION PROCESSES

3.1.1 Description of MDPs

The interaction of an agent with its environment can often be formalized as a *Markov decision process* (MDP)[4]. An MDP consists of a single agent interacting with an environment. The environment is presented to the agent as a set of reachable states, \mathcal{S} . In each state $s \in \mathcal{S}$, the agent can perform a single action a from a set of actions available from the current state, denoted as \mathcal{A}_s . Performing an action results in the agent transitioning to a new state s' with the probability of $P(s' | a, s)$. Upon transitioning to the new state, the agent receives a reward $R(s', a, s)$. The agent's goal is to find a mapping of actions to states that results in the maximum long term expected reward. This mapping is called a *policy* and is denoted as π . Once such a mapping is found, the MDP is said to be solved.

It is important to note that MDPs can be applied only to systems that exhibit the Markov property. Systems that exhibit this property have states that contain all the information required to select the optimal action from that state. This does not mean the entire history of the actions taken must be in the state. For instance, the game of chess exhibits the Markov property since the arrangement of the pieces completely encapsulates all that is needed from the history of the game.

To see how an MDP is used, consider a mountain climber. The climber starts

at the base of a mountain, and tries to reach the summit as quickly as possible. There are two main paths up the mountain. A direct, but difficult path, and an easier, but more circuitous, path. Near the end of the easier path, there are two approaches to the summit.

During the ascent, the climber will have to make camp several times. If he takes the difficult path, he will need to camp once, but if the easier path is chosen, then he will need to camp twice. Furthermore, every time the climber makes an attempt to advance up the mountain, there is no guarantee that he will actually advance. He may be forced to return to his most recent encampment, or even backtrack to a previous encampment.

This problem description can be formalized as an MDP by considering each of the climber's encampments as a state, and each of climbing trails leading from an encampment as an action. Each attempt to advance up the mountain results in the climber either advancing to the next state, remaining in the current state, or regressing to the previous state. Probabilities for each of these results are assigned to complete the state transition function. To encourage the agent to complete the task as quickly as possible, a negative reward is assigned to each action taken. The task begins with the climber at the base of the mountain, and terminates when he reaches the summit. Figure 3.1 and Table 3.1 illustrate this setup.

3.1.2 Solving MDPs

In order to find the optimal policy π^* , the various states must be compared to each other. A state is considered better if its actions are more likely to maximize the agent's long term expected reward. The long term expected reward from a state is called the state's *value*, and is denoted by $V(s)$. A state's value can be

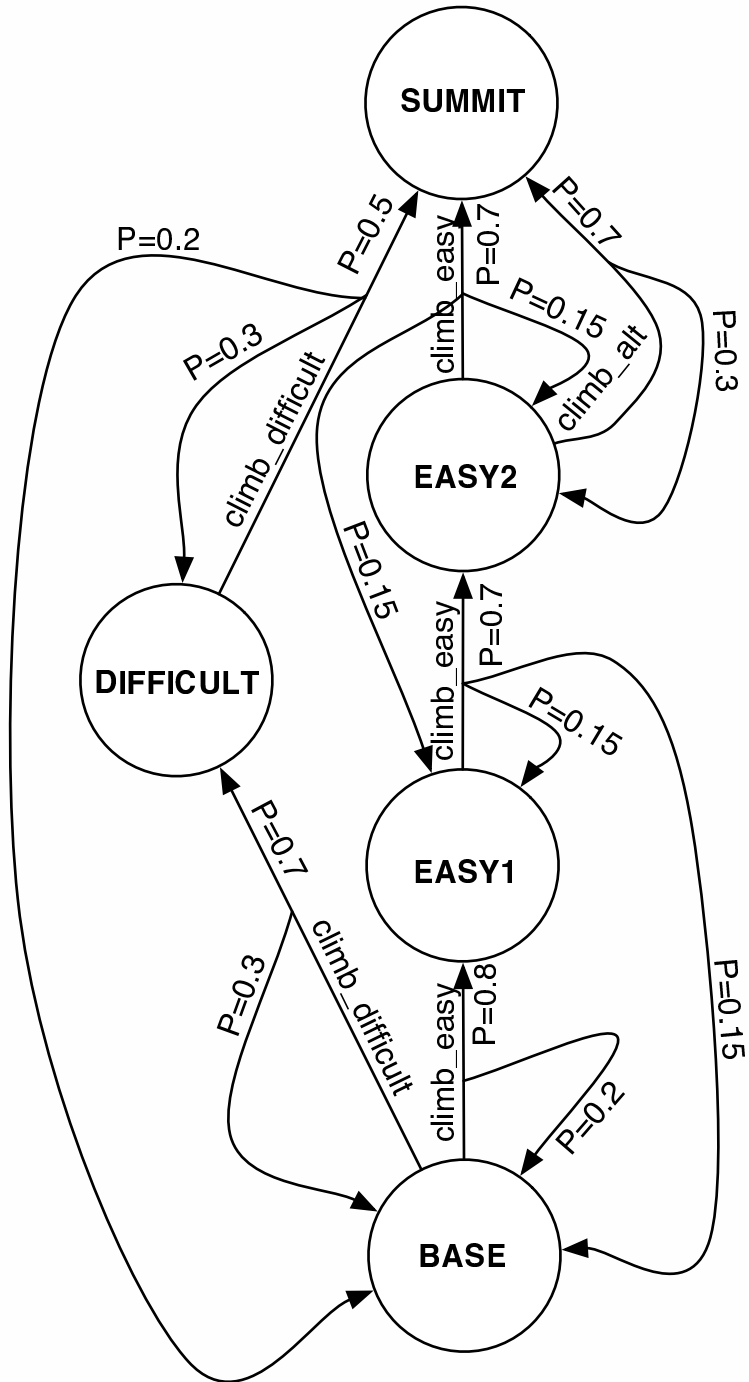


Figure 3.1: Mountain Climbing MDP

Table 3.1: Mountain Climbing MDP Transition Function Definitions

s	a	s'	$P(s' a, s)$	$R(s', a, s)$
BASE	climb_difficult	DIFFICULT	0.7	-1
BASE	climb_difficult	BASE	0.3	-1
BASE	climb_easy	EASY1	0.8	-1
BASE	climb_easy	BASE	0.2	-1
DIFFICULT	climb_difficult	SUMMIT	0.5	-1
DIFFICULT	climb_difficult	DIFFICULT	0.3	-1
DIFFICULT	climb_difficult	BASE	0.2	-1
EASY1	climb_easy	EASY2	0.7	-1
EASY1	climb_easy	EASY1	0.15	-1
EASY1	climb_easy	BASE	0.15	-1
EASY2	climb_easy	SUMMIT	0.7	-1
EASY2	climb_easy	EASY2	0.15	-1
EASY2	climb_easy	EASY1	0.15	-1
EASY2	climb_alt	SUMMIT	0.7	-1
EASY2	climb_alt	EASY2	0.3	-1

found, subject to certain conditions, through a finite number of iterations of the *Bellman equation*[5]. The Bellman equation is expressed as:

$$V_{k+1}(s) = \max_{a \in \mathcal{A}_s} \sum_{s'} P(s' | a, s) (R(s', a, s) + \gamma V_k(s')) \quad (3.1)$$

where k is the iteration index, and γ is the *discount factor*. The discount factor represents the relative importance of long term rewards compared to immediate rewards, and is defined on the interval $[0, 1)$.

It then follows that the current approximation of π^* can be expressed as:

$$\pi_{k+1}(s) = \operatorname{argmax}_{a \in \mathcal{A}_s} \sum_{s'} P(s' | a, s) (R(s', a, s) + \gamma V_k(s')) \quad (3.2)$$

The iterative nature of these equations allow policies to be found through dynamic programming techniques. The technique of primary concern to this work is *value iteration*. Value iteration works by repeatedly applying Equations 3.1 and 3.2 to every state. This continues until $\forall s \in \mathcal{S} : |V_{k+1}(s) - V_k(s)| < \theta$, where θ is the threshold for stability, at which point, the stable policy π is returned[6].

To illustrate this, recall the mountain climbing example from Section 3.1.1. For this MDP, γ is set to 0.7, and θ to 0.01. Initially the values for all the states are set to 0. Tables 3.2, 3.3, and 3.4 show the progress of value iteration after the first, second, and eighth iterations respectively. Note that a stable policy is found after the eighth iteration.

3.2 MULTIAGENT MARKOV DECISION PROCESSES

Standard MDPs are defined with regard to a single agent acting alone in an environment. In order to find optimal policies for agents in a cooperative multiagent system, an MDP extension known as *multiagent Markov decision processes* (MMDPs)[7] must be used.

In MMDPs each agent i in the set of agents α experiences the same state as all the other agents. This is called the *joint state*. Each agent has its own set of actions it can perform for each joint state. An action performed by an individual agent is referred to as its *local action*. The set of actions taken by all the agents in the current state, is called the *joint action*. For example, if there are two agents, and the first agent performs the action *foo* and the second agent performs the

θ	V	π	s	a	s'	$P(R + \gamma V_k)$	Σ	
1.0000	-1.0000	climb_easy	BASE	climb_difficult	DIFFICULT	-1.1900	-1.7000	
					BASE	-0.5100		
					climb_easy	EASY1		-1.3600
					BASE	-0.3400		
1.0000	-1.0000	climb_difficult	DIFFICULT	climb_difficult	SUMMIT	-0.5000	-1.3500	
					DIFFICULT	-0.5100		
					BASE	-0.3400		
1.0000	-1.0000	climb_easy	EASY1	climb_easy	EASY2	-1.1900	-1.7000	
					EASY1	-0.2550		
					BASE	-0.2550		
1.0000	-1.0000	climb_easy	EASY2	climb_easy	SUMMIT	-0.7000	-1.2100	
					EASY2	-0.2550		
					EASY1	-0.2550		
				climb_alt	SUMMIT	-0.7000	-1.2100	
					EASY2	-0.5100		
0.0000	0.0000		SUMMIT			0.0000	0.0000	

Table 3.2: Mountain Climbing MDP Values $k = 1$

θ	V	π	s	a	s'	$P(R + \gamma V_k)$	Σ	
0.7000	-1.7000	climb_difficult	BASE	climb_difficult	DIFFICULT	-1.3615	-2.0185	
					BASE	-0.6570		
					climb_easy	EASY1		-1.7520
					BASE	-0.4380		
0.3500	-1.3500	climb_difficult	DIFFICULT	climb_difficult	SUMMIT	-0.5000	-1.5215	
					DIFFICULT	-0.5835		
					BASE	-0.4380		
0.7000	-1.7000	climb_easy	EASY1	climb_easy	EASY2	-1.2929	-1.9499	
					EASY1	-0.3285		
					BASE	-0.3285		
0.2100	-1.2100	climb_alt	EASY2	climb_easy	SUMMIT	-0.7000	-1.3056	
					EASY2	-0.2771		
					EASY1	-0.3285		
				climb_alt	SUMMIT	-0.7000	-1.2541	
					EASY2	-0.5541		
0.0000	0.0000		SUMMIT			0.0000	0.0000	

Table 3.3: Mountain Climbing MDP Values $k = 2$

θ	V	π	s	a	s'	$P(R + \gamma V_k)$	Σ	
0.0075	-2.2975	climb_difficult	BASE	climb_difficult	DIFFICULT	-1.5186	-2.0185	
					BASE	-0.7825		
					climb_easy	EASY1		-1.9641
					BASE	-0.5217		
0.0040	-1.6706	climb_difficult	DIFFICULT	climb_difficult	SUMMIT	-0.5000	-1.5215	
					DIFFICULT	-0.6508		
					BASE	-0.5217		
0.0022	-2.0787	climb_easy	EASY1	climb_easy	EASY2	-1.3203	-1.9499	
					EASY1	-0.3683		
					BASE	-0.3912		
0.0000	-1.2658	climb_alt	EASY2	climb_easy	SUMMIT	-0.7000	-1.3056	
					EASY2	-0.2829		
					EASY1	-0.3683		
				climb_alt	SUMMIT	-0.7000	-1.2541	
					EASY2	-0.5658		
0.0000	0.0000		SUMMIT			0.0000	0.0000	

Table 3.4: Mountain Climbing MDP Values $k = 8$

action *bar*, then the joint action is $\langle foo, bar \rangle$. State transitions are defined in regard to the joint state and the joint action. Each agent receives a *joint reward*, which is similar to the reward function in a standard MDP, but is based off the joint state and the joint action instead. By having the agents receive a common joint reward, cooperation is ensured. This is because any agent that performs an action that would place any of the other agents at a disadvantage, is punished by the same amount that the other agent(s) are. Conversely, an agent is rewarded if it places another agent in an advantageous situation. Dynamic programming techniques like value iteration can be applied to determine the *joint policy* for the agents. The joint policy is the collection of all the individual agents' policies.

3.3 DECENTRALIZED MARKOV DECISION PROCESSES

MMDPs assume that all the agents can completely observe the environment. This is not necessarily true for certain problem domains. In response to this short-coming, *decentralized Markov decision processes* (DEC-MDPs) were developed[8].

A DEC-MDP is a distributed version of an MMDP. In a DEC-MDP, the agents still exist in a common environment, called the *global state*. However, no agent can directly observe this state. Instead, each agent experiences its own *local state* s_i . By examining its local state, each agent can partially observe the global state. A global state is considered *consistent* if none of the agents' local states are in conflict with any other agent's local state. The consistent global state is denoted by \hat{G} , and is defined as $\hat{G} = \prod_{i \in \alpha} s_i$. When necessary, the notation \hat{G}^s is used to denote a global state that is consistent with a specific local state s . It is important to note that communication between the agents must be explicitly modeled since

the agents are not omniscient.

DEC-MDPs are similar to MMDPs in that the state transition function is defined in terms of the global state and the agents' joint action. Also, agents receive a joint reward based on the global state transition and the joint action. Since the agents in an DEC-MDP have local states along with local actions, a *decentralized*, or local MDP is defined for each agent. Solving a local MDP results in a *local policy* π_i .

DEC-MDPs can be solved by solving each local MDP individually. By fixing all the remote agents' policies, the remote agents become part of the environment for the local agent. Once a local policy is discovered, the value of the local start state is compared to the previous value of the local start state. If the values are equal, then the agent is said to have a stable policy. The process repeats for each agent in the system, until all the agents have stabilized[3].

CHAPTER 4

APPLYING DEC-MDPS TO DSI

4.1 DESCRIPTION OF PROBLEM

The sensor interpretation problem examined in this and previous works can be modeled as a two-level belief network, with nodes representing Boolean events at the top, and nodes at the bottom representing individual pieces of Boolean sensor data. Conditional probabilities are assigned to denote the correlation between sensor data and events. This network structure, along with the accompanying conditional probabilities is referred to as a *probability grammar*, or simply as a “grammar”. Classes of grammars, like the belief networks they are based upon, are referred to by the number of nodes per level. For instance, a 2-10 grammar would contain 2 event nodes and 10 bottom level data nodes.

By examining the values of the data nodes, an agent can determine which events are likely to have occurred (e.g. have a Boolean value of true). This is called interpretation. Of particular interest is the *maximum a posteriori interpretation* (MAPI) of the data, as it is the optimal interpretation of a data set. Formally this is defined as:

$$\text{MAPI}(\epsilon) = \underset{h}{\operatorname{argmax}} P(h \mid \epsilon) \quad (4.1)$$

where ϵ is the data currently available to the agent, and h is the a possible interpretation of event(s) E .

The above description assumes a centralized interpretation system, but a distributed system can be easily obtained. In a distributed system, the data is spread among multiple agents. A data point may be initially local to a single agent, or it may be initially shared among several agents. Each agent in the system is tasked with correctly interpreting a subset of the events. In order to achieve this, the agents may be required to query each other for more data.

As an agent acquires more data, its MAPI may change. Therefore, each agent must determine how much *confidence* it has in its current interpretation. Confidence is expressed as the probability that the MAPI of its current subset of data is the same as the MAPI that would be determined if the agent had access to all the data available in the system (ϵ^*) [3]. This is defined as:

$$\begin{aligned} C(\text{MAPI}(\epsilon_i)) &= P(\text{MAPI}(\epsilon_i) = \text{MAPI}(\epsilon^*)) \\ &= \sum_{e \in \{\epsilon^* | \text{MAPI}(\epsilon_i) = \text{MAPI}(\epsilon^*)\}} P(e | \epsilon^*) \end{aligned} \tag{4.2}$$

where ϵ_i is the data currently available to agent i .

The advantage to using confidence as the termination criterion instead of simply $P(E_i | \epsilon_i)$ is that any level of confidence is guaranteed to be reached. Whereas with straight conditional probabilities an upper limit on the probability can be reached that is below the level of certainty desired.

In a distributed system it is unlikely that an agent's locally available data will be sufficient to interpret its event(s) with the desired level of confidence. In order to become sufficiently confident, the agent will need data communicated to it from the other agents. While having every agent simply broadcast all its local data to every other agent will guarantee that all the agents will reach their

confidence threshold, this is not considered a viable strategy for DSI networks. DSI networks are assumed to be very large networks of nodes with very limited resources especially with regards to power. Communication drains the nodes' power reserves, and therefore limits the lifespan of the network. Also, whenever a piece of data is received by an agent, the data point needs to be incorporated into the agent's model. This requires a certain amount of time and thereby increases the time to solution. These constraints require that the communication between the agents must be minimized.

4.2 PREVIOUS RESEARCH

DEC-MDPs have been previously applied to the presented DSI problem[3]. In the previous work, a 2-10 grammar was distributed between two agents. Each agent was required to reach a certain confidence level in determining the MAPI of its event. Initially, the data was distributed evenly between the two agents, with no common data held between the agents. Since the initial values of the data were not known to the agents before hand, each agent could have begun in one of several possible start states. In order to maintain the Markov property, every agent began in a dummy start state. This dummy start state contained no information, and was there merely to allow the agents to stochastically transition to their actual start states.

In order to reach the confidence threshold, the agents needed to exchange data. The agents would take turns deciding whether to communicate with the other agent or not. If an agent decided to initiate communication, it would either send one or more data points to the other agent, or request a specific set of data points to be sent to it. In order to encourage the agents to minimize communication, the

agents received a joint negative reward for each data point transmitted in a round. SEND actions resulted in a reward of $-|datasent|$, with REQUEST actions resulting in a reward of $(-1 - |datarequested|)$. The data available to each agent was considered the agent’s state, with terminal states indicated when both agents reached the desired level of confidence. Upon transitioning into a terminal state from a non-terminal state, the agents received a joint positive reward. This positive reward was required to encourage the agents to perform actions in which they were penalized to solve the problem.

While this setup was useful for developing an initial approach to using DEC-MDPs, there were three significant limitations. First, a two agent system is too small to yield any insights into large multiagent systems. Secondly, the agents used sequential actions (i.e. one agent would act, then the other agent would act, then the first agent would act again). For large systems, maintaining synchrony may be difficult and in some cases unwanted for performance reasons. Finally, the cost model used is too simplistic as it does not fully take into account the relative communication and computation costs involved in transmitting and incorporating data into the agents.

CHAPTER 5

EXTENDING TO N AGENTS

5.1 LOCAL STATE TO GLOBAL STATE MAPPING

Recall from Chapter 3 that the rewards and state transitions each agent experiences in a DEC-MDP are determined by the global state instead of the agent's local states. In order to predict the result of its action, an agent must attempt to determine the global state of the system. This task is complicated by the fact that several global states may be consistent with the agent's current local state.

In a two agent system, determining the global state reduces to simply determining the remote agent's state. Since the local agent is the only source of external data for the remote agent, the remote agent is guaranteed to have only the data that was initially distributed to it along with any data that was communicated to it by the local agent. The most difficult part of determining the remote agent's state becomes determining the values of the unseen data points.

In a system with more than two agents, the local agent is much less sure of what data each remote agent has, since the local agent knows only the contents of the messages it either sent or received. This means that each remote agent will have the data that was initially available to it, the data that it sent to the local agent, the data the local agent sent to it, possibly any combination of data that was initially available to the other remote agents, and possibly any combination of

data sent by the local agent to any of the other remote agents. Furthermore, as the number of agents in the system increases, the possible number of global states increases exponentially.

For example, consider a system with three agents named A , B , and C . Suppose each agent initially has one unique data point available to it. These data points are named A , B , and C and are initially distributed accordingly. Suppose Agent A neither sent nor received any data on the first turn. At the start of the second turn, agent A does not know whether agent B has access to only data point B or both B and C . Similarly, agent A does not know whether agent C has only the data point C or both B and C . This means that agent A can infer only that the system is in one of sixteen possible global states. Compare this to a two agent system where agent B has two initial data points and agent A has the one. With this data distribution, agent A can infer that that system is in one of a possible four global states.

5.2 EXTENDING THE BELLMAN EQUATION

Recall the Bellman equation (Equation 3.1). The Bellman equation is written with respect to standard MDPs. That is to say that, the reward and state transition functions are dependent on the agent's state. In a DEC-MDP the reward and state transition functions are dependent on the global state. However, each agent can only observe its own local state, and therefore make decisions based only on that limited information. This means that the Bellman equation must be modified to incorporate local state to global state mapping.

In the DEC-MDP described in this work, all actions are deterministic with respect to the global state. It then follows that $P(s' | a, s) = \sum_{\hat{G}; s'=\hat{G}_i} P(\hat{G} | s)$.

Since the joint reward function is dependent on the global state, the local reward received by the agent must be discounted by the probability of the agent being each possible global state. That is: $R(s', a, s) = \sum_{\hat{G}; s'=\hat{G}'_i} P(\hat{G} | s) R(\hat{G}, a, \hat{G}')$

Substituting the above equations in to the Bellman equation yields:

$$V_{k+1}(s) = \max_{a \in \mathcal{A}_s} \sum_{s'} \left(\sum_{\substack{\hat{G} \\ s'=\hat{G}'_i}} P(\hat{G} | s) \left(\left(\sum_{\substack{\hat{G} \\ s'=\hat{G}'_i}} P(\hat{G} | s) R(\hat{G}, a, \hat{G}') \right) + \gamma V_k(s') \right) \right) \quad (5.1)$$

5.3 NETWORK TOPOLOGY

Network topology can directly influence the cost of communication in real-world networks. With two agents, topology wasn't a concern, but with three or more agents, it makes sense to model the relative distances between agents and adjust communication costs accordingly.

In the system studied here, the agents are connected via a series of point-to-point links. Each agent has a direct connection to at least one other agent. Any agent can communicate with any other agent by routing communication along these links. (i.e. The communication network is not bipartite.) The communication cost between two agents increases with the number of hops between them. By explicitly modeling network topology, the agents are tasked with minimizing the total network utilization, instead of simply the number of data points exchanged.

5.4 PARALLEL ACTION RESOLUTION

The previous work[3] described a system where the agents performed actions in a round-robin manner. While a round-robin schema may be applicable to systems of a handful of agents that communicate over low latency links, it is not a reasonable for systems with a large number of agents or a system with high latency links. This is due to the long delay between successive actions performed by a specific agent. In order to accurately model large systems, agents must be able to perform actions asynchronously and in parallel.

In the system presented in this work, the agents synchronously perform actions in parallel. This means that on every time step, each agent evaluates its local state, selects its action. Once all actions are selected, all actions are performed in parallel based on local states at the start of the step. The local states are then updated, and the step next time step begins. While this is may not be as realistic for very large networks as asynchronous actions, it is simpler to model and could be used for moderate to large-scale networks.

5.5 ADDITION OF LIKELIHOOD-VECTOR ACTIONS

An original extension to the previous work, is allowing the agents to leverage computations performed by the other agents. To understand how this is done, the method of how the agents calculate probabilities must be explained.

Assume each agent has access to the complete belief network. That is, each agent knows from the grammar the priors of every event ($\forall E \in \mathcal{E} : P(E)$) and a vector of the conditional probabilities of every data point given every combination of events. Using this knowledge, each agent maintains a vector of $2^{|\mathcal{E}|}$ elements. This is the *likelihood-vector* and is used to maintain partial solutions during

interpretation.

For example, if $\mathcal{E} = \{E_1, E_2\}$, then the vector would be:

$$\vec{P}(\epsilon | \mathcal{E}) = \langle P(\epsilon | E_1 E_2), P(\epsilon | E_1 \bar{E}_2), P(\epsilon | \bar{E}_1 E_2), P(\epsilon | \bar{E}_1 \bar{E}_2) \rangle$$

Initially, the likelihood-vector is contains the priors of every combination of events in the system. Every time the agent interprets a new data point ϵ , it looks up the appropriate conditional probabilities from the grammar and multiplies each element of the likelihood-vector by the corresponding conditional probability. For example, if there are two possible events, E_1 and E_2 , interpreting the data point ϵ would result in a likelihood-vector containing:

$$\begin{aligned} \vec{P}(\epsilon | \mathcal{E}) = \langle &P(E_1 E_2)P(\epsilon | E_1 E_2), \quad P(E_1 \bar{E}_2)P(\epsilon | E_1 \bar{E}_2), \\ &P(\bar{E}_1 E_2)P(\epsilon | \bar{E}_1 E_2), \quad P(\bar{E}_1 \bar{E}_2)P(\epsilon | \bar{E}_1 \bar{E}_2) \rangle \end{aligned}$$

Since $\vec{P}(\epsilon | \mathcal{E})$ contains the products of the conditional probabilities for all currently available data, an agent can use a likelihood-vector from a remote agent to achieve the same result as if the local agent had access to all of the data available to the remote agent.

An agent can choose to send its likelihood-vector to a remote agent by performing a SEND-LIKELIHOOD action. Similarly, an agent can request a remote agent's likelihood-vector by performing a REQUEST-LIKELIHOOD action. An agent chooses to execute either a data point based action or a likelihood-vector based action according to the relative cost of having another agent's partial solution communicated to it versus receiving the raw data and computing a partial solution locally. A detailed explanation follows in Section 5.6.

5.6 DISCUSSION OF THE COST MODEL

On each step, each agent can perform one of five possible actions:

- NOTHING – The agent initiates no action. It will only respond to direct requests.
- SEND-DATA – The agent sends one or more data points in its possession to a remote agent.
- REQUEST-DATA – The agent requests one or more data points from a remote agent. The agent will only receive the data points the remote agent has access to.
- SEND-LIKELIHOOD – The agent sends a likelihood-vector representing all the data in its possession to a remote agent.
- REQUEST-LIKELIHOOD – The agent requests from a remote agent a likelihood-vector representing all the data in its possession.

Each of the actions have a unique cost (i.e. a negative reward) associated with it. Several factors are involved in determining the cost of each action. These include the computation costs, such as the processing power/time required to extract and integrate data points or likelihood vectors, and the communication costs, such as transmission delay.

Carver and Akavipat[9] previously determined the relative costs of data point and likelihood-vector computation and communication. Building on their work, a detailed cost model can be developed.

Assume all agents have access to the complete distributed belief network (i.e. the agents know the complete the grammar), and that each agent is assigned a

single event to interpret. Let:

- $C_q(agent)$ – The cost for an agent to retrieve a single data point from its internal model.
- $C_*(agent)$ – The cost of an agent to perform a multiplication/division operation.
- $C_+(agent)$ – The cost of an agent to perform an addition operation.
- $C_f(link)$ – The fixed cost of utilizing a communication link. This includes overhead to establish a link (e.g. handshaking) and communication latency.
- $C_v(link)$ – The variable communication cost. This value is inversely proportional to the amount of bandwidth available on the link.
- ϵ – The set of data points being acted upon.
- E – The number of events in the grammar.
- $dataname$ – The size of tag indicating which data point is being sent.
- $floatsize$ – The size of the floating point number used to store a single probability.

Using the above terms, the cost of specific operations can be developed. These include:

The cost to extract a set of data points:

$$X_d(agent, \epsilon) = |\epsilon|C_q(agent) \quad (5.2)$$

The cost to transmit a set of data points across a communication link:

$$T_d(link, \epsilon) = C_f(link) + |\epsilon|(1 + dataname)C_v(link) \quad (5.3)$$

The cost to integrate a set of data points into an agent's model:

$$I_d(agent, \epsilon) = |\epsilon|2^E C_*(agent) + 2^{E-1} C_+(agent) \quad (5.4)$$

2^E multiplications are required to integrate each data point's conditional probability table into the agent's likelihood-vector. The 2^{E-1} additions are required to marginalize out all the events, except for the one event the agent is interested in interpreting.

The cost to prepare, transmit, and decode a REQUEST-DATA message that was sent across a communication link:

$$T_{rd}(link, \epsilon_r) = C_f(link) + |\epsilon_r|(dataname)C_v(link) \quad (5.5)$$

The cost for an agent to extract a likelihood-vector:

$$X_l(agent) = 2^E C_*(agent) \quad (5.6)$$

Note that the $2^E C_*(agent)$ term represents the number of divisions required to remove the priors from the likelihood-vector prior to transmission to the remote agent.

The cost to transmit a likelihood-vector across a communication link:

$$T_l(link) = C_f(link) + 2^E (floatsize)C_v(link) \quad (5.7)$$

The cost to integrate a likelihood-vector into an agent's model:

$$I_l(agent) = 2^E C_*(agent) + 2^{E-1} C_+(agent) \quad (5.8)$$

The 2^E multiplications are needed to integrate the received likelihood-vector into the local likelihood-vector. As with $I_d(agent)$, the additions are required to

marginalize out all the events, except for the event the agent is tasked with interpreting.

The cost to prepare, transmit, and decode a REQUEST-LIKELIHOOD message that was sent across a communication link:

$$T_{rl}(link) = C_f(link) + C_v(link) \quad (5.9)$$

It then follows that, the cost of each action is:

$$Cost(NOTHING) = 0 \quad (5.10)$$

$$Cost(SEND-DATA, \epsilon, src, dest) = X_d(src, \epsilon) + T_d(LINK(src, dest), \epsilon) + I_d(dest, \epsilon) \quad (5.11)$$

$$Cost(REQUEST-DATA, \epsilon, src, dest) = T_{rd}(LINK(src, dest), \epsilon) + Cost\left(SEND-DATA, \bigcap(\epsilon, \epsilon_{dest}), dest, src\right) \quad (5.12)$$

$$Cost(SEND-LIKELIHOOD, src, dest) = X_l(src) + T_l(LINK(src, dest)) + I_l(dest) \quad (5.13)$$

$$Cost(REQUEST-LIKELIHOOD, src, dest) = T_{rl}(LINK(src, dest)) + Cost(SEND-LIKELIHOOD, dest, src) \quad (5.14)$$

where *src* and *dest* represent the source agent and the destination agent of the communication messages.

CHAPTER 6

IMPLEMENTATION

6.1 WHY IMPLEMENTATION IS HARD

Care must be taken when implementing a system to solve the DEC-MDP. Each agent has a large number of local states. In turn, each local state potentially has a large number of global states that are consistent with that local state.

Additionally, the evaluation of each state requires multiple conditional probabilities to be evaluated and normalized. Since each state must be examined several times, an efficient implementation is paramount. In this chapter, a computationally efficient state representation and probability calculations are provided. An approximation of the number of partial local states that exist for a specific agent given a specific grammar and data distribution will also be examined.

6.2 CODE METRICS

Implementation was accomplished using Common Lisp and *Franz's Allegro CL Professional 6.2 for Linux*. A previously developed belief network library for use in multiagent DSI simulations was utilized for probability calculations[1]. All other code was original. The original code contained 2,867 non-commented source lines of Common Lisp code in 140 functions in 56 files.

6.3 EVOLUTION OF STATE REPRESENTATION

6.3.1 Storing Available Data Only

In the previous work[3], the data that was currently available to each agent was enough to completely encapsulate an agent's entire state. Since there were only two agents, any data that was not initially distributed to an agent must have been received from the other agent. Coupling this fact with the observation that it is not useful to send a data point to an agent that already has it, each agent was only allowed to send data that it was initially distributed. To prevent an agent from sending the same data point repeatedly, the system "cheated" by allowing each agent to peek at the remote agent's state to determine what data was available to the remote agent. If the remote agent had data that was initially distributed to the local agent, then the local agent knew that the only source of that data was itself, and therefore would not send that data point.

While sufficient for a two agent system, this implementation immediately broke down for systems of three or more agents. For such systems, the individual agents should be allowed to send any data point that is available to it. This feature allows agents to reduce communication in large networks by requesting data from "closer" sources. If an agent is allowed to send any data point it currently has access to, a second problem develops. Peeking at a remote agent's state no longer tells the local agent whether or not the local agent sent a data point to the remote agent, but rather it indicates whether *any* agent has sent the data point to the remote agent. In other words, the agents learn the current data distribution in the entire system, and this is an unreasonable assumption.

6.3.2 Explicit Storing of Communication History

To alleviate the problems outlined in Section 6.3.1, each agent’s communication history was explicitly added to the local states. This expanded the local state to not only contain the data that was currently available to an agent, but also to have an associated list containing which data points the agent sent to which remote agent and another associated list containing which data points were received from which remote agent. These new fields are referred to as an agent’s *sent-history* and *received-history* respectively.

While the addition of communication histories eliminated the need for the agents to peek at each other’s states, it complicated the calculation of $P(s)$ and $P(\hat{G} | s)$. In Section 6.4.3 an approximation for these probabilities is discussed.

6.3.3 Addition of Confidence-Flags

Initially the state definition outlined in Section 6.3.2 was considered sufficient, however this was not the case.

The DEC-MDP used has an inherent partial order to the local states. That is, each state completely contains the immediately preceding state. For example, if an agent received data $D1$ from agent A then all following states must also contain $D1$ and reflect that $D1$ was received from agent A . It was wrongly assumed that this partial ordering would result in an eventual reduction in the number of global states consistent with a given local state, and that this reduction would prevent $V(s)$ from approaching $+\infty$.

Strengthening this assumption was the fact that with enough actions, each agent is guaranteed to to reached a goal state, since eventually every agent would have access to all data in the system. Once this occurs, $C(DATA(s)) = 1$. Also,

Table 6.1: An Example $\hat{\mathcal{G}}^s$

Agent A 's	Agent B 's	Agent C 's	Goal?
$\langle 15\ 0\ 0\ 0\ 3\ 1 \rangle$	$\langle 7\ 0\ 3\ 0\ 0\ 1 \rangle$	$\langle 5\ 0\ 1\ 1\ 0\ 0 \rangle$	No
$\langle 15\ 0\ 0\ 0\ 3\ 1 \rangle$	$\langle 7\ 0\ 3\ 2\ 0\ 1 \rangle$	$\langle 7\ 0\ 1\ 1\ 0\ 2 \rangle$	No

the partial ordering of states guarantees that no cycles can exist in the the graph of state transitions. Furthermore, it was assumed that an agent could only receive a positive reward once during a chain of actions.

This was not the case.

There can exist a local state s where the agent is guaranteed to transition to a terminal global state. However, once the agent performed the appropriate action and transitioned to the local state s' , the number of possible $\hat{\mathcal{G}}$ s actually increases, and more importantly not all the $\hat{\mathcal{G}}^{s'}$ s are terminal.

To illustrate this, consider a system of three agents A , B , and C with four data points $D1$ - $D4$. The data points were initially distributed with agent A having access to $D1$ and $D2$, B having $D2$ and $D3$, and C having $D2$ and $D4$.

Consider agent A as the local agent with $s = \langle 15\ 0\ 0\ 0\ 3\ 1 \rangle^*$. This local state projects to two different $\hat{\mathcal{G}}$ s. These possible global states are shown in Table 6.1.

If agent A sends 8 to agent B , then $s' = \langle 15\ 0\ 8\ 0\ 3\ 1 \rangle$ and the corresponding possible $\hat{\mathcal{G}}$'s are shown in Table 6.2.

Note that the last two $\hat{\mathcal{G}}$'s are impossible given any $\hat{\mathcal{G}}$, since agent B can not send it data on the same turn it received it.

When agent A was in the local state s , it could determine with absolute

*See Appendix B for an explanation of this notation.

Table 6.2: An Example $\hat{\mathcal{G}}^{s'}$

Agent A's	Agent B's	Agent C's	Goal?
$\langle 15\ 0\ 8\ 0\ 3\ 1 \rangle$	$\langle 15\ 0\ 3\ 0\ 8\ 1 \rangle$	$\langle 5\ 0\ 1\ 1\ 0\ 0 \rangle$	Yes
$\langle 15\ 0\ 8\ 0\ 3\ 1 \rangle$	$\langle 15\ 0\ 3\ 2\ 8\ 1 \rangle$	$\langle 7\ 0\ 1\ 1\ 0\ 2 \rangle$	Yes
$\langle 15\ 0\ 8\ 0\ 3\ 1 \rangle$	$\langle 15\ 0\ 3\ 8\ 8\ 1 \rangle$	$\langle 13\ 0\ 1\ 1\ 0\ 8 \rangle$	No
$\langle 15\ 0\ 8\ 0\ 3\ 1 \rangle$	$\langle 15\ 0\ 3\ 10\ 8\ 1 \rangle$	$\langle 15\ 0\ 1\ 1\ 0\ 10 \rangle$	Yes

certainty that sending $D1$ to agent B would result in a goal state. However, once agent A reached s' , it lost the temporal ordering of the communication histories, and therefore became less certain about the actual global state. This would not necessarily be a problem, except for the fact that agent A gained a positive reward for transitioning to a goal state from state s . Since s' in isolation isn't guaranteed to be part of a terminal global state, agent A will receive a second positive reward for transitioning from s' to s'' . This extra positive reward causes the value iteration algorithm to diverge towards $+\infty$ for $V(s)$.

To eliminate this problem, the agents' local states were given an extra field to indicate which agents had reached their confidence threshold. This field, *confidence-flags*, removed the uncertainty of whether the agent was in a terminal global state or not. For simplicity, this field updated automatically for all agents at no cost.

6.4 PERFORMANCE

6.4.1 Naive Lists Versus Generating Functions

The original design of the system naively created large lists containing every element in every set required. The elements of these sets were examined and then

Algorithm 6.1: Naive State Listing

```

for  $\forall i \in \alpha$  do
  Create  $\mathcal{S}_i$ 
  for  $\forall s \in \mathcal{S}_i$  do
    Create  $\mathcal{A}_s$ 
    Create  $\hat{\mathcal{G}}^s$ 
    for  $\forall \hat{G} \in \hat{\mathcal{G}}^s$  do
      for  $\forall a \in \mathcal{A}_s$  do
         $\hat{G}' \leftarrow APPLY(\hat{G}, a, \pi)$ 
         $s' \leftarrow \hat{G}'[i]$ 
      end for
      Free  $\hat{G}$ 
    end for
    Update  $V(s)$  from all  $s'$ 's
    Free  $s$ 
    Free  $\mathcal{A}_s$ 
  end for
end for

```

deallocated in turn. Algorithm 6.1 outlines the general procedure.

While this design was quick to develop and easy to maintain, it suffered from serious performance problems. This designed regularly required large portions of memory to be allocated then deallocated during execution. It was found during testing, that this would often trigger the Lisp environment's garbage collection routines, and thus cause execution to pause temporarily. Garbage collection would occur so often, that it severely impacted time to solution.

To counter this, the system was redesigned to limit the amount of memory required during operation. The set creation and allocation functions were replaced with generators. Generators are parametric functions that return the next item in a list, without the need to have the entire list explicitly stored. This conversion greatly reduced the memory requirements, the frequency of garbage collection, and system's time to solution. Using naive lists, solution time for a 3-4 grammar was

on the order of months; but with the conversion to generators, solution time was reduced to the order of one week.

6.4.2 precalculation of Confidence-Flags

In Section 6.3.3 it was shown that each local state has a field, confidence-flags, that indicates which agents in the system have reached sufficient confidence. Implicitly, confidence-flags are determined by the global state rather than by the local state.

It is important to note that not every possible value of confidence-flags is valid for every state. Obviously a state s_i , where $C(DATA(s_i))$ exceeds the confidence threshold, must include agent i in $CONFIDENCEFLAGS(s_i)$, but what is not so obvious is that each state can imply whether or not certain remote agents have reached their threshold as well. For instance if agent i has access to every data point, and also sent every data point to a remote agent j , then $j \in CONFIDENCEFLAGS(s), \forall s \succeq s_i$. There is no general function that can examine a local state and determine the set of valid values for the confidence-flag field. Therefore, the sets of possible confidence-flags for a local state must be determined through inspection.

Consider a local state with the confidence-flags removed. This is called a *partial local state* and is denoted as \dot{s} . Similarly, a *consistent partial global state* is \dot{G} . Note that each \dot{G} maps to a unique \hat{G} .

Prior to solving the DEC-MDP, all consistent partial global states are generated and the confidence of each each agent determined. The set of confident agents becomes a possible value for the confidence-flag field (also referred to as a confidence-flag pattern) for each agent. This value is then associated with each

Algorithm 6.2: Determine Confidence-Flags Patterns

```

for  $\forall \dot{s} \in \dot{S}$  do
  for  $\forall \dot{G} \in \dot{G}^{\dot{s}}$  do
    curpattern =  $\emptyset$ 
    for  $\forall i \in \alpha$  do
      {Determine which agents are confident}
      if  $C(\dot{G}_i) > \text{confidencethreshold}$  then
        append i to curpattern
      end if
    end for
    for  $\forall i \in \alpha$  do
      {Store the pattern for every partial local state}
      append curpattern to CONFIDENCEPATTERNS( $\dot{G}_i$ )
    end for
  end for
end for

```

agent's partial local state. Once every partial global state has been examined, the set of confidence-flag patterns are stored for each partial local state (see Algorithm 6.2). These patterns are then used to create s from \dot{s} while solving the DEC-MDP.

Predetermining which local states can have which confidence-flag patterns prior to solving the DEC-MDP reduces the total time to solution. precalculation eliminates many unreachable states from consideration. Also it ensures that every local state is examined exactly once during the value update phase of value iteration. If confidence-flags were determined at solution time, then certain confidence-flag patterns may be inadvertently skipped, or more likely certain states visited multiple times.

6.4.3 Precalculation of Normalization Constants

Each local state, s , requires an individual normalization constant for the correct calculation $P(\hat{G} | s)$. The normalization constant for each s must be found

by examining each \hat{G} . Since finding the normalization constants is a very computationally intensive task, precalculation is required.

Prior to solving a DEC-MDP for a given probability grammar and initial data distribution, all valid local states and their consistent global states are examined. Given the fact that each state is a tuple, $P(\hat{G} | s)$ can not be simply be approximated as $P(DATA(\hat{G}) | DATA(s))$, because multiple \hat{G} s and s s have identical $DATA(\hat{G})$ and $DATA(s)$ respectively. Furthermore, while $P(DATA(s))$ is easily found, $P(SENTHISTORY(s))$, $P(RECEIVEDHISTORY(s))$, and $P(CONFIDENCEFLAGS(s))$ are much more difficult since they depend on the joint policies. Therefore, an approximation utilizing only $P(DATA(s))$ and $P(DATA(\hat{G}))$ must be used.

Assuming that all communication histories and confidence-flags are equally likely, the following approximation can be used:

$$P(\hat{G} | s) \approx \frac{1}{k(s)} P(DATA(\hat{G}) | DATA(s)) \quad (6.1)$$

Calculation of the normalization constant $k(s)$ is found in a straight forward manner by summing all $P(DATA(\hat{G}) | DATA(s))$ for all $s \in \mathcal{S}$ and for all $\hat{G} \in \hat{\mathcal{G}}^s$. These probabilities are then summed according to s . This information is then stored in secondary storage for use in solving future DEC-MDPs (see Algorithm 6.3).

When the system begins solving a DEC-MDP, the stored normalization constants are read into memory for use. The current implementation stores all the normalization constants in a hash table. This is memory intensive, and could result in memory exhaustion for DSI systems with a large number of local states.

Algorithm 6.3: Determine Normalization Constants

```

for  $\forall s \in \mathcal{S}$  do
  NORMALIZATIONCONSTANT( $s$ ) = 0
  for  $\forall \hat{G} \in \hat{\mathcal{G}}^s$  do
    for  $\forall i \in \alpha$  do
      NORMALIZATIONCONSTANT( $\hat{G}_i$ )+ =  $P(\text{DATA}(\hat{G}) \mid \text{DATA}(\hat{G}_i))$ 
    end for
  end for
end for

```

A better implementation would use a paging algorithm to read in only the normalization constants required at a specific time.

6.5 DETERMINING THE NUMBER OF LOCAL STATES

When it is said that “a large number” of states exist for each agent, it is useful to know just how large that number actually is. While calculating the exact number of local states for an agent is difficult in general, a reasonable approximation can be found if certain restrictions are placed on the how data can be initially distributed among the agents.

Recall from Section 6.4.3 that the number of confidence-flag patterns for a local state s can not be found without inspection. Therefore, the number of partial local states will be found.

Assume that the agents’ sensors are fault-free (i.e. all data points are guaranteed to be available) and that any agent can send any data point to any other agent, as long as the sending agent does not know that the remote agent already has the data point. To simplify matters, assume that each data point is either initially available only to a single agent or is shared among every agent in system. (e.g. $\langle\langle A : \{D1 D2\}\rangle\langle B : \{D3\}\rangle\langle C : \{D4\}\rangle\rangle$ and

$\langle\langle A : \{D1 D2\}\rangle\langle B : \{D2 D3\}\rangle\langle C : \{D2 D4\}\rangle\rangle$ are valid distributions, but $\langle\langle A : \{D1 D2\}\rangle\langle B : \{D1 D3\}\rangle\langle C : \{D3 D4\}\rangle\rangle$ is not.)

Let us consider each field of a partial local state in turn. The cardinality of each field will be expressed as a function of the state's available data in terms of:

- a – The number of remote agents (i.e. $a \equiv |\alpha| - 1$).
- l – The number of initially local only data points.
- s – The number of initially shared data points.
- r – The number of initially remote only data points.
- i – The number of initially remote data points received by the local agent.

The number of possible data values for a partial state is exactly:

$$NUMDATAVALUES(i) = 2^{l+s+i} \quad (6.2)$$

Since each initially remote data point that has been received must have been sent by a remote agent, and each remote agent is a possible source of each remote data point, it follows that the number of possible ways i remote data points can be received is:

$$NUMRECVHISTS(i) = (2^a - 1)^i \binom{r}{i} \quad (6.3)$$

Each agent can send any initially local data point or any received initially remote data point. In order to count the number of possible sent-histories, each of these cases must be considered separately. There are exactly 2^l possible sent-histories per remote agent when only initially local only data points are considered.

Table 6.3: Values for $f(a, i)$

i	$f(2, i)$	$f(3, i)$
0	0.000000000	0.000000000
1	0.581988897	0.507894882
2	1.333333333	1.273746976
3	2.303314829	2.428571429
4	3.555555556	
5	5.172191382	

Finding the exact number of possible sent-histories for resending initially remote data exactly is a bit problematic since the source of those data points must be considered. Further complicating matters is the fact that each initially remote data point may have been received multiple times, each time from a different source.

A reasonable description of the number of sent-histories is:

$$NUMSENTHISTS(i) = (2^l + AVENUMRESENDABLE(i))^a \quad (6.4)$$

Empirically, $AVENUMRESENDABLE(i)$ was determined to be of the form:

$$AVENUMRESENDABLE(i) = (a - 1)2^{l-1}f(a, i) \quad (6.5)$$

Unfortunately, the nature of $f(a, i)$ was unable to be determined exactly. It appears to be similar to, but not exactly, a power function of with respect to i , with $f(a, 0) = 0$, for all positive integers of a . Actual values for $f(a, i)$ are found in Table 6.3.

Combining Equations 6.2, 6.3, and 6.4, yields:

$$\begin{aligned}
 |\dot{\mathcal{S}}| &= \sum_{i=0}^r \text{NUMDATAVALUES}(i) \text{NUMRECVHISTS}(i) \text{NUMSENTHISTS}(i) \\
 &= \sum_{i=0}^r 2^{l+s+i} (2^a - 1)^i \binom{r}{i} (2^l + (a-1)2^{l-1} f(a, i))^a
 \end{aligned}
 \tag{6.6}$$

It is important to note that for every value of i , the total number of partial local states, and partial global states is a sum of powers of two.

Application of Equation 6.6, along with support from empirical evidence, shows that for a 2-10 grammar with the data evenly distributed between the agents with no data initially shared, there are 248,832 partial local states per agent. A similarly distributed 3-6 grammar has 937,024 partial local states per agent.

6.6 USE OF GENERATORS

In Section 6.4.1, the use of generators for the representation of large sets was discussed. These generators were controlled by the distribution of data among the agents and by a series of configuration flags. The configuration flags controlled aspects of the agents' environment and behavior. Such aspects included whether or not sensors were faulty, whether agents could send data they were not initially distributed, whether there was a limit to the number of data points the agents could send in one message, and whether or not likelihood-vectors could be sent between agents.

Two primary generators, four secondary generators, along with several other ancillary generators were created for solving DEC-MDPs. The two main generators were the *data mask generator* and the *history generator*.

The data mask generator is a bit of a misnomer since it does not just generate data masks, but actually every combination of bits for a given bitmask in a larger bitfield. A data mask generator is created from two bitmasks. The first bitmask indicates which bits in a bitfield should be cycled. The second bitmask indicates which bits in a bitfield, if any, are locked to a certain value. For example, a generator with a cycle bitmask of 5 and an include bitmask of 2, would generate the bitmasks: 2, 3, 6, and 7.

A history generator creates every possible sent-history or received-history (depending on the context) for a given agent and data distribution. The history generator must generate not only every possible value for a given bitmask, but also distributed these values in every combination across a set of agents. To accomplish this, a history generator takes a series of bitmasks that are associated with specific agents. By supplying a history generator the initial data distribution, every valid *RECEIVEDHISTORY(s)* can be created. Conversely, supplying a history generator a series of bitmasks created by logically NANDing *DATA(s)* with the data distribution, every valid *SENTHISTORY(s)* can be created.

By combining two data mask generators along with two history generators, the *partial state generator* can be created. The two data mask generators are linked together to generate every valid *DATA(s)*. The two history generators are combined with the data distribution and *DATA(s)* to create the sent and received histories.

A *basic state generator* is one the simplest generators used within the DEC-MDP solving code. A basic state generator is a wrapper around a partial state generator. For each partial state generated by the internal partial state generator, the basic state generator looks up the list of valid confidence flags for

that partial state. It then combines the flags one at a time to create a true local state. Once all the valid confidence-flags were combined for current partial state, a new partial state is generated.

In order to generate potential global states consistent with a given local state, an *other agents' states generator* is used. The other agents' states generator links either multiple partial state generators or basic state generators together. The choice of which type of generator is used is dependent on whether partial global states or complete global state are desired. For purposes of discussion, basic state generators will be assumed.

One basic state generator is required for each remote agent. Basic state generators have the ability to have the values of certain fields of the generated states to be restricted. It is possible to require that certain fields include certain bitmasks. It is also possible to require that certain fields exclude certain bitmasks. By making later generators dependent on the output of the previous generators, it is possible to generate every consistent global state without any inconsistent global state being generated.

The last generator of note is the *action generator*. As its name suggests, an action generator generates every valid local action that can be performed from a supplied partial local state and data distribution.

6.7 CHECKPOINTING

Since a large amount of computation time is required to solve even small DEC-MDPs, the consequences of system failure (e.g. power failure, system reboot, user error, etc.) during solving becomes a serious concern. In order to prevent having to re-solve the DEC-MDP from the beginning, the algorithm outlined in

Section 3.3 was modified to incorporate periodic checkpointing of agents' local policies and state values. Algorithm 6.4 describes of how this was performed.

6.8 NETWORK SIMULATOR

In addition to the DEC-MDP code, a DSI network simulator was developed to execute the agents' policies under a variety of conditions. Ad hoc and planned network topologies were supported. Point-to-point and broadcast mediums could be simulated. In the case of broadcast ad hoc networks, the physical location of the individual agents was modeled, along with support of physically mobile agents. Through the use of an event queue, communication latency could be modeled. Failure of individual sensors and communication links was supported during the execution of the policies. The goal of the simulator was to allow the future development of policies and recovery methods for faulty DSI networks.

This simulator was not used for any experiments in this work.

Algorithm 6.4: Basic DEC-MDP Solving

```

Require: grammar, data-distribution, event-distribution, confidence-threshold
if confidence-flags exist for grammar, data-distribution, event-distribution, and
confidence-threshold then
  Load confidence-flags
else
  precalculate and save confidence-flags
end if
if normalization constants exist for grammar, data-distribution, event-
distribution, and confidence-threshold then
  Load normalization constants
else
  precalculate and save normalization constants
end if
{Resume from checkpoints}
for  $\forall \text{agent} \in \alpha$  do
  if policy exists for agent, grammar, data-distribution, event-distribution, and
confidence-threshold then
    Load policy
  end if
  if values exists for agent, grammar, data-distribution, event-distribution, and
confidence-threshold then
    Load values {values of  $V(s)$ }
  end if
end for
{Solve DEC-MDP}
numstable := 0 {number of consecutive stable policies found}
agentindex := 0 {current agent}
while numstable <  $|\alpha| - 1$  do
  oldV :=  $V(\text{dummy})$ 
  Value-iterate agentindex's local MDP
  newV :=  $V(\text{dummy})$ 
  if oldV = newV then
    numstable := numstable + 1
  else
    numstable := 0
  end if
  Save policy and values {checkpoint}
  agentindex := (agentindex + 1) mod  $|\alpha|$ 
end while
Save policies and values {save solution}

```

CHAPTER 7

EXAMINATION OF THE SYSTEM

7.1 FORMAL DESCRIPTION OF SYSTEM

A brief review of how a DEC-MDP was applied to the presented DSI problem is in order.

A two level Bayesian belief network is distributed among several agents. Each agent received a subset of the total amount of bottom level data points in the network. Each agent must interpret a subset of the top level events of the belief network. When an agent interprets the data available to it, it must determine how confident it is that its interpretation is the same interpretation that would result if the agent had access to all the data in the system. (See Equation 4.2.) Agents exchange data among themselves in parallel, in order to increase their confidence in their individual interpretations. Each agent determines what data to send or request at each step based on its own local view of the global state. Once all the agents are sufficiently confident in their interpretation, the process terminates.

It is important to note that each agent begins in one of $2^{|\epsilon_i^0|}$ (where ϵ_i^0 the set of data points initially available to agent i) possible start states. Since MDPs require a single start state, a dummy start state, *dummy*, is needed. Agents begin value iteration in this dummy start state. From this state, agents transition stochastically to one of their possible actual start states, at no cost. Once in an actual start state, the agent can then begin to evaluate its actions.

The probability of transitioning to each actual start state s is:

$P(s \mid \text{dummy}) = P(\text{DATA}(s))$, since there is only one possible sent-history and received-history at the start.

It is important to note that multiple terminal states can exist for each agent. Recall from Section 6.3.3 that each agent's local state contains a set of flags indicating which agents are sufficiently confident. These flags are updated automatically, at no cost to the agents.

Section 5.2 illustrated how the classic Bellman equation (Equation 3.1), had to be modified for use with this DEC-MDP. Combined with the information presented in Section 6.4.3, the state transition function $P(s' \mid a, s)$ was approximated as $P(\hat{G} \mid s) \approx \frac{1}{k(s)}P(\text{DATA}(\hat{G}) \mid \text{DATA}(s))$.

In Section 3.3, it was shown that each agent in a DEC-MDP receives a joint reward based on the joint action taken in the current global state and which global state the agents transition into. Let $R(\hat{G}, a, \hat{G}')$ be the joint reward an agent receives when it performs action a in the consistent global state \hat{G} and transitions to the consistent global state \hat{G}' . Since total network utilization needs to be minimized, the joint reward function is defined as:

$$R(\hat{G}, a, \hat{G}') = \begin{cases} -Cost(a) - \sum_{j \in \alpha; j \neq i} Cost(\pi_j(\hat{G}_j)) + K & \text{if } \hat{G} \text{ is not terminal and} \\ & \hat{G}' \text{ is terminal} \\ -Cost(a) - \sum_{j \in \alpha; j \neq i} Cost(\pi_j(\hat{G}_j)) & \text{otherwise} \end{cases} \quad (7.1)$$

where π is the current joint policy, i is the local agent, and K is a positive constant of sufficient magnitude to ensure a positive reward regardless of the

course of actions leading to \hat{G}' . Note that a positive reward is received only for the first transition into a terminal state. This positive reward motivates the agents to perform a sequence of actions with negative immediate rewards in hopes of gaining a positive reward.

K was set to 1,000 for all experiments performed in this work.

In order to limit the size of the policy search tree, agents were only allowed to include at most two raw data points per SEND-DATA or REQUEST-DATA message. The cost of each action was assigned as follows:

- $Cost(\text{NOTHING}) = 0$
- $Cost(\text{SEND-DATA}, data, src, dest) = DISTANCE(src, dest)|data|$
- $Cost(\text{REQUEST-DATA}, data, src, dest) =$
 $DISTANCE(src, dest) + Cost\left(\text{SEND-DATA}, \cap(data, data_{dest}), dest, src\right)$
- $Cost(\text{SEND-LIKELIHOOD}, src, dest) = 4DISTANCE(src, dest)$
- $Cost(\text{REQUEST-LIKELIHOOD}, src, dest) =$
 $DISTANCE(src, dest) + Cost(\text{SEND-LIKELIHOOD}, dest, src)$

where $DISTANCE(agent_0, agent_1)$ is the cost of the lowest cost path available at the time of the action execution between $agent_0$ and $agent_1$. The cost of 4 per hop for SEND-LIKELIHOOD was assigned on an ad hoc basis. The cost was set to be higher than maximum cost of single SEND-DATA action, but low enough to give likelihood-vectors an economic advantage when sending a large amount of data to a specific agent.

The Bellman discount factor, γ , was set to 0.99, since significant weight must be placed on previous negative rewards in order for value iteration to minimize the

cost of the complete communication sequence of the agents. The threshold for value iteration stability, θ , was set to 0.01.

7.2 EXPERIMENTAL PROCEDURE

The 3-4 grammar described in Appendix C was examined with data distributed among the agents as follows: $\langle\langle A : \{D1 D2\}\rangle\langle B : \{D2 D3\}\rangle\langle C : \{D2 D4\}\rangle$. Each agent was responsible for a single event. Each agent’s event corresponded to the agent’s name. (i.e. Agent A was responsible for event A .) This setup resulted in 1,936 partial and 2,649 complete local states per agent. 11,664 consistent global states were examined.

This grammar, event distribution, and data distribution were examined with varying network topologies. These topologies were a fully connected network, and the three possible linear networks where adjacent agents were connected by a single communication link. The individual local MDPs were solved via value iteration. The optimal policies and the grammar are located in Appendix C.

7.3 EXAMINATION OF POLICIES

In all cases studied, the depth of the solution was 1. It is noteworthy that all policies, regardless of topology, found were identical with respect to the final set of reachable states. Optimality of the policies was confirmed through inspection. Examination of the policies revealed that:

- Agent A is always confident.
- Agent B is confident when $D2 = TRUE$. When B is not confident, it requires $D1$ to become confident.

- Agent C is confident when $D4 = FALSE$. When C is not confident, it always requires $D3$. Additionally, when $D2 = FALSE$ and $D4 = TRUE$, C requires $D1$.

Further examination revealed that agent C performs an interesting action when $DATA(s) = \{\bar{D}2 D4\}$. It requests $\{D1 D3\}$ from agent B even though B does not have access to $D1$ at the time of the request. This action is perfectly valid and results in the same cost as if C requested only $D3$. This is because an agent can request up to the maximum payload size of data points, but it is only charged for the number of data points actually received. In this case, the cost of C requesting $\{D1 D3\}$ is the same as if C requested only $D3$.

7.4 EFFECT OF LIKELIHOOD-VECTORS

Unfortunately, due to small number of data points used in the experimental setup no conclusions about the efficacy of likelihood-vectors could be made. In the system studied, there did not exist enough data to make utilizing likelihood-vectors economical. Larger DSI networks need to be studied. The costs of likelihood-vectors should be changed to incorporate $|\mathcal{E}|$, where \mathcal{E} is the set of all events, in order to more accurately model the actual cost of likelihood-vectors.

CHAPTER 8

CONCLUSIONS

8.1 CONCLUSIONS

This work extended the application of DEC-MDPs to DSI systems of an arbitrary number of agents. The development of this extension revealed several complications that were not apparent in the examination of systems of only two agents. One such complication was that there may exist local states that guarantee a transition to a terminal global state, but once that new state is reached, it is not guaranteed to be part of a terminal global state. This uncertainty was resolved by explicitly indicating which local states are part of a terminal global state. Another issue was the complexity of DEC-MDPs with respect to DSI systems. The complexity of the number of local states was found to be $O(r!k^a)$ where r is the number of initially remote data points, and a is the number of remote agents. Issues regarding the performance solving DEC-MDPs were discovered, and overcome. Most importantly, this work showed deficiencies that must be overcome before DEC-MDPs can be applied to real-world DSI systems.

8.2 FUTURE WORK

Final determination of the the average number of remote data points an agent can resend to remote agents would prove valuable as an analysis tool for determining the complexity for specific DSI systems. (See Equation 6.5.)

Clearly the large number of states for even systems with a very small number of agents is of great concern. Effort should be made to determine effective state reduction techniques that provide tractable policy searches while retaining “good enough” solutions. Another course of action could be to develop techniques to solve very large DEC-MDP such as asynchronous value iteration or sweepless dynamic programming.

Another potential future direction for this research would be to investigate how to apply asynchronous actions to the system. Such a system would need to explicitly model both computation time and communication time. Care must be taken to properly define the reward function for the DEC-MDP in order for the results of the asynchronous system to be meaningful.

Both this work and the previous work assumed completely reliable instantaneous communication between the agents. Two obvious extensions to pursue would be to add communication latency and to add faulty communication links. Of the two, the addition of faulty communication links appears to be the more pressing research topic, since any deviation from the training environment could cause the agents’ policies to fail to produce any conclusion at all. It would be interesting to examine how a variety of run-time strategies such as caching, routing table updates, and strategies predicated on the other agents’ optimal policies would provide adequate performance in a faulty environment.

REFERENCES

- [1] N. Carver and V. Lesser, “Domain monotonicity and the performance of local solutions strategies for cdps-based distributed sensor interpretation,” *International Journal of Autonomous Agents and Multi-Agent Systems*, vol. 6, pp. 36–76, 2003.
- [2] V. Lesser and L. Ertan, “Distributed interpretation: A model and experiment,” *IEEE Transactions on Computers*, vol. 29, no. 12, pp. 1144–1163, 1980.
- [3] J. Shen, V. Lesser, and N. Carver, “Minimizing communication cost in a distributed bayesian network using a decentralized mdp,” in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 678–685, ACM Press, 2003.
- [4] R. A. Howard, *Dynamic Programming and Markov Processes*. MIT Press, 1960.
- [5] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.
- [6] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. Cambridge, MA: MIT Press, 1998.
- [7] C. Boutilier, “Sequential optimality and coordination in multiagent systems,” in *Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI-99)*, pp. 478–485, 1999.

- [8] P. Xuan and V. Lesser, “Multi-agent policies: From centralized ones to decentralized ones,” in *AAMAS '02: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1098–1105, ACM Press, 2002.
- [9] N. Carver and R. Akavipat, “Analyzing the efficiency of information merging in distributed sensor interpretation and diagnosis,” in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.

APPENDICES

APPENDIX A — Glossary of Notation

- α Set of all agents in the system
- i An individual agent, usually the local agent; $i \in \alpha$
- j An individual agent, usually a remote agent; $j \in \alpha$
- \mathcal{S} Set of all reachable states; Set of all local states
- s A state; A local state; $s \in \mathcal{S}$
- s' Next state; Next local state; $s \preceq s'$
- \mathcal{S}_i Set of all local states for agent i
- s_i Agent i 's local state; $s_i \in \mathcal{S}_i$
- $\dot{\mathcal{S}}$ The set of all partial local states
- \dot{s} A partial local state; $\dot{s} \in \dot{\mathcal{S}}$
- $\hat{\mathcal{G}}$ The set of all consistent global states
- $\hat{\mathcal{G}}^s$ The set of all global states consistent with the local state s
- \hat{G} A consistent global state; $\hat{G} \in \hat{\mathcal{G}}$
- \hat{G}^s A global state consistent with the local state s ; $\hat{G}^s \in \hat{\mathcal{G}}^s$
- $\dot{\hat{\mathcal{G}}}^s$ The set of all partial global states consistent with the local state s
- $\dot{\hat{G}}$ A partial consistent global state; $\dot{\hat{G}} \in \dot{\hat{\mathcal{G}}}^s$
- \mathcal{A}_s Set of all actions available in state s ; Set of all local actions available in local state s
- a An action; A local action; $a \in \mathcal{A}_s$
- π A policy
- π^* An optimal policy
- π_k Current approximation of π^*
- π_i Agent i 's local policy
- ϵ^* All data in the system

ϵ_i Data currently available to agent i ; $\epsilon_i \subseteq \epsilon^*$

\mathcal{E} The set of all events

E A set of events; $E \subseteq \mathcal{E}$

E_i The set of events to be interpreted by agent i

APPENDIX B — State Notation

In order to compactly write partial local states, the individual fields in a state are represented by a series of bitmasks. Each data point corresponds to an individual bit in these masks. The most significant bit corresponds to data point $D1$, while the n th most significant corresponds to data point Dn .

The first bitmask in a local state represents which data points are currently available to the agent. The second bitmask represents the values of the available data. The next a bitmasks, where a is the number of remote agents in the system, represent which data points were sent by the the local agent to which remote agents. Each individual mask corresponds to a specific remote agent. The order of the masks correspond to the order of the remote agent names in the system. (e.g. Given agents A , B , and C . Agent B 's two masks correspond to agents A and C respectively.) The last a bitmasks represent which data points were received from which remote agent, and are defined similarly.

For example, given a system with agents A , B , and C , and data points $D1 - D4$,

$A : \langle 15\ 4\ 0\ 0\ 3\ 1 \rangle$ would correspond to agent A being in a state containing:

$$\left\{ \begin{array}{ll} DATA : & \{\bar{D}1\ D2\ \bar{D}3\ \bar{D}4\} \\ SENTHISTORY : & \{\langle B : \emptyset \rangle \langle C : \emptyset \rangle\} \\ RECEIVEDHISTORY : & \{\langle B : \{D3\ D4\} \rangle \langle C : \{D4\} \rangle\} \end{array} \right\}$$

This notation is easily extended to handle the addition of confidence flags by appending another bitmask. This final bitmask's least significant bit corresponds to the agent with the earliest unique identifier (i.e. the agent's name) with successive agents corresponding to more significant bits. If an agent has reached its confidence threshold, then its corresponding bit is set to 1, otherwise it is set to 0.

For example, given a system with agents A , B , and C , and data points $D1 - D4$,

$A : \langle 15\ 4\ 0\ 0\ 3\ 1\ 3 \rangle$ would correspond to agent A being in a state containing:

$$\left\{ \begin{array}{ll} \text{DATA :} & \{\bar{D}1\ D2\ \bar{D}3\ \bar{D}4\} \\ \text{SENTHISTORY :} & \{\langle B : \emptyset \rangle \langle C : \emptyset \rangle\} \\ \text{RECEIVEDHISTORY :} & \{\langle B : \{D3\ D4\} \rangle \langle C : \{D4\} \rangle\} \\ \text{CONFIDENCEFLAGS :} & \{A, B\} \end{array} \right\}$$

APPENDIX C — 3-4 Optimal Policies

Optimal policies were discovered for the 3-4 grammar described in Figure C.1 and Table C.1. Three agents were each tasked with interpreting a single event with a confidence of 0.9. Each agent was distributed one unique data point and data point that was shared among all the agents. (See Table C.2 for distribution.) Value iteration with a stability threshold of 0.01 was used to solve each local MDP. Discovery of optimal policies was performed four separate times; once for for a ring topology, and once each for each possible linear topology where the adjacent agents are connected by a single link.

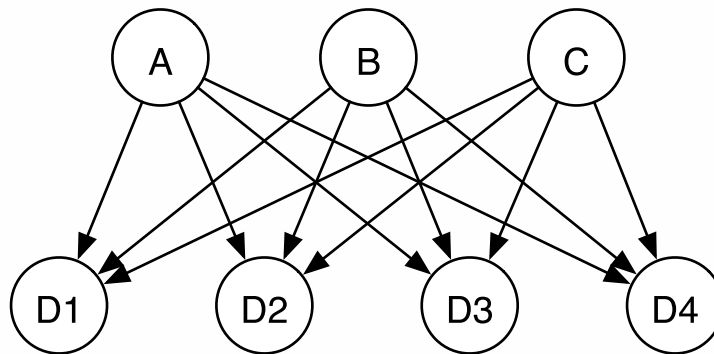


Figure C.1: An Example 3-4 Belief Network

Experiments were performed on a dual 1.2 GHz AMD Athlon with 2 GB of RAM. Each processor ran an independent process of *Franz's Allegro CL Professional 6.2 for Linux*. The Lisp code was compiled with settings of speed 3,

Table C.1: 3-4 Grammar Definition

$P(A) = 0.01$	$P(B) = 0.8$	$P(C) = 0.35$
$P(A D1) = 0.3$	$P(B D1) = 0.85$	$P(C D1) = 0.35$
$P(A D2) = 0.88$	$P(B D2) = 0.72$	$P(C D2) = 0.26$
$P(A D3) = 0.77$	$P(B D3) = 0.56$	$P(C D3) = 0.91$
$P(A D4) = 0.1$	$P(B D4) = 0.11$	$P(C D4) = 0.46$

Table C.2: Data and Event Distribution

Agent	A	B	C
Event	A	B	C
Data	$D1 D2$	$D2 D3$	$D2 D4$

safety 1, debug 1, space 0. With these settings, it took approximately 6 days of computing time to generate a single policy.

For reference, refer to Table C for the key to the bitmasks used in the policy representations.

Table C.3: Key to Bitmasks Used for 3-4 Grammar

2^i	8	4	2	1
Confidence-Flags		C	B	A
Data	$D1$	$D2$	$D3$	$D4$

Each of the following tables represents a policy. The first column contains the initial globally consistent start state. Each subsequent column represents the joint action at time step. A blank box indicates that the agents previously reached a

global terminal state. Note that these tables contain only the states actually visited by the agents in the course of executing their optimal policies.

Table C.4: Fully Connected Topology

$\hat{G} @ t = 0$	Actions @ $t = 1$
$A : \langle 12000005 \rangle$ $B : \langle 60000005 \rangle$ $C : \langle 50000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12000001 \rangle$ $B : \langle 60000001 \rangle$ $C : \langle 51000001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12000005 \rangle$ $B : \langle 62000005 \rangle$ $C : \langle 50000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12000001 \rangle$ $B : \langle 62000001 \rangle$ $C : \langle 51000001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12400007 \rangle$ $B : \langle 64000007 \rangle$ $C : \langle 54000007 \rangle$	
$A : \langle 12400003 \rangle$ $B : \langle 64000003 \rangle$ $C : \langle 55000003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING
$A : \langle 12400007 \rangle$ $B : \langle 66000007 \rangle$ $C : \langle 54000007 \rangle$	
$A : \langle 12400003 \rangle$ $B : \langle 66000003 \rangle$ $C : \langle 55000003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING

continued on next page

Table C.4 continued from previous page

$\hat{G} @ t = 0$	Actions @ $t = 1$
$A : \langle 12800005 \rangle$ $B : \langle 6000005 \rangle$ $C : \langle 5000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12800001 \rangle$ $B : \langle 6000001 \rangle$ $C : \langle 5100001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12800005 \rangle$ $B : \langle 6200005 \rangle$ $C : \langle 5000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12800001 \rangle$ $B : \langle 6200001 \rangle$ $C : \langle 5100001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 121200007 \rangle$ $B : \langle 6400007 \rangle$ $C : \langle 5400007 \rangle$	
$A : \langle 121200003 \rangle$ $B : \langle 6400003 \rangle$ $C : \langle 5500003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING
$A : \langle 121200007 \rangle$ $B : \langle 6600007 \rangle$ $C : \langle 5400007 \rangle$	
$A : \langle 121200003 \rangle$ $B : \langle 6600003 \rangle$ $C : \langle 5500003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING

Table C.5: Line Topology ABC

$\hat{G} @ t = 0$	Actions @ $t = 1$
$A : \langle 12000005 \rangle$ $B : \langle 60000005 \rangle$ $C : \langle 50000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12000001 \rangle$ $B : \langle 60000001 \rangle$ $C : \langle 51000001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12000005 \rangle$ $B : \langle 62000005 \rangle$ $C : \langle 50000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12000001 \rangle$ $B : \langle 62000001 \rangle$ $C : \langle 51000001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12400007 \rangle$ $B : \langle 64000007 \rangle$ $C : \langle 54000007 \rangle$	
$A : \langle 12400003 \rangle$ $B : \langle 64000003 \rangle$ $C : \langle 55000003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING
$A : \langle 12400007 \rangle$ $B : \langle 66000007 \rangle$ $C : \langle 54000007 \rangle$	
$A : \langle 12400003 \rangle$ $B : \langle 66000003 \rangle$ $C : \langle 55000003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING

continued on next page

Table C.5 continued from previous page

$\hat{G} @ t = 0$	Actions @ $t = 1$
$A : \langle 12800005 \rangle$ $B : \langle 6000005 \rangle$ $C : \langle 5000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12800001 \rangle$ $B : \langle 6000001 \rangle$ $C : \langle 5100001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12800005 \rangle$ $B : \langle 6200005 \rangle$ $C : \langle 5000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12800001 \rangle$ $B : \langle 6200001 \rangle$ $C : \langle 5100001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 121200007 \rangle$ $B : \langle 6400007 \rangle$ $C : \langle 5400007 \rangle$	
$A : \langle 121200003 \rangle$ $B : \langle 6400003 \rangle$ $C : \langle 5500003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING
$A : \langle 121200007 \rangle$ $B : \langle 6600007 \rangle$ $C : \langle 5400007 \rangle$	
$A : \langle 121200003 \rangle$ $B : \langle 6600003 \rangle$ $C : \langle 5500003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING

Table C.6: Line Topology ACB

$\hat{G} @ t = 0$	Actions @ $t = 1$
$A : \langle 12000005 \rangle$ $B : \langle 60000005 \rangle$ $C : \langle 50000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12000001 \rangle$ $B : \langle 60000001 \rangle$ $C : \langle 51000001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12000005 \rangle$ $B : \langle 62000005 \rangle$ $C : \langle 50000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12000001 \rangle$ $B : \langle 62000001 \rangle$ $C : \langle 51000001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12400007 \rangle$ $B : \langle 64000007 \rangle$ $C : \langle 54000007 \rangle$	
$A : \langle 12400003 \rangle$ $B : \langle 64000003 \rangle$ $C : \langle 55000003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING
$A : \langle 12400007 \rangle$ $B : \langle 66000007 \rangle$ $C : \langle 54000007 \rangle$	
$A : \langle 12400003 \rangle$ $B : \langle 66000003 \rangle$ $C : \langle 55000003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING

continued on next page

Table C.6 continued from previous page

$\hat{G} @ t = 0$	Actions @ $t = 1$
$A : \langle 12800005 \rangle$ $B : \langle 6000005 \rangle$ $C : \langle 5000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12800001 \rangle$ $B : \langle 6000001 \rangle$ $C : \langle 5100001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12800005 \rangle$ $B : \langle 6200005 \rangle$ $C : \langle 5000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12800001 \rangle$ $B : \langle 6200001 \rangle$ $C : \langle 5100001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 121200007 \rangle$ $B : \langle 6400007 \rangle$ $C : \langle 5400007 \rangle$	
$A : \langle 121200003 \rangle$ $B : \langle 6400003 \rangle$ $C : \langle 5500003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING
$A : \langle 121200007 \rangle$ $B : \langle 6600007 \rangle$ $C : \langle 5400007 \rangle$	
$A : \langle 121200003 \rangle$ $B : \langle 6600003 \rangle$ $C : \langle 5500003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING

Table C.7: Line Topology BAC

$\hat{G} @ t = 0$	Actions @ $t = 1$
$A : \langle 12000005 \rangle$ $B : \langle 60000005 \rangle$ $C : \langle 50000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12000001 \rangle$ $B : \langle 60000001 \rangle$ $C : \langle 51000001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12000005 \rangle$ $B : \langle 62000005 \rangle$ $C : \langle 50000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12000001 \rangle$ $B : \langle 62000001 \rangle$ $C : \langle 51000001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12400007 \rangle$ $B : \langle 64000007 \rangle$ $C : \langle 54000007 \rangle$	
$A : \langle 12400003 \rangle$ $B : \langle 64000003 \rangle$ $C : \langle 55000003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING
$A : \langle 12400007 \rangle$ $B : \langle 66000007 \rangle$ $C : \langle 54000007 \rangle$	
$A : \langle 12400003 \rangle$ $B : \langle 66000003 \rangle$ $C : \langle 55000003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING

continued on next page

Table C.7 continued from previous page

$\hat{G} @ t = 0$	Actions @ $t = 1$
$A : \langle 12800005 \rangle$ $B : \langle 6000005 \rangle$ $C : \langle 5000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12800001 \rangle$ $B : \langle 6000001 \rangle$ $C : \langle 5100001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 12800005 \rangle$ $B : \langle 6200005 \rangle$ $C : \langle 5000005 \rangle$	$A \rightarrow B$ SEND-DATA 8 NOTHING NOTHING
$A : \langle 12800001 \rangle$ $B : \langle 6200001 \rangle$ $C : \langle 5100001 \rangle$	$A \rightarrow C$ SEND-DATA 8 $B \rightarrow A$ REQUEST-DATA 8 $C \rightarrow B$ REQUEST-DATA 10
$A : \langle 121200007 \rangle$ $B : \langle 6400007 \rangle$ $C : \langle 5400007 \rangle$	
$A : \langle 121200003 \rangle$ $B : \langle 6400003 \rangle$ $C : \langle 5500003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING
$A : \langle 121200007 \rangle$ $B : \langle 6600007 \rangle$ $C : \langle 5400007 \rangle$	
$A : \langle 121200003 \rangle$ $B : \langle 6600003 \rangle$ $C : \langle 5500003 \rangle$	NOTHING $B \rightarrow C$ SEND-DATA 2 NOTHING

VITA

Graduate School
Southern Illinois University

Jonathan Koren

Date of Birth: August 15, 1976

668 State Highway 149, Royalton, Illinois 62983

Southern Illinois University at Carbondale
Bachelor of Science, Computer Science, December 1998

Thesis Title:

Minimizing Communication Cost in an N-Agent Distributed Bayesian Network
by Using a Decentralized MDP

Major Professor: Dr. Norman F. Carver III