2-2007

Towards efficient planning for real world partially observable domains

Pradeep R. VARAKANTHAM
Singapore Management University, pradeepv@smu.edu.sg

Citation

VARAKANTHAM, Pradeep R.. Towards efficient planning for real world partially observable domains. (2007). 1-110. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/948

Towards efficient planning for real world partially observable domains

by

Pradeep Varakantham

A Thesis Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Science)

February 2007

Dedication

This dissertation is dedicated to my parents and my brother.

Acknowledgements

I would like to thank all the people who have helped me complete my thesis.

First and foremost, I would like to thank my advisor, Milind Tambe for his attention, guidance, insight and support at every step of this thesis. Not just on my thesis, his advise has contributed in my development as an individual and academician.

I wish to thank Manuela Veloso, Sven Koenig, Stacy Marsella and Fernando Ordonez for being on my thesis committee. Their valuable comments were instrumental in structuring my dissertation. Manuela Veloso through her insightful comments helped me understand the pragmatic issues with the contributions. Sven Koenig always asked the right questions and was constructive in his criticism. Stacy Marsella provided valuable feedback and pointed to similar contributions, in my discussions with him. Fernando Ordonez provided an outsider's view on the contributions made in this thesis.

I am thankful to Makoto Yokoo for being an excellent collaborator, who was involved in building a significant chunk of this thesis and also for providing crucial feedback that aided me in shaping the thesis. I am grateful to Rajiv Maheswaran for guiding me during the early phases of my PhD and for the numerous stimulating discussions that have helped me significantly in this thesis. I sincerely thank Ranjit Nair for being an excellent colleague and co-author, and also for providing sound advise.

I am grateful to all the members of the TEAMCORE research group for being an amiable bunch of friends and collaborators. Praveen Paruchuri, Nathan Schurr, Jonathan Pearce, Emma Bowring, Janusz Marecki, Tapana Gupta and Zvi Topol have always been very helpful and supportive.

Lastly and most importantly , I would like to express my gratitude to my family. In particular, I would like to thank my parents and brother for believing in me and pushing me to get a doctorate degree.

Abstract

My research goal is to build large-scale intelligent systems (both single- and multi-agent) that reason with uncertainty in complex, real-world environments. I foresee an integration of such systems in many critical facets of human life ranging from intelligent assistants in hospitals to offices, from rescue agents in large scale disaster response to sensor agents tracking weather phenomena in earth observing sensor webs, and others. In my thesis, I have taken steps towards achieving this goal in the context of systems that operate in partially observable domains that also have transitional (non-deterministic outcomes to actions) uncertainty. Given this uncertainty, Partially Observable Markov Decision Problems (POMDPs) and Distributed POMDPs present themselves as natural choices for modeling these domains.

Unfortunately, the significant computational complexity involved in solving POMDPs (PSPACE-Complete) and Distributed POMDPs (NEXP-Complete) is a key obstacle. Due to this significant computational complexity, existing approaches that provide exact solutions do not scale, while approximate solutions do not provide any usable guarantees on quality. My thesis addresses these issues using the following key ideas: The first is exploiting structure in the domain. Utilizing the structure present in the dynamics of the domain or the interactions between the agents allows improved efficiency without sacrificing on the quality of the solution. The second is direct approximation in the value space. This allows for calculated approximations at each step of the algorithm, which in turn allows us to provide usable quality guarantees; such quality guarantees may be specified in advance. In contrast, the existing approaches approximate in the belief space leading to an approximation in the value space (indirect approximation in value space), thus making it difficult to compute functional bounds on approximations. In fact, these key ideas allow for the efficient computation of optimal and quality bounded solutions to complex, large-scale problems, that were not in the purview of existing algorithms.

Contents

List Of Figures

List Of Tables

Chapter 1

Introduction

Recent years have seen an exciting growth of applications (deployed and emerging) of agents and multiagent systems in many facets of our daily lives. These applications mandate that agents act in complex, uncertain domains, and they range from intelligent assistants in hospitals to office [Scerri et al., 2002; Leong and Cao, 1998; Magni et al., 1998], to rescue agents in large scale disaster response [Kitano et al., 1999], to sensor agents tracking weather phenomena in earth observing sensor webs [Lesser et al., 2003], and others. However, for a successful transition of these applications to real world domains, the underlying uncertainty has to be taken into account.

Partially Observable Markov Decision Problems (POMDPs) and Distributed Partially Observable Markov Decision Problems (Distributed POMDPs) are becoming popular approaches for modeling decision problems for agents and teams of agents operating in real world uncertain environments [Pollack et al., 2003a; Simmons and Koenig, 1995; Bowling and Veloso, 2002; Roth et al., 2005; Varakantham et al., 2005; Nair et al., 2003c, 2005]. This is owing to the ability of these models to capture uncertainty present in real world environments: unknown initial configuration of the domain, non deterministic outcomes to actions and noise in the sensory perception. Furthermore, these models can also capture the utilities associated with different outcomes due to their ability to reason with costs and rewards.

Unfortunately, the computational cost of optimal policy generation in POMDPs (PSPACE-Complete) and distributed POMDPs (NEXP-Complete) [Bernstein et al., 2000] is prohibitive, requiring increasingly efficient algorithms to solve decision problems in large-scale domains. Furthermore, many domains [Kitano et al., 1999; Pollack et al., 2003a; Scerri et al., 2002; Leong and Cao, 1998; Magni et al., 1998] require that the efficiency gains do not cause significant losses in optimality of the policy generated; indeed, it is important for the algorithms to bound any loss in quality. Thus the key challenge is to provide efficiency gains in POMDPs and Distributed POMDPs with bounded quality loss.

In single agent POMDPs, there has been significant progress made with respect to efficiency, using two types of solution techniques: exact [Feng and Zilberstein, 2005; Cassandra et al., 1997b] and approximate [Pineau et al., 2003; Smith and Simmons, 2005]. Exact techniques provide optimal solutions, avoiding the problems with respect to quality bounds, however suffer from considerable computational inefficiency. On the other hand, approximate techniques provide efficient techniques that scale to larger problems but at the expense of quality bounds. Turning now to distributed POMDPs, researchers have pursued two different approaches here as well: exact [Nair et al., 2003a; Hansen et al., 2004b] and approximate [Becker et al., 2003; Nair et al., 2003a; Peshkin et al., 2000a; Becker et al., 2004]. Unfortunately, the exact approaches have so far been limited to two agents, with comparatively little attention focussed on them. On the other hand, approximate approaches either limit agent interactions (transition independence) [Becker et al., 2003] or approximate observability of the local state [Becker et al., 2004] or find local optimal solutions [Nair et al., 2003a; Peshkin et al., 2000a]. Though these approaches for distributed POMDPs provide improvement in performance, they still suffer from similar drawbacks: (a) computational inefficiency given large numbers of agents (b) lack of bounds on solution quality.

My thesis takes steps to address these problems of efficiency, while providing guarantees on solution quality. To that end, I have proposed two key solution mechanisms:

1. Exploiting structure inherent in the domain: I have investigated two types of structure, that often arise in real world domains:

   (a) Physical limitations/Progress structure in the process being modeled (structure in dynamics): These techniques restrict policy computation to the belief space polytope that remains reachable given the physical limitations of a domain. One example of a physical limitation in a process is from a personal assistant domain where an agent assists a user: if the user is at a location, it is highly improbable for him/her to be 5 miles away in the next 5 seconds. I introduce new techniques, particularly one based on applying Lagrangian methods to compute a bounded belief space support in polynomial time. These techniques are complementary to many existing exact and approximate POMDP policy generation algorithms. In fact, these exact techniques provide an order of magnitude speedup over the fastest existing exact solvers.

   (b) Structure in the interactions of agents: Techniques for distributed POMDPs have traditionally considered agents in a multi-agent environment with full interactivity i.e. all agents interact with all other agents. However, in domains like sensor networks, each node(agent) interacts only with the nodes that are adjacent to it in the network. Distributed Constraint Optimization (DCOP) is a model for coordination, where the solution techniques rely on exploiting these kinds of limited interaction structures [Modi et al., 2003a; Petcu and Faltings, 2005; Maheswaran et al., 2004], however with an inability to handle uncertainty. On the other hand, distributed POMDP techniques handle uncertainty without exploiting structure in the interactions. I have combined these two approaches to propose a new model, Network Distributed POMDPs (ND-POMDP). In this thesis, I have provided solution techniques for these distributed POMDPs, that build over exact and locally optimal DCOP approaches, namely DPOP (Distributed Pseuodo-tree OPtimization), DBA (Distributed Breakout Algorithm), and DSA (Distributed Stochastic Algorithm). Furthermore, I have also provided a heuristic search technique called SPIDER, that exploit the interaction structure. All these algorithms provide a significant improvement in performance of the policy computation for a team of agents. Furthermore, SPIDER provides this efficiency while providing quality guarantees on the solution.

2. Direct approximation in the value space: Existing approaches [Pineau et al., 2003; Zhou and Hansen, 2001; Montemerlo et al., 2004] to approximation in POMDPs and Distributed POMDPs have focussed on sampling the belief space and approximating the optimal value function with the value computed for the sampled belief space (indirect value approximation). The key novelty in my technique is to directly approximate in the value space, so that every approximation phase has a bounded (pre-computable) quality loss. I have illustrated the utility of this technique in the context of both POMDPs and Distributed POMDPs. In single agent POMDPs, this idea translates to efficiently computing policies that are at most ϵ (approximation parameter) away from the optimal value function. In distributed POMDPs, the execution of the idea translates to computing policies that are at most ϵ (approximation parameter) away from a (tight) upper bound on the optimal value function (computed by approximating the Distributed POMDP as a centralized Markov Decision Problem or MDP). Both these techniques were shown to be faster than best known existing solvers, while providing guarantees on solution quality *missing in previous work.*

The rest of this document is organized as follows: Chapter 2 contains a background of the domains, models and algorithms used in this thesis. Chapter 3 explains the structure exploitation of the dynamics of a domain in single agent POMDPs. Direct value approximation for POMDPs is presented in Chapter 4, while Chapter 5 provides the experimental results for the single agent POMDP techniques. Chapter 6 elucidates the exploitation of network structure, while Chapter 7 contains an exposition for the direct value approximation technique for in distributed POMDPs.

Chapter 8 describes the structure exploitation of the dynamics for distributed POMDPs. Related work is presented in detail in Chapter 9, while the conclusion is presented in Chapter 10.

Chapter 2

Background

This chapter provides a brief background on the experimental domains, the models employed, and existing algorithms to solve the models.

2.1 Domains

To illustrate the applicability of my techniques, I have considered different types of domains. These are personal assistant agents (Section 2.1.1), sensor networks (Section 2.1.2), an illustrative tiger problem (Section 2.1.3) and other problems from literature.

2.1.1 Personal Assistant Agents (PAA)

Recent research has focused on individual agents or agent teams that assist humans in offices, at home, in medical care and in many other spheres of daily activities [Schreckenghost et al., 2002; Pollack et al., 2003b; htt, 2003; Scerri et al., 2002; Leong and Cao, 1998; Magni et al., 1998]. Such agents must often monitor the evolution of a process or state over time (including that of the human, the agents are deployed to assist) and make periodic decisions based on such monitoring. For example, in office environments, agent assistants may monitor the location of users in transit and make decisions such as delaying, canceling meetings or asking users for more information [Scerri et al., 2002]. Similarly, in assisting with caring for the elderly [Pollack et al., 2003b] and therapy planning [Leong and Cao, 1998; Magni et al., 1998], agents may monitor users' states/plans and make periodic decisions such as sending reminders. Henceforth in this document, I refer to such agents as PAAs. Owing to the great promise of PAAs, addressing decision making in these agents represents a critical problem.

Unfortunately, PAAs must monitor and make decisions despite significant uncertainty in their observations (as the true state of the world may not be known explicitly) and actions (outcome of agents' actions may be non-deterministic). Furthermore, actions have costs, e.g., delaying a meeting has repercussions on attendees. Researchers have turned to decision-theoretic frameworks to reason about costs and benefits under uncertainty. However, this research has mostly focused on Markov decision processes (MDPs) [Scerri et al., 2002; Leong and Cao, 1998; Magni et al., 1998], ignoring the observational uncertainty in these domains, and thus potentially degrading agent performance significantly and/or requiring unrealistic assumptions about PAAs' observational abilities. POMDPs address such uncertainty, but the long run-times for generating optimal policies for POMDPs remains a significant hurdle in their use in PAAs.

A key PAA domain that we present here is the task management problem (TMP). This is a key problem within CALO (Cognitive Agent that Learns and Organises), a software personal assistant project [htt, 2003]. In this domain, a set of dependent tasks is to be performed by a group of users before a deadline, e.g. a group of users are working on getting a paper done before the deadline. Agents monitor the progress of their users, and help in finishing the tasks before a deadline by doing reallocations at certain points in time. Furthermore, agents also make a decision on whom to reallocate a task, thus they must monitor status of other users who are capable of doing it.

This problem is complicated as the agents need to reason about reallocation in the presence of transitional and observational uncertainty. Transitional uncertainty arises because there is non-determinism in the way users make progress. For example, a user might finish two units of progress in one time unit, or might not do anything in one time unit. On the other hand, observational uncertainty is present because of two reasons:

1. Acquiring exact progress made on a task is difficult.

2. Knowing whether other (capable) users are free or not is difficult.

Agents can ask their users about the progress made, when there is a lot of uncertainty in the state. When the user responds, agent knows the exact progress of the user on the task. This however comes at a cost of disturbing the user and occurs only with a certain probability as users may or may not respond to agent's request. Thus each agent needs to find a strategy that guides its operation at each time step, till the deadline. This strategy would consist of executing either a "wait", or "ask user", or "reallocate" task to other users, at each time step. These reallocation points are when a user is not making sufficient progress on the tasks. Agents decide on when and whom to reallocate, based on the observations they obtain about the progress made by the user on the task. These observations however are noisy, because it is difficult to acquire the exact progress made by the user on a task.

POMDPs provide a framework to analyze and obtain policies in TMP type domains. In a TMP, a POMDP policy can take into account the possibly uneven progress of different users, e.g., some users may make most of their progress well before the deadline, while others do the bulk of their work closer to the deadline. In contrast, an instantaneous decision-maker cannot take into account such dynamics of progress. For instance, consider a TMP scenario where there are five levels of task progress $x \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$ and five decision points before the deadline $t \in \{1, 2, 3, 4, 5\}$. Observations are five levels of task progress $\{0.00, 0.25, 0.50, 0.75, 1.00\}$ and time moves forward in single steps, i.e. $T([x, t], a, [\tilde{x}, \tilde{t}]) = 0$ if $\tilde{t} \neq t + 1$. While transition uncertainty implies irregular task progress, observation uncertainty implies agent may observe progress x as for instance x or $x + 0.25$ (unless $x = 1.00$). Despite this uncertainty in observing task progress, a PAA needs to choose among waiting (W), asking user for info (A), or reallocate task to other users(R). A POMDP policy tree that takes into account both the uncertainty in observations and future costs of decisions, and maps observations to actions, for this scenario is shown in Figure 2.1 (nodes=actions, links=observations). In more complex domains with additional actions such as delaying deadlines, cascading effects of actions will require even more careful planning afforded by POMDP policy generation.

One other key characteristic of TMP is that the human can restrict the usage of certain actions in certain states, thus associating a reward of negative infinity with certain actions. Additionally, a POMDP algorithm solving TMP problems needs to have the following characteristics: (a) A plan for a pre-specified quality guarantee. (b) A quality bound valid for all possible starting belief points. (c) A policy that can be computed efficiently.

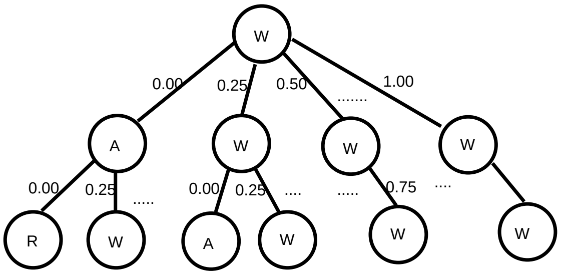


Figure 2.1: Partial Sample Policy for a TMP

2.1.2 Distributed Sensor Network

In this section, I provide an illustrative problem within the distributed sensor net domain, motivated by the real-world challenge in [Lesser et al., 2003][1]. This is an important problem, because of the wide applicability of sensor networks [Chintalapudi et al., 2005; S. Funiak and Sukthankar,

[1] For simplicity, this scenario focuses on binary interactions. However, the algorithms introduced in this thesis allow n-ary interactions.

2006] in many real world problems. One key example is the tracking of weather phenomena in earth observing sensor webs. This is thus a pre-existing domain, one that has been attacked by other multiagent researchers. One key aspect of this domain is the locality of interactions among multiple agents and hence DCOP is a good formalism to model this domain. Owing to the ability of DCOP algorithms to exploit locality in interactions, the algorithms developed are based on DCOP algorithms.

Here, each sensor node can scan in one of four directions — North, South, East or West (see Figure 2.2). To track a target and obtain associated reward, two sensors with overlapping scanning areas must coordinate by scanning the same area simultaneously. Thus, the target position constitutes a world state, and each sensor has four actions: scan-north, scan-south, scan-east, scan-west. We assume that there are two independent targets and that each target's movement is uncertain and unaffected by the sensor agents. Based on the area it is scanning, each sensor receives observations that can have false positives and false negatives. Each agent incurs a cost for scanning whether the target is present or not, but no cost if it turns off.

As seen in this domain, each sensor interacts with only a limited number of neighboring sensors. For instance, sensors 1 and 3's scanning areas do not overlap, and cannot effect each other except indirectly via sensor 2. The sensors' observations and transitions are independent of each other's actions. Existing distributed POMDP algorithms are inefficient for such a domain because they are not geared to exploit locality of interaction. Thus, they will have to consider all possible action choices of even non-interacting agents in trying to solve the distributed POMDP. Distributed constraint satisfaction (DisCSP) [Mailler and Lesser, 2004b; Modi et al., 2001] and distributed constraint optimization (DCOP) [Mailler and Lesser, 2004a] have been applied to sensor nets but they cannot capture the uncertainty in the domain.

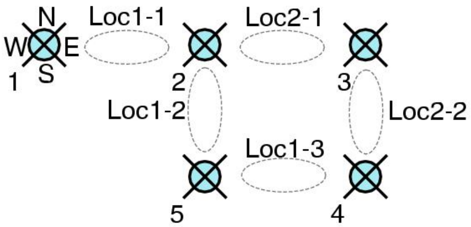


Figure 2.2: Sensor net scenario: If present, target1 is in Loc1-1, Loc1-2 or Loc1-3, and target2 is in Loc2-1 or Loc2-2.

2.1.3 Illustrative Domain: Tiger Problem

This multiagent tiger problem from [Nair et al., 2003a] is an illustrative problem from the literature. Two agents are in a corridor facing two doors "left" and "right". Behind one door lies a hungry tiger, and behind the other lies a reward. The set of states, S, is {SL, SR}, where SL indicates tiger behind the left door, and SR indicates tiger behind right door. The agents can jointly or individually open either door. In addition, the agents can independently listen for the presence of the tiger. Thus, the set of actions, A1 = A2 = {'OpenLeft', 'OpenRight', 'Listen'}. The transition function, P specifies that the problem is reset whenever an agent opens one of the doors. However, if both agents listen, the state remains unchanged. After every action each agent receives an observation about the new state. The observation functions are identical and will return either TL or TR with different probabilities depending on the joint action taken and the resulting world state. For example, if both agents listen and the tiger is behind the left door (state is SL), each agent independently receives the observation TL with probability 0.85 and TR with probability 0.15. For more details on this domain, refer to [Nair et al., 2003a].

2.1.4 Others

For single agent POMDPs, I have used the following domains: Tiger grid, Hallway, Hallway2, Aircraft, Tag and Scotland yard. Of these problems Tiger-grid, Hallway, Hallway2, Aircraft, Tag are benchmark problems from the literature [Pineau et al., 2003; Smith and Simmons, 2005]. Hallway, Hallway2, Aircraft and Tag are path planning problems from robotics. While Scotland yard is a problem derived from the scotland yard game, with 216 states, 16 actions and 6 observations (See http://en.wikipedia.org/wiki/Scotland_Yard_(board_game)).

2.2 Models

I will assume readers are familiar with POMDPs and Distributed POMDPs; however, I will briefly describe POMDPs and Distributed POMDPs to introduce my terminology and notation.

2.2.1 Single Agent POMDPs

A POMDP can be represented using the tuple $\{S, A, T, O, \Omega, R\}$, where S is a finite set of states; A is a finite set of actions; Ω is a finite set of observations; $T(s, a, s')$ provides the probability of transitioning from state s to s' when taking action a; $O(s', a, o)$ is probability of observing o after taking an action a and reaching s'; $R(s, a)$ is the reward function. A belief state b, is a probability distribution over the set of states S. A value function over a belief state is defined as: $V(b) = \max_{a \in A} \{R(b,a) + \beta \, \Sigma_{b' \in B} T(b,a,b')V(b')\}$.

2.2.2 Distributed POMDPs: MTDP

The distributed POMDP model that we base our work on is MTDP [Pynadath and Tambe, 2002], however other models [Bernstein et al., 2000] could also be used. These distributed POMDP models are more than just two single agent POMDPs working independently. In particular, given a team of n agents, an MTDP [Pynadath and Tambe, 2002] is defined as a tuple: $\langle S, A, P, \Omega, O, R\rangle$. S is a finite set of world states $\{s_1, \ldots, s_m\}$. $A = \times_{1 \leq i \leq n} A_i$, where $A_1, \ldots, A_n$, are the sets of action for agents 1 to n. A joint action is represented as $\langle a_1, \ldots, a_n\rangle$. $P(s_i, \langle a_1, \ldots, a_n\rangle, s_f)$, the transition function, represents the probability that the current state is s_f, if the previous state is s_i and the previous joint action is $\langle a_1, \ldots, a_n\rangle$. $\Omega = \times_{1 \leq i \leq n} \Omega_i$ is the set of joint observations where Ω_i is the set of observations for agents i. $O(s, \langle a_1, \ldots, a_n\rangle, \omega)$, the observation function, represents the probability of joint observation $\omega \in \Omega$, if the current state is s and the previous joint action is $\langle a_1, \ldots, a_n\rangle$. We assume that observations of each agent is independent of each other's observations. Given the world state and joint actions, the observation function can be expressed as $O(s, \langle a_1, \ldots, a_n\rangle, \omega) = O_1(s, \langle a_1, \ldots, a_n\rangle, \omega_1) \cdot \ldots \cdot O_n(s, \langle a_1, \ldots, a_n\rangle, \omega_n)$. The agents receive a single immediate joint reward $R(s, \langle a_1, \ldots, a_n\rangle)$ which is shared equally.

Each agent i chooses its actions based on its local *policy*, π_i, which is a mapping of its observation history to actions. Thus, at time t, agent i will perform action $\pi_i(\vec{\omega}_i^t)$ where $\vec{\omega}_i^t = \omega_i^1, \ldots, \omega_i^t$. $\pi = \langle \pi_1, \ldots, \pi_n\rangle$ refers to the joint policy of the team of agents. In this model, execution is distributed but planning is centralized.

2.3 Existing Algorithms

In this section, I present some of the existing algorithms for solving POMDPs and Distributed POMDPs that would be referred to in detail in later parts of this document.

2.3.1 Exact Algorithms for POMDPs

Currently, the most efficient exact algorithms for POMDPs are value iteration algorithms, specifically GIP [Cassandra et al., 1997b] and RBIP [Feng and Zilberstein, 2004b, 2005]. These are dynamic programming algorithms, which perform two steps at each iteration: (a) generating all potential policies and (b) pruning dominated policies to obtain a a minimal set of dominant policies called the parsimonious set. Figure 2.3.1 provides a pictorial depiction of these two steps, where each line (in the graphs) represents the value vector corresponding to a policy. The first figure shows the dominated policies on the bottom of the graph (circled). These dominated policies are computed by using linear programming.

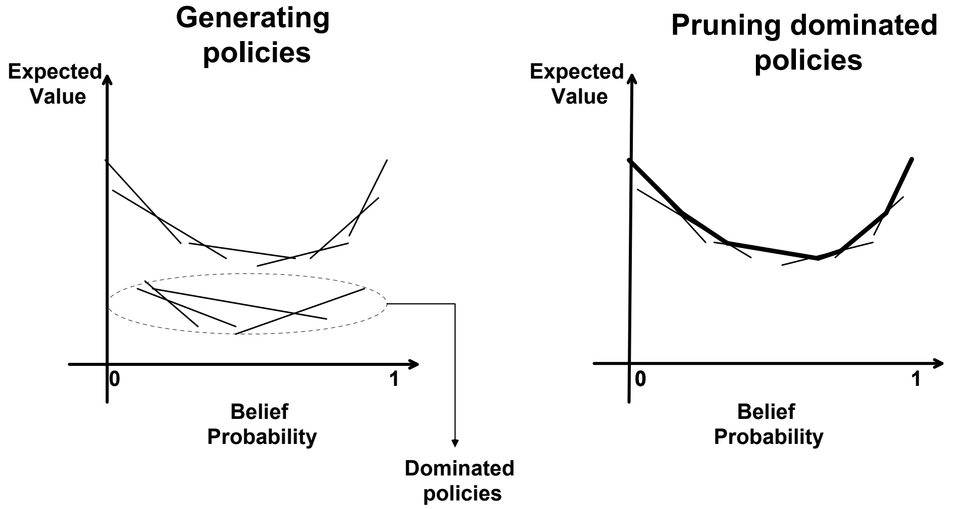


Figure 2.3: Two steps of value iteration in GIP and RBIP

Given a parsimonious set (represented as value vectors corresponding to policies) at time t, $\mathcal{V}_t$, we generate the parsimonious set at time $t-1$, $\mathcal{V}_{t-1}$ as follows (notation similar to the one used in [Cassandra et al., 1997b] and [Feng and Zilberstein, 2004b]):

1. $\left\{v^{a,o,i}_{t-1}(s) = r(s,a)/|\Omega| + \beta\, \Sigma_{s' \in S} Pr(o, s'|s, a) v^i_t(s')\right\} =: \hat{\mathcal{V}}^{a,o}_{t-1}$ where $v^i_t \in \mathcal{V}_t$.
2. $\mathcal{V}^{a,o}_{t-1} = PRUNE(\hat{\mathcal{V}}^{a,o}_{t-1})$
3. $\mathcal{V}^{a}_{t-1} = PRUNE(\cdots(PRUNE(\mathcal{V}^{a,o_1}_{t-1} \oplus \mathcal{V}^{a,o_2}_{t-1})\cdots \oplus \mathcal{V}^{a,o_{|\Omega|}}_{t-1})$
4. $\mathcal{V}_{t-1} = PRUNE(\bigcup_{a\in A} \mathcal{V}^{a}_{t-1})$

Each $PRUNE$ call executes a linear program (LP) which is recognized as a computationally expensive phase in the generation of parsimonious sets [Cassandra et al., 1997b; Feng and Zilberstein, 2004b]. Our approach effectively translates into obtaining speedups by reducing the quantity of these calls.

2.3.2 Approximate Algorithms for POMDPs

Here we concentrate on two of the most efficient approximation algorithms that provide quality bounds, Point-Based Value Iteration (PBVI) [Pineau et al., 2003] and Heuristic Search Value Iteration (HSVI) [Smith and Simmons, 2005]. In these algorithms, a policy computed for a sampled set of belief points is extrapolated to the entire belief space. PBVI/HSVI are anytime algorithms, where the set of belief points being planned for is expanded over time. The expansion ensures that the belief points are uniformly distributed over the entire belief space. The heuristics used to accomplish this belief set expansion differentiate PBVI and HSVI. However, to obtain this set of belief points, both algorithms require specification of a starting belief point.

Since our approach (for solving POMDPs approximately) focusses on quality bounds, we will discuss the quality bounds in PBVI/HSVI. For PBVI, this bound is provided by:

$$(R_{max} - R_{min}) * \epsilon_b / (1-\gamma)^2,$$

where R_{max} and R_{min} represent the maximimum and minimum possible reward for any action in any state and ϵ_b = $max_{b' \in \Delta} min_{b \in \mathcal{B}} \|b - b'\|_1$, where Δ is the entire belief space and $\mathcal{B}$ is the set of belief points. Computing ϵ_b requires solving a Non-Linear Program, NLP (shown in Algorithm 1). Although HSVI has a slightly different error bound, it still requires the same NLP to be solved.

Algorithm 1 Non-Linear Program to obtain ϵ_b

```
**Maximize** $\epsilon_b$
*subject to the constraints*
    $\Sigma_{1 \leq i \leq |S|} b[i] = 1$
    $b[i] \geq 0$ and $b[i] \leq 1, \forall i \in \{1, \ldots, |S|\}$
    $\epsilon_b < \Sigma_{1 \leq i \leq |S|} \left| b[i] - b_k[i] \right|, \forall b_k \in \mathcal{B}$
```

2.3.3 JESP algorithm for Distributed POMDPs

Given the NEXP-complete complexity of generating globally optimal policies for distributed POMDPs [Bernstein et al., 2000], locally optimal approaches [Peshkin et al., 2000b; Chadès et al., 2002; Nair et al., 2003a] have emerged as viable solutions. Since CS-JESP algorithm (provided later) builds on JESP(Joint Equilibrium-Based Search for Policies) [Nair et al., 2003a] algorithm, JESP is outlined below (Algorithm 2). The key idea is to find the policy that maximizes the joint expected reward for one agent at a time, keeping policies of the other $n-1$ agents fixed. This process is repeated until an equilibrium is reached (local optimum is found). Multiple local optima are not encountered since planning is centralized. Key innovation in JESP is based on the realization that if policies of all other $n-1$ agents are fixed, then the remaining agent faces a normal single-agent POMDP, but with an extended state space. Thus, in line 4, given a known starting belief state, we use dynamic programming over belief states of this newer more complex single-agent POMDP, to compute agent 1's optimal response to fixed policies of the remaining $n-1$ agents.

Algorithm 2 JESP()

```
1: $\Pi' \leftarrow$ randomly selected joint policy, $prevVal \leftarrow$ value of $\Pi'$, $conv \leftarrow 0$, $\Pi \leftarrow \Pi'$
2: while $conv \neq n$ do
3:     for $i \leftarrow 1$ to $n$ do
4:         $val, \Pi_i \leftarrow$ OPTIMALBESTRESPONSE$(b, \Pi', T)$
5:         if $val = prevVal$ then
6:             $conv \overset{+}{\leftarrow} 1$
7:         else
8:             $\Pi'_i \leftarrow \Pi_i$, $prevVal \leftarrow val$, $conv \leftarrow 1$
9:         end if
10:        if $conv = n$ then break
11:    end for
12: end while
13: return $\Pi$
```

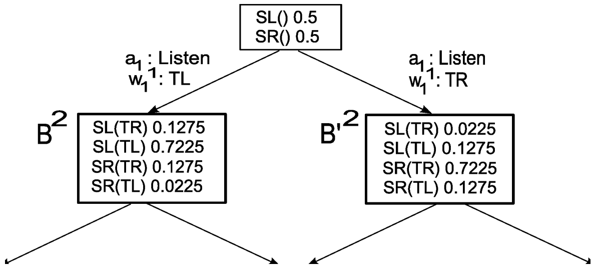


Figure 2.4: Trace of tiger scenario in JESP

The key is then to define the extended state in JESP. For a two agent case, for each time t, the extended state of agent 1 is defined as a tuple $e_1^t = \langle s^t, \vec{\omega}_2^t \rangle$, where $\vec{\omega}_2^t$ is the observation history of the other agent. By treating e_1^t as the state of agent 1 at time t, we can define the transition function and observation function for the resulting single-agent POMDP for agent 1 as follows:

$$\begin{aligned} P'(e_1^t, a_1^t, e_1^{t+1}) &= \Pr(e_1^{t+1} | e_1^t, a_1^t) \\ &= P(s^t, (a_1^t, \pi_2(\vec{\omega}_2^t)), s^{t+1}) \\ &\quad \cdot O_2(s^{t+1}, (a_1^t, \pi_2(\vec{\omega}_2^t)), \omega_2^{t+1}) \end{aligned} \tag{2.1}$$

$$\begin{aligned} O'(e_1^{t+1}, a_1^t, \omega_1^{t+1}) \quad &= \Pr(\omega_1^{t+1} | e_1^{t+1}, a_1^t) \\ &= O_1(s^{t+1}, (a_1^t, \pi_2(\vec{\omega}_2^t)), \omega_1^{t+1}) \end{aligned} \tag{2.2}$$

In other words, when computing agent 1's best-response policy via dynamic programming given the fixed policy of its teammate, we maintain a distribution over the extended states e_1^t, rather than over the world states s^t. Figure 2.4 shows a trace of the belief state evolution for the multi-agent tiger domain, described in Section 2.1.3, e.g. e_1^2 of SL(TR) indicates an extended state where the tiger is behind the left door and agent 2 has observed TR. However, as noted above, the main shortcoming of this technique is that it computes a locally optimal policy assuming a fixed starting belief state, and this assumption is embedded in its dynamic programming as shown in line 4 of algorithm 2 — it does not generate policies over continuous belief spaces.

Chapter 3

Exploiting structure in dynamics

This thesis aims to practically apply POMDPs to real world domains by introducing novel speedup techniques that are particularly suitable for such settings. The key insight is that in some dynamic domains where processes evolve over time, large but shifting parts of the belief space in POMDPs (i.e., regions of uncertainty) remain unreachable. Thus, we can focus policy computation on this reachable belief-space polytope that changes dynamically. For instance, consider a PAA monitoring a user driving to a meeting. Given knowledge of user's current location, the reachable belief region is bounded by the maximum probability of the user being in different locations at the next time step as defined by the transition function. Current POMDP algorithms typically fail to exploit such belief region reachability properties. POMDP algorithms that restrict belief regions fail to do so dynamically [Roy and Gordon, 2002; Hauskrecht and Fraser, 2000].

Our techniques for exploiting belief region reachability exploit three key domain characteristics: (i) not all states are reachable at each decision epoch, because of limitations of physical processes or progression of time; (ii) not all observations are obtainable, because not all states are reachable; (iii) the maximum probability of reaching specific states can be tightly bounded. We introduce polynomial time techniques based on Lagrangian analysis to compute tight bounds on belief state probabilities. These techniques are complementary to most existing exact and approximate POMDP algorithms. We enhance two state-of-the-art exact POMDP algorithms [Cassandra et al., 1997a; Feng and Zilberstein, 2004a] delivering over an order of magnitude speedup for a PAA domain.

3.1 Dynamic Belief Supports

Our approach consists of three key techniques: (i) dynamic state spaces (DS); (ii) dynamic beliefs (DB); (iii) dynamic disjunctive beliefs (DDB)[1] These ideas may be used to enhance existing POMDP algorithms such as GIP and RBIP. The key intuition is that for domains such as PAA, *progress* implies a dynamically changing polytope (of belief states) remains reachable through time, and policy computation can be speeded up by computing the parsimonious set over just this polytope. The speedups are due to the elimination of policies dominant in regions outside this polytope. DS provides an initial bound on the polytope, while DB (which captures DS) and DDB provide tighter bounds on reachable belief states through a polynomial-time technique obtained from Lagrangian analysis.

These techniques do not alter the relevant parsimonious set w.r.t. reachable belief states and thus, *yield an optimal solution* over the reachable belief states. The resulting algorithms (DS,DB,DDB) applied to enhance GIP are shown in Algorithm 3, where the functions GET-BOUND and DB-GIP are the main additions, with significant updates in other GIP functions (otherwise, the GIP descriptions follows [1,3]). We discuss our key enhancements in Algorithm 3 at the end of each subsection below. Our enhancements have currently been applied only to finite horizon problems and their applicability to infinite horizon problems remains an issue for future work.

3.1.1 Dynamic States(DS)

We first provide an intuitive explanation of DS using the example domain PAA. A natural method for PAAs to represent a user's state (such as in TMP) is with one consisting of a spatial element, (in a TMP, capturing the progress of each task), and a temporal element, capturing the stage of the decision. The transition matrix is then a static function of the state. This approach is used in [Scerri et al., 2002] for an adjustable autonomy problem addressed with MDPs. We note that in these kinds of domains, one cannot reach all states from a given state. For example, in the TMP scenario presented in Section 2.1.1, if there are limits on how tasks progress (one cannot advance more than one progress level in one time step, $T([x,t],a,[\tilde{x},t+1]) = 0$ if $\tilde{x} - x > 0.25$) and we know that at $t = 1$ we are at either $x = 0.00$ or $x = 0.25$, then we know at $t = 2$, $x \notin \{0.75, 1.00\}$ and at $t = 3$, $x \neq 1.00$.

Given this example, we now introduce the general concept of DS. The key insight is that the state space at each point in time can be represented more compactly in a dynamic fashion.

[1] We also have a technique called Dynamic observations (DO) that was presented in an earlier paper [Varakantham et al., 2005]

Algorithm 3 DB-GIP

```
Func POMDP-SOLVE ($L, S, A, T, \Omega, O, R$)
$(\{S_t\}, \{O_t\}, \{B_t^{max}\})$ = DB-GIP ($L, S, A, T, \Omega, O, R$)
$t \leftarrow L$;$V_t \leftarrow 0$
for $t = L$ to 1 do
    $\mathcal{V}_{t-1}$ = DP-UPDATE($\mathcal{V}_t, t$)
end for
Func DP-UPDATE ($\mathcal{V}, t$)
for all $a \in A$ do
    $\mathcal{V}_{t-1}^a \leftarrow \phi$
    for all $\omega_t \in O_t$ do
        for all $v_t^i \in V$ do
            for all $s_{t-1} \in S_{t-1}$ do
                $v_{t-1}^{a,\omega_t,i}(s_{t-1}) = r_{t-1}(s_{t-1}, a)/|O_t| + \gamma\Sigma_{s_t \in S_t} Pr(\omega_t, s_t|s_{t-1}, a)v_t^i(s_t)$
            end for
        end for
        $\mathcal{V}_{t-1}^{a,\omega_t} \leftarrow PRUNE(\{v_{t-1}^{a,\omega_t,i}\}, t)$
    end for
    $\mathcal{V}_{t-1}^a \leftarrow PRUNE(\mathcal{V}_{t-1}^a \oplus \mathcal{V}_{t-1}^{a,\omega_t}, t)$
end for
$\mathcal{V}_{t-1} \leftarrow PRUNE(\bigcup_{a \in A} \mathcal{V}_{t-1}^a, t)$
return $\mathcal{V}_{t-1}$
Func LP-DOMINATE($w, U, t$)
LP vars: $d, b(s_t)[\forall s_t \in S_t]$
LP max $d$ subject to:
    $b \cdot (w - u) \geq d, \forall u \in U$
    $\Sigma_{s_t \in S_t} b(s_t) \leftarrow 1$
    $b(s_t) <= b_t^{max}(s_t)$; $b(s_t) >= 0$
if $d \geq 0$ return $b$ else return nil
Func BEST($b, U$)
$max \leftarrow Inf$
for all $u \in U$ do
    if $(b \cdot u > max)$ or $((b \cdot u = max)$ and $(u <_{lex} w))$ then
        $w \leftarrow u$; $max \leftarrow b \cdot u$
    end if
end for
return $w$
Func PRUNE($U, t$)
$W \leftarrow \phi$
while $U \neq \phi$
$u \leftarrow$ any element in $U$
if POINT-DOMINATE($u, W, t$) = true then
    $U \leftarrow U - u$
else
    $b \leftarrow$LP-DOMINATE($u, W, t$)
    if $b = nil$ then $U \leftarrow U - u$
    else $w \leftarrow BEST(b, U)$;$W \leftarrow W \bigcup w$;$U \leftarrow U - w$
end if
return $W$
```

```
**Func** POINT-DOMINATE($w, U, t$)
**for all** $u \in U$ **do**
    if $w(s_t) \leq u(s_t), \forall s_t \in S_t$ then return true
**end for**
return false
**Func** DB-GIP($L, S, A, T, \Omega, O, R$)
$t \leftarrow 1$; $S_t =$Set of starting states
**for all** $s_t \in S_t$ **do**
    $b_t^{max}(s_t) = 1$
**end for**
**for** $t = 1$ to $L - 1$ **do**
    **for all** $s \in S_t$ **do**
        ADD-TO($S_{t+1}$,REACHABLE-STATES($s, T$))
        $\Omega_{t+1} =$ GET-RELEVANT-OBS($S_{t+1}, O$)
        C = GET-CONSTRAINTS ($s_t$)
        $b_{t+1}^{max}(s_{t+1}) =$ MAX$_{c \in C}$(GET-BOUND($s_{t+1}, c$))
    **end for**
**end for**
return ($\{S_t\}, \{\Omega_t\}, \{b_t^{max}\}$)
**Func** GET-BOUND($s_t, constraint$)
y$_{min}$ = MIN$_{s \in S_{t-1}}$$(constraint.c[s]/constraint.d[s])$
y$_{max}$ = MAX$_{s \in S_{t-1}}$$(constraint.c[s]/constraint.d[s])$
INT = GET-INTERSECT-SORTED($constraint, y_{min}, y_{max}$)
**for all** $i \in$ INT **do**
    $Z$ = SORT((($i + \epsilon) * constraint.d[s] - constraint.c[s]$),$\forall s \in S_{t-1}$
    $sumBound = 1, numer = 0, denom = 0$
    /* IN ASCENDING ORDER */
    **for all** $z \in Z$ **do**
        $s$ = FIND-CORRESPONDING-STATE($z$)
        **if** $sumBound - bound[s_{t-1}] > 0$ **then**
            $sumBound- = bound[s_{t-1}]$
            $numer+ = bound[s_{t-1}] * constraint.c[s_{t-1}]$
            $denom+ = bound[s_{t-1}] * constraint.d[s_{t-1}]$
        **end if**
        **if** $sumBound - bound[s_{t-1}] <= 0$ **then**
            $numer+ = sumBound * constraint.c[s_{t-1}]$
            $denom+ = sumBound * constraint.d[s_{t-1}]$
            BREAK-FOR
        **end if**
    **end for**
    **if** $numer/denom > i$ and $numer/denom < max$ **then**
        return $numer/denom$
    **end if**
**end for**
```

This will require the transition matrix and reward function to be dynamic themselves. Given knowledge about the initial belief space (e.g. possible beginning levels of task progress), we show how we can obtain dynamic state spaces and also that this representation does not affect the optimality of the POMDP solution. Let L be the length of a finite horizon decision process. Let S be the set of all possible states that can be occupied during the process. At time t, let $S_t \subset S$ denote the set of all possible states that could occur at that time. Thus, for any reachable belief state, we have $\sum_{s_t \in S_t} b_t(s_t) = 1$. Then, we can obtain S_t for $t \in 1, \ldots L$ inductively if we know the set $S_0 \subset S$ for which $s \notin S_0 \Rightarrow b_0(s) = 0$, as follows:

$$S_{t+1} = \left\{ s' \in S : \exists\, a \in A, s \in S_t \text{ s.t. } T_t(s, a, s') > 0 \right\} \tag{3.1}$$

The belief probability for a particular state $\tilde{s}$ at time $t+1$ given a starting belief vector at time t (b_t) action (a) and observation (ω) can be expressed as follows:

$$b_{t+1}(\tilde{s}) := \frac{O_t(\tilde{s}, a, \omega) \sum_{s_t \in S_t} T_t(s_t, a, \tilde{s}) b_t(s_t)}{\sum_{s_{t+1} \in S_{t+1}} O_t(s_{t+1}, a, \omega) \sum_{s_t \in S_t} T_t(s_t, a, s_{t+1}) b_t(s_t)}$$

This implies that the belief vector b_{t+1} will have support only on S_{t+1}, i.e. $\tilde{s} \notin S_{t+1} \Rightarrow b_{t+1}(\tilde{s}) = 0$, if b_t only has support in S_t and S_{t+1} is generated as in (3.1). Thus, we can model a process that migrates among dynamic state spaces $\{S_t\}_{t=1}^{L}$ indexed by time or more accurately, the stage of the decision process as opposed to a transitioning within static global state set S.

Proposition 1 *Given S_0, we can replace a static state space S with dynamic state spaces $\{S_t\}$ generated by (3.1), dynamic transition matrices and dynamic reward functions in a finite horizon POMDP without affecting the optimality of the solution obtained using value function methods.*

Proof. If we let P_t denote the set of policies available at time t, V_t^p denote the value of policy p at time t and, V_t^* denote the value of the optimal policy at time t, we have $V_L^*(b_L) = \max_{p \in P_L} b_L \cdot \alpha_L^p$ where $\alpha_L^p = [V_L^p(s_1) \cdots V_L^p(s_{|S|})]$ for $s_i \in S$.

When $t = L$, we have $V_L^p(s) = R_L(s, a(p))$ where R_L is the reward function at time L and $a(p)$ is the action prescribed by the policy p. Since $b_L(s) = 0$ if $s \notin S_L$, then $V_L^*(b_L) = \max_{p \in P_L} \tilde{b}_L \cdot \tilde{\alpha}_L^p$ where $|\tilde{b}_L| = |\tilde{\alpha}_L^p| = |S_L|$ and $\tilde{\alpha}_L^p = [V_L^p(\tilde{s}_1) \cdots V_L^p(\tilde{s}_{|S_L|})]$ for $\tilde{s}_i \in S_L$. Calculating the value function at time $L-1$, we have $V_{L-1}^*(b_{L-1}) = \max_{p \in P_{L-1}} b_{L-1} \cdot \alpha_{L-1}^p$ where $\alpha_{L-1}^p = [V_{L-1}^p(s_1) \cdots V_{L-1}^p(s_{|S|})]$ for $s_i \in S$.

When $t = L-1$, we have

$V_{L-1}^p(s) = R_{L-1}(s, a(p)) + \gamma \sum_{s' \in S} T_{L-1}(s, a(p), s') \sum_{\omega \in \Omega} O(s', a, \omega) V_L^{p_\omega}(s')$, where $p_\omega \in P_L$ is the policy subtree of the policy tree $p \in P_{L-1}$ when observing ω after the initial action. Since $b_{L-1}(s) = 0$ if $s \notin S_{L-1}$, then $V_{L-1}(b_{L-1}) = \max_{p \in P_{L-1}} \tilde{b}_{L-1} \cdot \tilde{\alpha}^p_{L-1}$ where $|\tilde{b}_{L-1}| = |\tilde{\alpha}^p_{L-1}| = |S_{L-1}|$ and $\tilde{\alpha}^p_{L-1} = [V^p_L(\tilde{s}_1) \cdots V^p_L(\tilde{s}_{|S_{L-1}|}])$ for $\tilde{s}_i \in S_{L-1}$. Applying this reasoning inductively, we can show that we only need $V^p_t(s_t)$ for $s_t \in S_t$. Furthermore, if $s_t \in S_t$, then

$$V^p_t(s_t) \;=\; R_t(s_t, a(p)) + \gamma \sum_{s_{t+1} \in S_{t+1}} T_t(s_t, a(p), s_{t+1}) \sum_{\omega \in \Omega} O(s_{t+1}, a, \omega) V^{p_\omega}_{t+1}(s_{t+1}). \quad (3.2)$$

Thus, we only need $\{V^{\omega(p)}_{t+1}(s_{t+1}) : s_{t+1} \in S_{t+1}\}$. ■

The value functions expressed for beliefs over dynamic state spaces S_t have identical expected rewards as when using S. The advantage in this method is that in generating the set of value vectors which are dominant at some underlying belief point (i.e. the parsimonious set) at a particular iteration, we eliminate vectors that are dominant over belief supports that are not reachable. This reduces the set of possible policies that need to be considered at the next iteration. Line 6 of DB-GIP function and the DP-UPDATE function of Algorithm 3 provide the algorithm for finding the dynamic states.

3.1.2 Dynamic Beliefs(DB)

By introducing dynamic state spaces, we are attempting to more accurately model the support on which reachable beliefs will occur. We can make this process more precise by using information about the initial belief distribution, the transition and observation probabilities to bound belief dimensions with positive support. For example, if we know that our initial belief regarding task progress can have at most 0.10 probability of being at 0.25 with the rest of the probability mass on being at 0.00, we can find the maximum probability of being at 0.25 or 0.50 at the next stage, given a dynamic transition matrix. Below we outline a polynomial-time procedure by which we can obtain such bounds on belief support.

Figure 3.1.2 provides an example comparison of belief bounds obtained using DB and GIP. In the figure, each rectangular box represents a state and the states in the one column represent the states of the POMDP. Each column represents an iteration of dynamic programming and the arrows between the boxes represent the transitions between states. The number inside each state represents the maximum possible belief probability for a state at that iteration. With GIP, this number remains 1 for all the states at all the iterations. While with DB, given the dynamics of the domain, it is possible to obtain a configuration as shown in the figure on the right side.

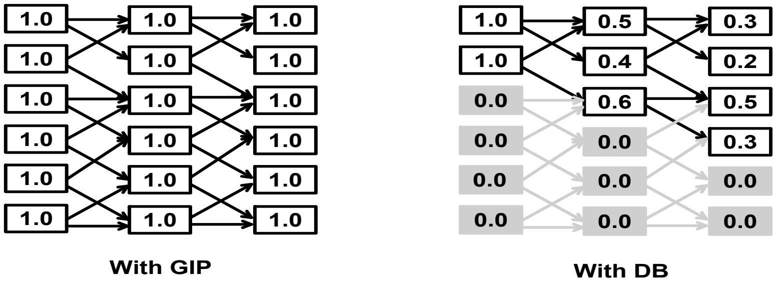


Figure 3.1: Comparison of GIP and DB with respect to belief bounds

Let $B_t \subset [0\ 1]^{|S_t|}$ be a space such that $P(b_t \notin B_t) = 0$. That is, there exists no initial belief vector and action/observation sequence of length $t-1$ such that by applying the standard belief update rule, one would get a belief vector b_t not captured in the set B_t. Then, we have

$$b_{t+1}(s_{t+1}) \geq \min_{a\in A, o\in O_t, b_t\in B_t} F(s_{t+1}, a, o, b_t) =: b_{t+1}^{\min}(s_{t+1})$$
$$b_{t+1}(s_{t+1}) \leq \max_{a\in A, o\in O_t, b_t\in B_t} F(s_{t+1}, a, o, b_t) =: b_{t+1}^{\max}(s_{t+1})$$

where $F(s_{t+1}, a, o, b_t) :=$

$$\frac{O_t(s_{t+1}, a, o)\sum_{s_t\in S_t} T_t(s_t, a, s_{t+1})b_t(s_t)}{\sum_{\tilde{s}_{t+1}\in S_{t+1}} O_t(\tilde{s}_{t+1}, a, o)\sum_{s_t\in S_t} T_t(s_t, a, \tilde{s}_{t+1})b_t(s_t)}$$

Thus, if

$$B_{t+1} = [b_{t+1}^{\min}(s_1) b_{t+1}^{\max}(s_1)] \times \cdots \times [b_{t+1}^{\min}(s_{|S_{t+1}|}) b_{t+1}^{\max}(s_{|S_{t+1}|})],$$

then we have $P(b_{t+1} \notin B_{t+1}) = 0$.

We now show how $b_{t+1}^{\max}(s_{t+1})$ (and similarly $b_{t+1}^{\min}(s_{t+1})$) can be generated through a polynomial-time procedure deduced from Lagrangian methods. Given an action a and observation ω, we can express the problem as

$$\max_{b_t\in B_t} b_{t+1}^{a,\omega}(s_{t+1}) \quad \text{s.t.} \quad b_{t+1}^{a,\omega}(s_{t+1}) = c^T b_t / d^T b_t$$

where $c(s) = O_t(s_{t+1}, a, \omega)T_t(s_t, a, s_{t+1})$ and
$d(s) = \sum_{s_{t+1} \in S_{t+1}} O_t(s_{t+1}, a, \omega)T_t(s_t, a, s_{t+1})$. We rewrite the problem in terms of the new variables as follows:

$$\min_x \left(-c^T x / d^T x\right) \;\text{ s.t. } \sum_i x_i = 1,\; 0 \leq x_i \leq b_t^{\max}(s_i) =: \bar{x}_i - (3)$$

where $\sum_i b_t^{\max}(s_i) \geq 1$ to ensure existence of a feasible solution. Expressing this problem as a Lagrangian, we have

$$\mathcal{L} = \left(-c^T x / d^T x\right) + \lambda(1 - \sum_i x_i) + \sum_i \bar{\mu}_i(x_i - \bar{x}_i) - \sum_i \mu_i x_i$$

from which the KKT conditions imply

$$\begin{aligned}
&x_k = \bar{x_k} && \lambda = [(c^T x)d_k - (d^T x)c_k]/(d^T x)^2 + \bar{\mu}_k \\
&0 < x_k < \bar{x_k} && \lambda = [(c^T x)d_k - (d^T x)c_k]/(d^T x)^2 \\
&x_k = 0 && \lambda = [(c^T x)d_k - (d^T x)c_k]/(d^T x)^2 - \mu_k.
\end{aligned}$$

Because λ is identical in all three conditions and $\bar{\mu}_k$ and μ_k are non-negative for all k, the component with the lowest value of $(d^T x)\lambda = [(c^T x)/(d^T x)]d_k - c_k$ must receive a maximal allocation (assuming $\bar{x}_k < 1$) or the entire allocation otherwise. For example, if size of state space is 3 and the values of the expression $[(c^T x)/(d^T x)]d_k - c_k$ for different values of k are 5, 6, 7 (assuming a state space of 3). Since all the λs (over all x_k) are identical, the above values need to be made equal by deciding on the allocations for each of the x_ks. Since $\sum_k x_k = 1$, it cannot be the case that all these values are reduced, since reduction happens only in the third equation for λ where $x_k = 0$ (since there is a subtraction of non negative variable μ_k). Thus, it is imperative that smaller of these values increase. As can be observed from the equations of λ, values can be increased only in the case of $x_k = \bar{x_k}$ (since there is a non negative variable $\bar{\mu}_k$ in the equation), and hence full allocation for smaller values of $(d^T x)\lambda = [(c^T x)/(d^T x)]d_k - c_k$.

Using this reasoning recursively, we see that if x^* is an extremal point (i.e. a candidate solution), then the values of its components $\{x_k\}$ must be constructed by giving as much weight possible to components in the order prescribed by $z_k = yd_k - c_k$, where $y = (c^T x^*)/(d^T x^*)$. Given a value of y, one can construct a solution by iteratively giving as much weight as possible (without violating the equality constraint) to the component not already at its bound with the lowest z_k.

The question then becomes finding the maximum value of y which yields a consistent solution. We note that y is the value we are attempting to maximize, which we can bound with $y_{\max} = \max_i c_i/d_i$ and $y_{\min} = \min_i c_i/d_i$. We also note that for each component k, z_k describes a line over the support $[y_{\min}, y_{\max}]$. We can then find the set of all points where the set of lines described by $\{z_k\}$ intersect. There can be at most $(N-1)N/2$ intersections points. We can then partition the support $[y_{\min}, y_{\max}]$ into disjoint intervals using these intersection points yielding at most $(N-1)N/2+1$ regions. In each region, there is a consistent ordering of $\{z_k\}$ which can be obtained in polynomial time. An illustration of this can be seen in Figure 3.1.2. Beginning with the region furthest to the right on the real line, we can create the candidate solution implied by the ordering of $\{z_k\}$ in that region and then calculate the value of y for that candidate solution. If the obtained value of y does not fall within region, then the solution is inconsistent and we move to the region immediately to the left. If the obtained value of y does fall within the region, then we have the candidate extremal point which yields the highest possible value of y, which is the solution to the problem. By using this technique we can dynamically propagate forward bounds on feasible belief states. Line 12 and 13 of the DB-GIP function in Algorithm 3 provide the procedure for DB. The GET-CONSTRAINTS function on Line 12 gives the set of c^T and d^T vectors for each state at time t for each action and observation.

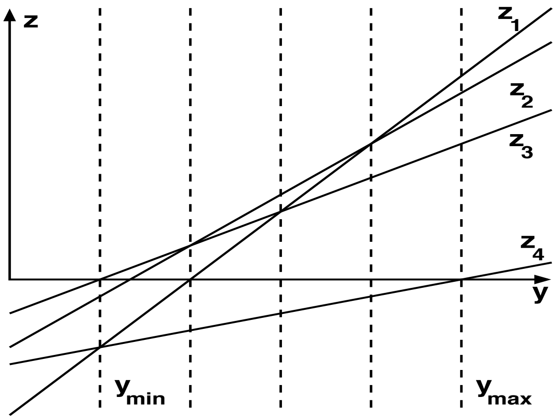


Figure 3.2: Partition Procedure for Solving Belief Maximization Lagrangian

In the belief maximization equation of (3), if $b_t^{max}(s_i)$ is equal to 1 for all states s_i, then it can be easily proved that the maximum value is equal to $max_k\ c_k/d_k$. Thus this special case doesn't even require the complexity of the lagrangian method, and can be solved in O($|S|log(|S|)$). However, if the maximum possible value of belief probability in the previous stage is not equal to 1, $max_k\ c_k/d_k$ can serve only as a bound and not the exact maximum. A simple improvement to

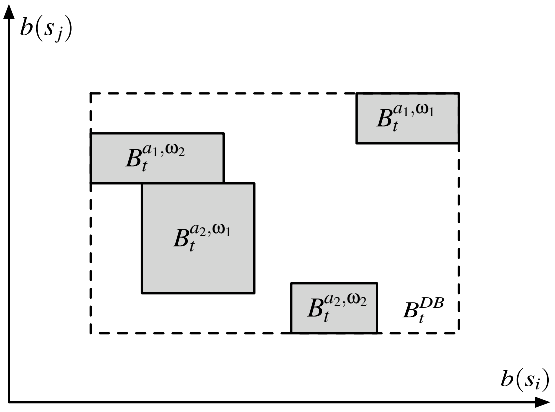


Figure 3.3: Illustration of DDB vs DB

the above method is assigning x_ks their maximum value (until the sum is 1) based on the order of c_k/d_k. However, as can be seen in the example below, this method doesn't yield the maximum.

$$\begin{aligned}&\max\left((0.6x_1 + 0.3x_2 + 0.7x_3)\,/\,(0.8x_1 + 0.5x_2 + 0.9x_3)\right)\\ &\text{s.t. } \; 0 < x_1 < 0.8, 0 < x_2 < 0.6, 0 < x_3 < 0.5, \sum_i x_i = 1\end{aligned}$$

By using dynamic beliefs, we increase the costs of pruning by adding some constraints. However, there is an overall gain because we are looking for dominant vectors over a smaller support and this reduces the cardinality of the parsimonious set, leaving fewer vectors to consider at the next iteration.

3.1.3 Dynamic Disjunctive Beliefs(DDB)

The key insight in DB is that given a bounded set of beliefs at the beginning of the problem, there are many beliefs that are not possible at later stages. By eliminating reasoning about policies that are optimal at unreachable beliefs, run-time can be improved significantly without sacrificing the quality of the solution. This is accomplished by performing the pruning operation over the reachable belief polytope rather than the entire simplex. In DB, we outlined the procedure to obtain the maximum belief to be assigned to a particular state at a particular epoch for a particular action and observation. Let us denote this as $b_{t+1}^{a,\omega,\max}(s)$, which is the output of the constrained optimization problem solved with Lagrangian techniques. We can similarly find the minimum possible belief.

In DB, the dynamic belief polytope for an epoch t is created as follows:

1. Find the maximum and minimum possible belief for each state over all actions and observations: $b_{t+1}^{\max}(s) = \max_{a,\omega} b_{t+1}^{a,\omega,\max}(s)$, $b_{t+1}^{\min}(s) = \max_{a,\omega} b_{t+1}^{a,\omega,\min}(s)$.

2. Create a belief polytope that combines these bounds over all states: $B_{t+1} =$

$$[b_{t+1}^{\min}(s_1)\ b_{t+1}^{\max}(s_1)] \times \cdots \times [b_{t+1}^{\min}(s_{|S|})\ b_{t+1}^{\min}(s_{|S|})].$$

While this is an appropriate bound in that any belief outside this is not possible given the initial beliefs, transition probabilities and observation probabilities, it is possible to make an even tighter expression of the feasible beliefs. We refer to this new method for reducing feasible beliefs as Dynamic Disjunctive Belief (DDB) bounds. The disjunctions are due to the fact that future beliefs depend on particular action-observation pairs ($\max$ is over a, ω in (1) above). By eliminating the conditioning on actions and observations, we may be including infeasible beliefs. This is illustrated in Figure 3.3, for a two-observation, two-action system over a two-dimensional support. We see that B_t^{DB}, while smaller than the entire space of potential beliefs, is larger than necessary as it is not possible to believe anything outside of $\cup_{i=1,2} B_t^{a_i,\omega_i}$. Thus, we introduce the DDB method for computing feasible belief spaces:

1. Obtain $b_{t+1}^{a,\omega,\max}(s)$ and $b_{t+1}^{a,\omega,\min}(s),\ \forall a \in A, \omega \in \Omega$

2. Create multiple belief polytopes, one for each action-observation pair, as follows:$B_{t+1}^{a_i,\omega_i} =$

$$[b_{t+1}^{a_i,\omega_i,\min}(s_1) \quad b_{t+1}^{a_i,\omega_i,\max}(s_1)] \quad \times \quad \cdots \quad \times \quad [b_{t+1}^{a_i,\omega_i,\min}(s_{|S|}) \quad b_{t+1}^{a_i,\omega_i,\max}(s_{|S|})].$$

The feasible belief space is then $B_t^{DDB} = \cup_{a,w} B_t^{a,w}$. However, this is disjunctive and cannot be expressed in the LP that is used for pruning. Instead, we prune over each $B_t^{a,w}$ and take the union of dominant vectors for these supports. This increases the number of LP calls for a fixed epoch but the LPs cover smaller spaces and will yield fewer vectors at the end. Figure 3.1.3 provides an instance from the example in Figure 2.3.1, where DB and DDB provide improved performance when compared against GIP. With GIP the parsimonious set consisted of six policies. However, given a belief bound of 0.2-0.8 with DB, the parsimonious set only consists of four policies as opposed to six. Similarly with DDB, if we assume there were two small regions (corresponding to the region 0.2-0.8) 0.2-0.3 and 0.7-0.8(say). In this instance, the parsimonious set obtained with DDB would only consist of two policies. This reduction in the size of the parsimonious set provides improvement in performance because of the cascade effect it has on the sizes of the parsimonious set at future iterations.

I will present the experimental results for these techniques (DS, DB, DDB) in chapter 5.

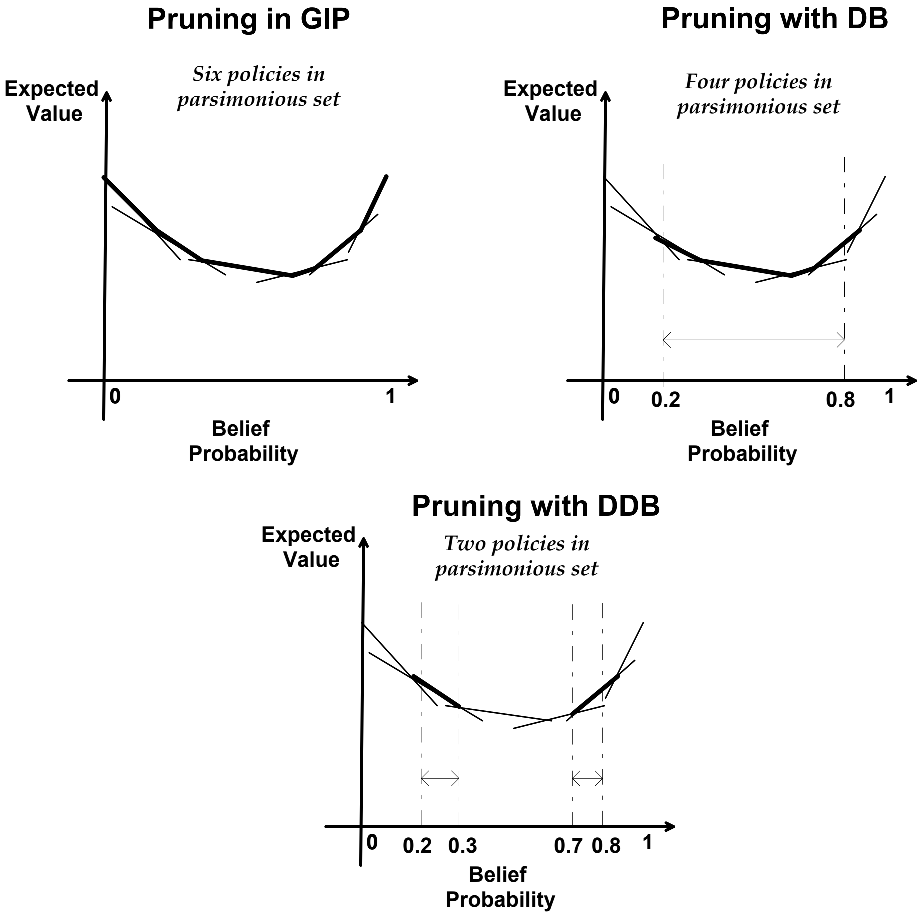


Figure 3.4: Illustration of pruning in DB and DDB when compared against GIP

Chapter 4

Direct value approximation for POMDPs

Approximate algorithms, a currently popular approach, address the significant computational complexity in POMDP policy generation by sacrificing solution quality for speed [Pineau et al., 2003; Smith and Simmons, 2005; Zhou and Hansen, 2001; Hauskrecht, 2000a]. Furthermore, most of the approximate algorithms do not provide any guarantees on the quality loss and the ones (point-based approaches [Pineau et al., 2003; Smith and Simmons, 2005]) which provide expressions for error bounds have the following drawbacks: (a) The bounds in these point-based approximation algorithms are based on maximum and minimum possible single-stage reward, which can take extreme values, leading to a very loose bound which is not useful in many domains, especially in those that have penalties (e.g., $R_{min} \ll 0$). (b) The computation of the bound for a particular policy is itself a potentially significant expense (e.g., requiring a non-linear program). (c) The algorithms cannot guarantee that they will yield a policy that can achieve a pre-specified error bound.

This earlier work in approximately solving POMDPs has focused primarily on sampling the belief space, and finding policies corresponding to a sampled set of belief points. These policies are then extrapolated to the entire belief space [Pineau et al., 2003; Smith and Simmons, 2005; Zhou and Hansen, 2001; Hauskrecht, 2000a]. On the contrary, we propose an approach, Expected Value Approximation (EVA), that approximates policies directly based on the their expected values. Thus, this approach to approximation is beneficial in domains that require tight bounds on solution quality. Furthermore, EVA provides a bound that does not depend on the maximum and minimum possible rewards ($R_{\max}$ and $R_{\min}$).

The value function in a POMDP is piecewise linear and can be expressed by a set of vectors (representative of policies). Approximate algorithms generate smaller vector sets (and thus, a reduced policy space) than the optimal sets. Existing approaches generate these vector sets by sampling the possible beliefs and finding the vectors that dominate over this space. However in EVA, we approximate directly in the value space by representing the optimal set of vectors with a set of vectors whose expected reward will be within a desired bound of the optimal reward. In a multi-stage decision process, solved using dynamic programming techniques, the reduction of vectors at one stage can result in fewer vectors getting generated at future stages. This can result in the improvement of overall performance, because pruning (the most expensive step in solving a POMDP) requires fewer number of steps for the reduced set.

4.1 EVA Algorithm

The value function in a POMDP is piecewise linear and can be expressed by a set of vectors. Approximate algorithms generate fewer vector sets than the optimal algorithms. Existing approaches generate these reduced vector sets by sampling the belief space and finding the vectors that apply only at these points. In our approach, Expected Value Approximation (EVA), we choose a reduced set of vectors by approximating the value space with a subset of vectors whose expected reward will be within a desired bound of the optimal reward.

Using an approximation (subset) of the optimal parsimonious set will lead to lower expected quality at some set of belief points. Let ϵ denote the maximum loss in quality we will allow at any belief point. We henceforth refer to any vector set that is at most ϵ away from the optimal value at all points in the belief space (as illustated in Fig 4.1) as an ϵ-parsimonious set. The key problem in EVA is to determine this ϵ-parsimonious set efficiently.

To that end, we employ a heuristic that extends the pruning strategies presented in GIP. In GIP, a parsimonious set $\mathcal{V}$ corresponding to a set of vectors, $\mathcal{U}$ is obtained in three steps:

1. Initialize the parsimonious set $\mathcal{V}$ with the dominant vectors at the simplex points.

2. For some chosen vector $u \in \mathcal{U}$, execute a LP to compute the belief point b where u dominates the current parsimonious set $\mathcal{V}$.

3. Compute the vector u' with highest expected value in the set $\mathcal{U}$ at the belief point, b; remove vector u' from $\mathcal{U}$ and add it to $\mathcal{V}$.

EVA modifies the first two steps, to obtain the ϵ-parsimonious set:

1. Since we are interested in representing the optimal parsimonious set with as few vectors as possible, the initialization process only selects one vector over the beliefs in the simplex extrema. We choose a vector with the highest expected value at the most number of simplex belief points, choosing randomly to break ties.

2. The LP is modified to check for ϵ-dominance, i.e., dominating all other vectors by ϵ at some belief point. Algorithm 4 provides a modified LP with b_t^{max} and b_t^{min}.

Algorithm 4 LP-DOMINATE($w, U, b_t^{max}, b_t^{min}, \epsilon$)

```
1: variables: $d, b(s_t)\ \forall s_t \in S_t$
2: maximize $d$
3: subject to the constraints
4:   $b \cdot (w - u) \geq d + \epsilon, \forall u \in U$
5:   $\Sigma_{s_t \in S_t} b(s_t) = 1$, $b(s_t) \leq b_t^{max}(s_t)$, $b(s_t) \geq b_t^{min}(s_t)$
6: if $d \geq 0$ return $b$ else return nil
```

The key difference between the LP used in GIP and the one in Algorithm 4 is the ϵ in RHS of line 4 which checks for expected value dominance of the given vector w over a vector $u \in U$. Including ϵ as part of the RHS constrains w to dominate other vectors by at least ϵ. In the following propositions, we prove the correctness of the EVA algorithm and the error bound provided by EVA. Let $\mathcal{V}^\epsilon$ and $\mathcal{V}^*$ denote the ϵ-parsimonious and optimal parsimonious set, respectively.

Proposition 2 *$\forall b \in \Delta$, the entire belief space, if $v_b^\epsilon = \arg\max_{v^\epsilon \in \mathcal{V}^\epsilon} v^\epsilon \cdot b$ and $v_b^* = \arg\max_{v^* \in \mathcal{V}^*} v^* \cdot b$, then $v_b^\epsilon \cdot b + \epsilon \geq v_b^* \cdot b$.*

Proof. We prove this by contradiction. Assume $\exists b \in \Delta$ such that $v_b^\epsilon \cdot b + \epsilon < v_b^* \cdot b$. This implies $v_b^\epsilon \neq v_b^*$, and $v_b^* \notin \mathcal{V}^\epsilon$. We now consider the situation(s) when v_b^* is considered by EVA. At these instants, there will be a current parsimonious set $\mathcal{V}$ and a set of vectors still to be considered $\mathcal{U}$. Let

$$\hat{b} = \arg\max_{\tilde{b} \in \Delta} \{\min_{v \in \mathcal{V}} (v_b^* \cdot \tilde{b} - v \cdot \tilde{b})\}$$

be the belief point at which v_b^* is best w.r.t. $\mathcal{V}$. Let

$$\hat{v}_{\hat{b}} = \arg\max_{v \in \mathcal{V}} v \cdot \hat{b}$$

be the vector in $\mathcal{V}$ which is best at $\hat{b}$. Let

$$\hat{u}_{\hat{b}} = \arg\max_{u \in \mathcal{U}} u \cdot \hat{b}$$

be the vector in $\mathcal{U}$ which is best at $\hat{b}$. There are three possibilities:

1. $v_b^* \cdot \hat{b} < \hat{v}_{\hat{b}} \cdot \hat{b} + \epsilon$: This implies $v_b^* \cdot \hat{b} - \hat{v}_{\hat{b}} \cdot \hat{b} < \epsilon$. By the definition of $\hat{b}$, we have

$$v_b^* \cdot b - \hat{v}_b \cdot b < v_b^* \cdot \hat{b} - \hat{v}_{\hat{b}} \cdot \hat{b} < \epsilon$$

where $\hat{v}_b = \arg\max_{v \in \mathcal{V}} v \cdot b$. This implies

$$v_b^* \cdot b < \hat{v}_b \cdot b + \epsilon \leq v_b^\epsilon \cdot b + \epsilon,$$

which is a contradiction.

2. $v_b^* \cdot \hat{b} \geq \hat{v}_{\hat{b}} \cdot \hat{b} + \epsilon$ and $v_b^* \cdot \hat{b} \geq \hat{u} \cdot \hat{b}$: This means v_b^* would have been included in the ϵ-parsimonious set, $v_b^* \in \mathcal{V}^\epsilon$, which is a contradiction.

3. $v_b^* \cdot \hat{b} \geq \hat{v}_{\hat{b}} \cdot \hat{b} + \epsilon$ and $v_b^* \cdot \hat{b} < \hat{u} \cdot \hat{b}$: $\hat{u}$ will be included in $\mathcal{V}$ and v_b^* is returned to $\mathcal{U}$ to be considered again until one of previous two terminal conditions occur. ■

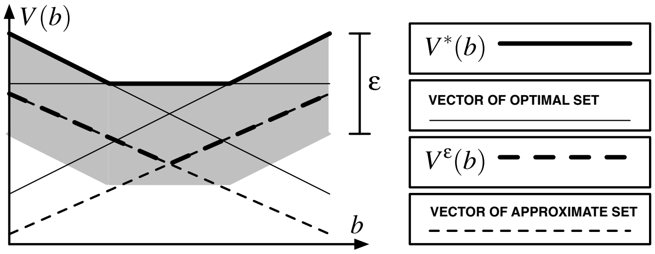


Figure 4.1: EVA: An example of an ϵ-parsimonious set

Proposition 3 *The error introduced by EVA at each stage of the policy computation, is bounded by* $2\epsilon|\Omega|$ *for GIP-type cross-sum pruning.*

Proof. The EVA algorithm introduces an error of ϵ in a parsimonious set whenever a pruning operation (PRUNE) is performed, due to Proposition 2. In GIP, there are three pruning steps at each stage of policy computation.

1. $\mathcal{V}^{a,o} = PRUNE(\mathcal{V}^{a,o,i})$: After this step, each $\mathcal{V}^{a,o}$ is at most ϵ away from optimal $\forall a, \forall o$.

2. $\mathcal{V}^a = PRUNE(\cdots(PRUNE(\mathcal{V}^{a,o_1} \oplus \mathcal{V}^{a,o_2})\cdots \oplus \mathcal{V}^{a,o_{|\Omega|}})$: We begin with $\mathcal{V}^{a,o_1}$ which is away from optimal by at most ϵ. Each pruning operation adds 2ϵ to the bound (ϵ for the new term $\mathcal{V}^{a,o_i}$ and ϵ for the $PRUNE$). There are $|\Omega| - 1$ prune operations. Thus, each $\mathcal{V}^{a,o}$ is away from the optimal by at most $2\epsilon(|\Omega| - 1) + \epsilon$.

3. $\mathcal{V}' = PRUNE(\bigcup_{a \in A} \mathcal{V}^a)$: The error of $\bigcup_{a \in A} \mathcal{V}^a$ is bounded by the error of $\mathcal{V}^a$. The $PRUNE$ adds ϵ, leading to a total one-stage error bound of $2\epsilon|\Omega|$. ■

Proposition 4 *The total error introduced by EVA (for GIP-type cross-sum pruning) is bounded by $2\epsilon|\Omega|T$ for a T-horizon problem.*

Proof. Let $V_t^{\epsilon}(b)$ and $V_t^{*}(b)$ denote the EVA-policy and optimal value function, respectively, at time t. If $A_t^{\epsilon} \subseteq A$, is the set of actions at the roots of all policy-trees associated with $\mathcal{V}_t^{\epsilon}$, the EVA vector set for time t and $e_t = \max_{b \in B}\{V_t^{*}(b) - V_t^{\epsilon}(b)\}$, then, $V_{t-1}^{\epsilon}(b) =$

$$
\begin{aligned}
&= max_{a \in A_t^{\epsilon}}\{R(b,a) + \Sigma_{b' \in B} P(b'|b,a) V_t^{\epsilon}(b')\} \\
&\geq max_{a \in A_t^{\epsilon}}\{R(b,a) + \Sigma_{b' \in B} P(b'|b,a) V_t^{*}(b')\} \\
&\quad - \Sigma_{b' \in B} P(b'|b,a) e_t \\
&= max_{a \in A_t^{\epsilon}}\{R(b,a) + \Sigma_{b' \in B} P(b'|b,a) V_t^{*}(b')\} - e_t \\
&\geq max_{a \in A}\{R(b,a) + \Sigma_{b' \in B} P(b'|b,a) V_t^{*}(b')\} - 2\epsilon|\Omega| - e_t \\
&= V_{t-1}^{*}(b) - 2\epsilon|\Omega| - e_t
\end{aligned}
$$

The last inequality is due to Proposition 2, and the other relations are by definition. The above implies that $e_{t-1} = e_t + 2\epsilon|\Omega|$ and with $e_T = 2\epsilon|\Omega|$, we have the error for the EVA-policy at the beginning of execution, $e_1 = 2\epsilon|\Omega|T$ (the total error bound for EVA). ■

Similarly, it can be proved that for γ-discounted infinite horizon problems, the total error bound is $\frac{2\epsilon|\Omega|}{1-\gamma}$.

I present the experimental results for EVA combined with the belief bound techniques in chapter 5.

Chapter 5

Results for DS, DB, DDB and EVA

This chapter focuses on experimental results with the techniques introduced in chapters 3 and 4. While these techniques could be used in conjunction with different exact algorithms, including both GIP and RBIP, in this chapter we will focus on enhancing the GIP algorithm. All our enhancements were implemented over GIP [1] We implemented over GIP, as our enhancements over GIP performed better than over RBIP [Varakantham et al., 2005]. Thus in the following paragraphs, DS refers to GIP+DS, DB to GIP+DB, DDB to GIP+DDB and EVA to EVA+GIP. All the experiments[2] compare the performance (run-time) of GIP, RBIP and our enhancements over GIP. The experimental setup consisted of 10 TMP problems. Each problem had pre-specified run-time upper limit of 20000 seconds.

We conducted two sets of experiments with regards to the enhancements presented in Chapter 3 and Chapter 4. The first set of experiments focused on the Task Management Problem (TMP) [Varakantham et al., 2005] in software personal assistant domains. In comparing with other algorithms, it is useful to recall from chapter 2, that this domain has a reward of negative infinity associated with certain actions. As suggested earlier in chapter 2, a POMDP algorithm solving TMP problems needs to have the following characteristics: (a) Compute policy for a pre-specified quality bound; (b) This bound must hold for all possible starting belief points – because we may start the problem in any possible starting belief states; (c) Efficiency of policy computation is of the essence because if agents require significant amounts of computation prior to each task allocation, then that could hinder human task performance.

Existing approaches to solving POMDPs have limited applicability in this domain. Approximate approaches provide trivial bounds owing to the presence of the negative infinity reward (R_{min}). Furthermore, algorithms like PBVI (and HSVI) can provide a guarantee on solution

[1] Our enhancements were implemented over Anthony Cassandra's POMDP solver "http://pomdp.org/pomdp/code/index.shtml.

[2] Machine specs for all experiments: Intel Xeon 2.8 GHZ processor, 2GB RAM, Linux Redhat 8.1

quality corresponding to a fixed starting belief point only, thus failing in (b) above. Though exact algorithms provide quality guarantees, they do so at the cost of computational complexity, losing out on (c). Our belief bound techniques along with EVA, though limited in their applicability individually in these problems, can in combination handle the constraints mentioned above.

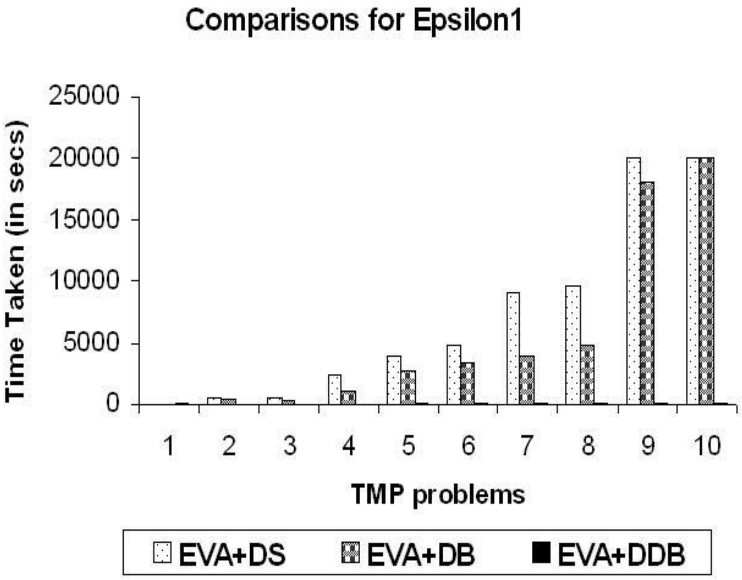


Figure 5.1: Comparison of performance of EVA+DS, EVA+DB, and EVA+DDB for ϵ=0.01

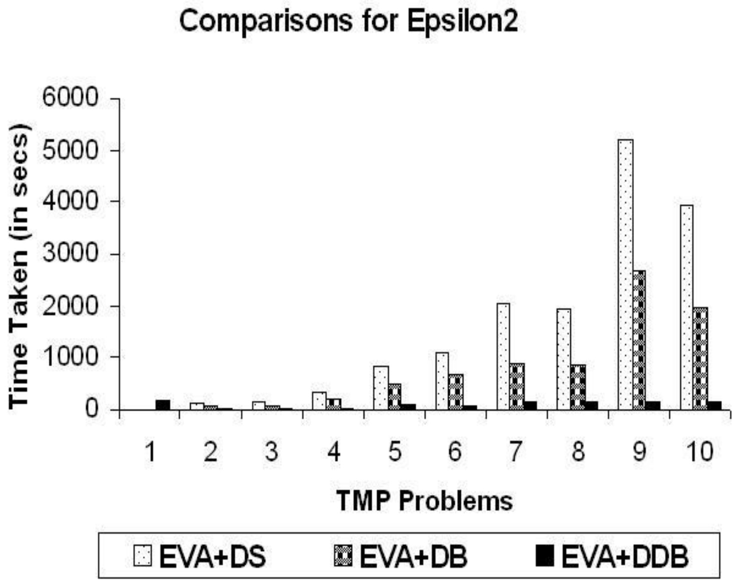


Figure 5.2: Comparison of performance of EVA+DS, EVA+DB, and EVA+DDB for ϵ=0.02

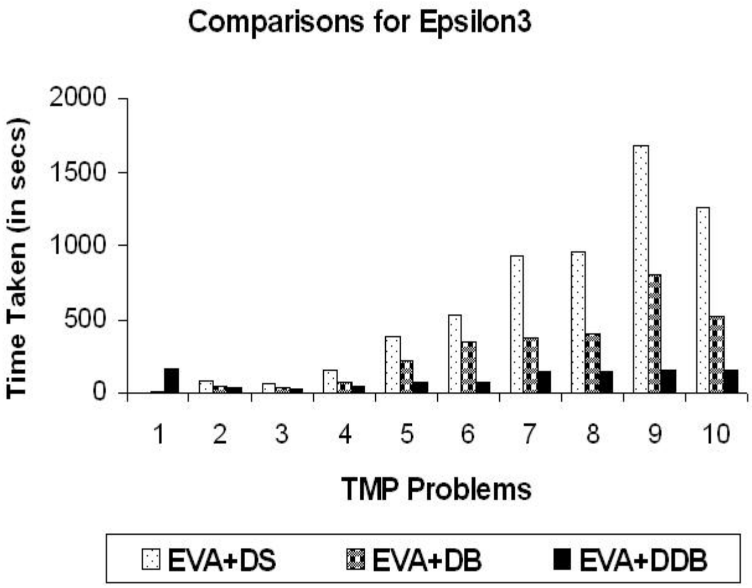


Figure 5.3: Comparison of performance of EVA+DS, EVA+DB, and EVA+DDB for ϵ=0.03

None of GIP, RBIP, EVA, DS, DB, DDB terminated within the prespecified limit of 20,000 seconds for either of the problems. EVA was run with a low error bound to illustrate the utility of DS, DB and DDB techniques. We combined our exact enhancements (DS, DB, DDB) with EVA. Figures 5.1, 5.2 and 5.3 provides the comparison of performances of these combined techniques for varying ϵ values in EVA. In Figures 5.1, 5.2 and 5.3, x-axis indicates the TMP problems, while the y-axis indicates the time to solution in seconds. The three bars, shown for each problem, indicate the run-times of EVA+DS, EVA+DB, and EVA+DDB. All the three figures clearly illustrate the dominance of DDB over DB and DS. For instance in Figure 5.1, EVA+DDB provides 66.9-fold speedup over EVA+DS and 33.4-fold speedup for TMP problem 8. The key to note is that none of the original techniques worked within the 20000 seconds limit, and thus that EVA+DDB runs in less than 100 seconds and often is not visible on the chart shows the significant speedups obtained by EVA+DDB.

Second set of experiments illustrate the utility of EVA in other kinds of problems that do not have all the constraints of TMP. We considered problems that did not have any rewards of negative infinity and where quality bound was desired for a given starting belief point. For this, we provide comparisons with approximate approaches that provide quality bounds. We experimented with five problems: Tiger-grid, Hallway, Hallway2, Aircraft (from Anthony Cassandra's website) and Scotland yard[3]. Figure 5.4 provides comparisons against PBVI[4]. In Figure 5.4, the x-axis indicates the problem, and the y-axis indicates the time taken to solution on a log scale. For each problem, the first bar is the time for solving the NLP(computing error bound) in PBVI; the second and third bars are run-times of PBVI and EVA for a fixed error bound; the fourth bar is the run-time for EVA for a tighter bound (half of the bound used for bars 2 and 3). The time taken by PBVI for each problem is the sum of first and second bars.

The first aspect of comparison is the time overhead in computing the error bounds. In EVA, error bound computation is negligible as it requires only a multiplication and hence is not presented in the figure. However for PBVI, it can be noted from first bars of Figure 5.4 that this takes a non-trivial amount of time and in some cases is comparable to the time taken by PBVI. For instance in Hallway2, the NLP (first bar) takes 143 seconds, which is 1/4 of PBVI's run-time (second bar). More importantly, the error bound computation time is comparable to the time taken by EVA.

The second point of the study is the run-time performance of the actual algorithms (not including the time taken for error bound computation). Due to the dependence of point-based algorithms (PBVI and HSVI) on the starting belief point, all results for PBVI are averaged over ten randomly generated starting beliefs. Furthermore, to avoid punishing PBVI (in terms of run-time) for planning multiple times, we removed the anytime nature of PBVI, i.e., made it to plan for a set of belief points (computed according to the belief point selection heuristic from [Pineau et al., 2003]) that provides the given error bound.

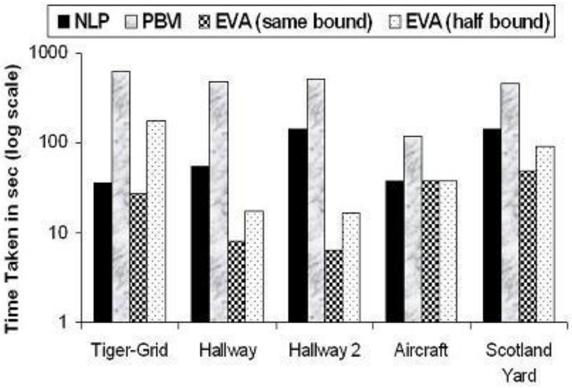


Figure 5.4: Run time comparison of EVA and PBVI

[3] Problem inspired from scotland yard game (216 s,16 a,6 o)

[4] We do not have results with HSVI2, an approach that is shown to better than PBVI. However, it should be noted that the run-time and quality results presented here are only to indicate that EVA can provide comparable performance to existing approaches while providing quality bound advantages

Second and third bars for each problem in Figure 5.4 provide this run-time comparison of EVA and PBVI for a given quality bound on the solution. It shows that EVA outperforms PBVI. For instance, the speedup obtained with EVA is 59.6-fold for Hallway. We also experimented with the "Tag" problem from [Pineau et al., 2003]. PBVI could not finish within the pre-specified limit of 2000 seconds, while EVA terminated within 700 seconds with a quality of -9.19(approximately equal to the value reported in [Pineau et al., 2003]).

	PBVI	**EVA (same bound)**	**EVA (same time)**
Tiger-Grid	-1.692	-1.62	-1.420
Hallway	0.122	0.118	0.267
Hallway2	0.038	0.027	0.08
Aircraft	7.416	7.416	7.416
Scotland Yard	0.073	-0.377	0.214

Table 5.1: Comparison of expected value for PBVI and EVA

In PBVI, there is no clear dominance of one belief-point-selection heuristic over the others [Pineau et al., 2003; Pineau and Gordon, 2005] for all the problems. However, there is dominance of certain heuristics on some problems in terms of both quality and time to solution. To account for this, we provide a comparison of PBVI, assuming that it provided a much tighter bound (half of the actual bound). In other words, given that a heuristic may improve PBVI solution quality by 100look at half of the actual bound for a comparison. Values in the fourth bar (for each problem) of Figure 5.4 indicate that EVA outperforms PBVI even in that case. For instance, there is still a 27.3-fold speedup for hallway.

Table 5.1 presents the third aspect of comparison between PBVI and EVA: actual solution quality. For the same error bound on solution quality, PBVI (column 2) performs better for Hallway, Hallway2 and Scotland Yard, while EVA (column 3) performs better for Tiger-Grid. However, if the restriction is on time to solution, EVA (column 4) obtains higher quality than PBVI in all five problems. For instance, PBVI obtains a quality of 0.122, while EVA obtains a quality of 0.267 for the Hallway problem. Thus the key point made here is that EVA runs faster for the same quality bound, but the actual solution quality is not always better than PBVI. However for a given time to solution, EVA obtains better quality than PBVI in all the problems.

Chapter 6

Exploiting interaction structure in Distributed POMDPs

In this chapter, I continue the theme of exploiting structure for efficiency, but now in distributed POMDPs. The structure exploited is in the interactions between the agents of a distributed POMDP. Earlier, researchers have attempted two different approaches to address the complexity of distributed POMDPs. First type of algorithms sacrifice global optimality and instead concentrate on local optimality [Nair et al., 2003a; Peshkin et al., 2000a]. On the other hand, the second kind of approaches have focused on restricted types of domains, e.g. with transition independence or collective observability [Becker et al., 2003, 2004]. While these approaches have led to useful advances, the complexity of the distributed POMDP problem has limited most experiments to a central policy generator planning for just two agents.

We introduce a third complementary approach called Networked Distributed POMDPs (ND-POMDPs), that is motivated by domains such as distributed sensor nets [Lesser et al., 2003], distributed UAV teams and distributed satellites, where an agent team must coordinate under uncertainty, but with agents having strong locality in their interactions. For example, within a large distributed sensor net, small subsets of sensor agents must coordinate to track targets. To exploit such local interactions, ND-POMDPs combine the planning under uncertainty of POMDPs with the local agent interactions of distributed constraint optimization (DCOP) [Modi et al., 2003b; Maheswaran et al., 2004; Yokoo and Hirayama, 1996]. DCOPs have successfully exploited limited agent interactions in multiagent systems, with over a decade of algorithm development. Distributed POMDPs benefit by building upon such algorithms that enable distributed planning, and provide algorithmic guarantees. DCOPs benefit by enabling (distributed) planning under uncertainty — a key DCOP deficiency in practical applications such as sensor nets [Lesser et al., 2003].

Taking inspiration from DCOP algorithms, we provide two algorithms for ND-POMDPs, a locally optimal algorithm, LID-JESP and a global optimal algorithm, GOA. First, within LID-JESP we present two ways of exploiting the locality of interaction, namely DBA/DSA and HLD. DBA/DSA exploits the external interaction structure, by combining the existing JESP algorithm of Nair *et al.* [Nair et al., 2003a] and DCOP algorithms, *Distributed Breakout Algorithm (DBA) and its stochastic variant, Distributed Stochastic Algorithm(DSA)* [Yokoo and Hirayama, 1996]. This approach thus combines the dynamic programming of JESP with the innovation that it uses distributed policy generation instead of JESP's centralized policy generation. On the other hand, *hyper-link-based decomposition* (HLD) exploits the structure introduced inside an agent, because of the interaction graph. Concretely, this method works by decomposing each agent's local optimization problem into loosely-coupled optimization problems for each hyper-link. This allows us to further exploit the locality of interaction, resulting in faster run times for both DBA and DSA without any loss in solution quality.

Finally, by empirically comparing the performance of the algorithm with benchmark algorithms that do not exploit network structure, we illustrate the gains in efficiency made possible by exploiting network structure in ND-POMDPs. Through detailed experiments, we show how this can result in speedups without sacrificing on solution quality. We also present detailed complexity results that indicate the difference introduced because of exploiting interaction structure.

6.1 ND-POMDPs

We define an ND-POMDP to be a specialization of MTDP as follows. In particular, we define ND-POMDP as a group Ag of n agents as a tuple $\langle S, A, P, \Omega, O, R, b\rangle$, where $S = \times_{1 \le i \le n} S_i \times S_u$ is the set of world states. S_i refers to the set of local states of agent i and S_u is the set of unaffectable states. Unaffectable state refers to that part of the world state that cannot be affected by the agents' actions, e.g. environmental factors like target locations that no agent can control. $A = \times_{1 \le i \le n} A_i$ is the set of joint actions, where A_i is the set of action for agent i.

We assume a *transition independent* distributed POMDP model, where the transition function is defined as $P(s, a, s') = P_u(s_u, s'_u) \cdot \prod_{1 \le i \le n} P_i(s_i, s_u, a_i, s'_i)$, where $a = \langle a_1, \ldots, a_n\rangle$ is the joint action performed in state $s = \langle s_1, \ldots, s_n, s_u\rangle$ and $s' = \langle s'_1, \ldots, s'_n, s'_u\rangle$is the resulting state. Agent i's transition function is defined as $P_i(s_i, s_u, a_i, s'_i) = \Pr(s'_i|s_i, s_u, a_i)$ and the unaffectable transition function is defined as $P_u(s_u, s'_u) = \Pr(s'_u|s_u)$. Becker *et al.* [Becker et al., 2004] also relied on transition independence, and Goldman and Zilberstein [Goldman and Zilberstein, 2004] introduced the possibility of uncontrollable state features. In both works, the authors assumed that the state is *collectively observable*, an assumption that does not hold for our domains of interest.

$\Omega = \times_{1 \le i \le n} \Omega_i$ is the set of joint observations where Ω_i is the set of observations for agents i. We make an assumption of *observational independence*, i.e., we define the joint observation function as $O(s, a, \omega) = \prod_{1 \le i \le n} O_i(s_i, s_u, a_i, \omega_i)$, where $s = \langle s_1, \ldots, s_n, s_u \rangle$, $a = \langle a_1, \ldots, a_n \rangle$, $\omega = \langle \omega_1, \ldots, \omega_n \rangle$, and $O_i(s_i, s_u, a_i, \omega_i) = \Pr(\omega_i | s_i, s_u, a_i)$.

The reward function, R, is defined as $R(s, a) = \sum_l R_l(s_{l1}, \ldots, s_{lk}, s_u, \langle a_{l1}, \ldots, a_{lk} \rangle)$, where each l could refer to any sub-group of agents and $k = |l|$. In the sensor grid example, the reward function is expressed as the sum of rewards between sensor agents that have overlapping areas ($k = 2$) and the reward functions for an individual agent's cost for sensing ($k = 1$). Based on the reward function, we construct an *interaction hypergraph* where a hyper-link, l, exists between a subset of agents for all R_l that comprise R. *Interaction hypergraph* is defined as $G = (Ag, E)$, where the agents, Ag, are the vertices and $E = \{l | l \subseteq Ag \wedge R_l \text{ is a component of } R\}$ are the edges. *Neighborhood* of i is defined as $N_i = \{j \in Ag | j \neq i \wedge (\exists l \in E,\ i \in l \wedge j \in l)\}$. $S_{N_i} = \times_{j \in N_i} S_j$ refers to the states of i's neighborhood. Similarly we define $A_{N_i} = \times_{j \in N_i} A_j$, $\Omega_{N_i} = \times_{j \in N_i} \Omega_j$, $P_{N_i}(s_{N_i}, a_{N_i}, s'_{N_i}) = \prod_{j \in N_i} P_j(s_j, a_j, s'_j)$, and $O_{N_i}(s_{N_i}, a_{N_i}, \omega_{N_i}) = \prod_{j \in N_i} O_j(s_j, a_j, \omega_j)$.

b, the distribution over the initial state, is defined as $b(s) = b_u(s_u) \cdot \prod_{1 \le i \le n} b_i(s_i)$ where b_u and b_i refer to the distributions over initial unaffectable state and i's initial state, respectively. We define $b_{N_i} = \prod_{j \in N_i} b_j(s_j)$. We assume that b is available to all agents (although it is possible to refine our model to make available to agent i only b_u, b_i and b_{N_i}). The goal in ND-POMDP is to compute joint policy $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ that maximizes the team's expected reward over a finite horizon T starting from b. π_i refers to the individual policy of agent i and is a mapping from the set of observation histories of i to A_i. π_{N_i} and π_l refer to the joint policies of the agents in N_i and hyper-link l respectively.

ND-POMDP can be thought of as an *n-ary* DCOP where the variable at each node is an individual agent's policy. The reward component R_l where $|l| = 1$ can be thought of as a local constraint while the reward component R_l where $l > 1$ corresponds to a non-local constraint in the constraint graph. In the next section, we push this analogy further by taking inspiration from the DBA algorithm [Yokoo and Hirayama, 1996], an algorithm for distributed constraint satisfaction, to develop an algorithm for solving ND-POMDPs.

The following proposition shows that given a factored reward function and the assumptions of transitional and observational independence, the resulting value function can be factored as well into value functions for each of the edges in the interaction hypergraph.

Proposition 5 *Given transitional and observational independence and* $R(s,a) = \sum_{l \in E} R_l(s_{l1},$ $\ldots, s_{lk}, s_u, \langle a_{l1}, \ldots, a_{lk} \rangle)$,

$$V^t_\pi(s^t, \vec{\omega}^t) = \sum_{l \in E} V^t_{\pi_l}(s^t_{l1}, \ldots, s^t_{lk}, s^t_u, \vec{\omega}^t_{l1}, \ldots \vec{\omega}^t_{lk}) \tag{6.1}$$

where $V^t_\pi(s^t, \vec{\omega})$ *is the expected reward from the state* s^t *and joint observation history* $\vec{\omega}^t$ *for executing policy* π, *and* $V^t_{\pi_l}(s^t_{l1}, \ldots, s^t_{lk}, s^t_u, \vec{\omega}^t_{l1}, \ldots \vec{\omega}^t_{lk})$ *is the expected reward for executing* π_l *accruing from the component* R_l.

Proof: Proof is by mathematical induction. Proposition holds for $t = T - 1$ (no future reward). Assume it holds for $t = \tau$ where $1 \leq \tau < T - 1$. Thus,

$$V^\tau_\pi(s^\tau, \vec{\omega}^\tau) = \sum_{l \in E} V^\tau_{\pi_l}(s^\tau_{l1}, \ldots, s^\tau_{lk}, s^\tau_u, \vec{\omega}^\tau_{l1}, \ldots \vec{\omega}^\tau_{lk})$$

We introduce the following abbreviations:

$$\begin{aligned}
p^t_i &\overset{\triangle}{=} P_i(s^t_i, s^t_u, \pi_i(\vec{\omega}^t_i), s^{t+1}_i) \cdot O_i(s^{t+1}_i, s^{t+1}_u, \pi_i(\vec{\omega}^t_i), \omega^{t+1}_i) \\
p^t_u &\overset{\triangle}{=} P_i(s^t_u, s^{t+1}_u) \\
r^t_l &\overset{\triangle}{=} R_l(s^t_{l1}, \ldots, s^t_{lk}, s^t_u, \pi_{l1}(\vec{\omega}^t_{l1}), \ldots, \pi_{lk}(\vec{\omega}^t_{lk})) \\
v^t_l &\overset{\triangle}{=} V^t_{\pi_l}(s^t_{l1}, \ldots, s^t_{lk}, s^t_u, \vec{\omega}^t_{l1}, \ldots \vec{\omega}^t_{lk})
\end{aligned}$$

We show that proposition holds for $t = \tau - 1$,

$$\begin{aligned}
V^{\tau-1}_\pi(s^{\tau-1}, \vec{\omega}^{\tau-1}) &= \sum_{l \in E} r^{\tau-1}_l + \sum_{s^\tau, \omega^\tau} p^{\tau-1}_u p^{\tau-1}_1 \ldots p^{\tau-1}_n \sum_{l \in E} v^\tau_l \\
= \sum_{l \in E} (r^{\tau-1}_l + &\sum_{s^\tau_{l1}, \ldots, s^\tau_{lk}, s^\tau_u, \omega^\tau_{l1}, \ldots, \omega^\tau_{lk}} p^{\tau-1}_{l1} \ldots p^{\tau-1}_{lk} p^{\tau-1}_u v^\tau_l) = \sum_{l \in E} v^{\tau-1}_l \square
\end{aligned}$$

We define *local neighborhood utility* of agent i as the expected reward for executing joint policy π accruing due to the hyper-links that contain agent i:

$$\begin{aligned}
V_\pi[N_i] = &\sum_{s_i, s_{N_i}, s_u} b_u(s_u) \cdot b_{N_i}(s_{N_i}) \cdot b_i(s_i) \cdot \\
&\sum_{l \in E \ s.t. \ i \in l} V^0_{\pi_l}(s_{l1}, \ldots, s_{lk}, s_u, \langle\rangle, \ldots, \langle\rangle)
\end{aligned} \tag{6.2}$$

Proposition 6 Locality of interaction: *The local neighborhood utilities of agent i for joint policies π and π' are equal ($V_{\pi}[N_i] = V_{\pi'}[N_i]$) if $\pi_i = \pi'_i$ and $\pi_{N_i} = \pi'_{N_i}$.*

Proof sketch: Equation 6.2 sums over $l \in E$ such that $i \in l$, and hence any change of the policy of an agent $j \notin i \cup N_i$ cannot affect $V_{\pi}[N_i]$. Thus, any such policy assignment, π' that has different policies for only non-neighborhood agents, has equal value as $V_{\pi}[N_i]$.$\Box$

Thus, increasing the local neighborhood utility of agent i cannot reduce the local neighborhood utility of agent j if $j \notin N_i$. Hence, while trying to find best policy for agent i given its neighbors' policies, we do not need to consider non-neighbors' policies. This is the property of *locality of interaction* that is used in later sections.

6.2 Locally Optimal Policy Generation, LID-JESP

The locally optimal policy generation algorithm called LID-JESP (Locally interacting distributed joint equilibrium search for policies) is based on the DBA algorithm [Yokoo and Hirayama, 1996] and JESP [Nair et al., 2003a]. In this algorithm (see Algorithm 5), each agent tries to improve its policy with respect to its neighbors' policies in a distributed manner similar to DBA. Initially each agent i starts with a random policy and exchanges its policies with its neighbors (lines 3-4). It then computes its local neighborhood utility (see Equation 6.2) with respect to its current policy and its neighbors' policies. Agent i then tries to improve upon its current policy by calling function GETVALUE (see Algorithm 7), which returns the local neighborhood utility of agent i's best response to its neighbors' policies. This algorithm is described in detail below. Agent i then computes the gain (always ≥ 0 because at worst GETVALUE will return the same value as $prevVal$) that it can make to its local neighborhood utility, and exchanges its gain with its neighbors (lines 8-11). If i's gain is greater than any of its neighbors' gain[1], i changes its policy (FINDPOLICY) and sends its new policy to all its neighbors. This process of trying to improve the local neighborhood utility is continued until termination. Termination detection is based on using a termination counter to count the number of cycles where $gain_i$ remains $= 0$. If its gain is greater than zero the termination counter is reset. Agent i then exchanges its termination counter with its neighbors and set its counter to the minimum of its counter and its neighbors' counters. Agent i will terminate if its termination counter becomes equal to the diameter of the interaction hypergraph.

[1] The function $\mathbf{argmax}_j$ disambiguates between multiple j corresponding to the same max value by returning the lowest j.

Figure 6.2 provides an execution of the LID-JESP algorithm for a small example of three sensor agents connected in a chain. The execution begins with agents A1, A2, and A3 taking policies (randomly) p1, p2 and p3 respectively.

Algorithm 5 LID-JESP(i, ND-POMDP)

```
Compute interaction hypergraph and $N_i$
$d \leftarrow$ diameter of hypergraph, $terminationCtr_i \leftarrow 0$
$\pi_i \leftarrow$ randomly selected policy, $prevVal \leftarrow 0$
Exchange $\pi_i$ with $N_i$
while $terminationCtr_i < d$ do
    for all $s_i, s_{N_i}, s_u$ do
        $B_i^0(\langle s_u, s_i, s_{N_i}, \langle\rangle\rangle) \leftarrow b_u(s_u) \cdot b_i(s_i) \cdot b_{N_i}(s_{N_i})$
        $prevVal \overset{+}{\leftarrow} B_i^0(\langle s_u, s_i, s_{N_i}, \langle\rangle\rangle) \cdot \textsc{Evaluate}(i, s_i, s_u, s_{N_i}, \pi_i, \pi_{N_i}, \langle\rangle, \langle\rangle, 0, T)$
    end for
    $gain_i \leftarrow \textsc{GetValue}(i, B_i^0, \pi_{N_i}, 0, T) - prevVal$
    if $gain_i > 0$ then $terminationCtr_i \leftarrow 0$
    else $terminationCtr_i \overset{+}{\leftarrow} 1$
    Exchange $gain_i$,$terminationCtr_i$ with $N_i$
    $terminationCtr_i \leftarrow \mathbf{min}_{j \in N_i \cup \{i\}} terminationCtr_j$
    $maxGain \leftarrow \max_{j \in N_i \cup \{i\}} gain_j$
    $winner \leftarrow$ argmax$_{j \in N_i \cup \{i\}} gain_j$
    if $maxGain > 0$ and $i = winner$ then
        $\textsc{FindPolicy}(i, b, \langle\rangle, \pi_{N_i}, 0, T)$
        Communicate $\pi_i$ with $N_i$
    else if $maxGain > 0$ then
        Receive $\pi_{winner}$ from $winner$ and update $\pi_{N_i}$
    end if
end while
return $\pi_i$
```

6.2.1 Finding Best Response

The algorithm, GetValue, for computing the best response is a dynamic-programming approach similar to that used in JESP. Here, we define an *episode* of agent i at time t as $e_i^t = \left\langle s_u^t, s_i^t, s_{N_i}^t, \vec{\omega}_{N_i}^t \right\rangle$. Treating episode as the state, results in a single agent POMDP, where the transition function and observation function can be defined as:

$$\begin{aligned} P'(e_i^t, a_i^t, e_i^{t+1}) = & P_u(s_u^t, s_u^{t+1}) \cdot P_i(s_i^t, s_u^t, a_i^t, s_i^{t+1}) \cdot P_{N_i}(s_{N_i}^t, \\ & s_u^t, a_{N_i}^t, s_{N_i}^{t+1}) \cdot O_{N_i}(s_{N_i}^{t+1}, s_u^{t+1}, a_{N_i}^t, \omega_{N_i}^{t+1}) \\ O'(e_i^{t+1}, a_i^t, \omega_i^{t+1}) = & \, O_i(s_i^{t+1}, s_u^{t+1}, a_i^t, \omega_i^{t+1}) \end{aligned}$$

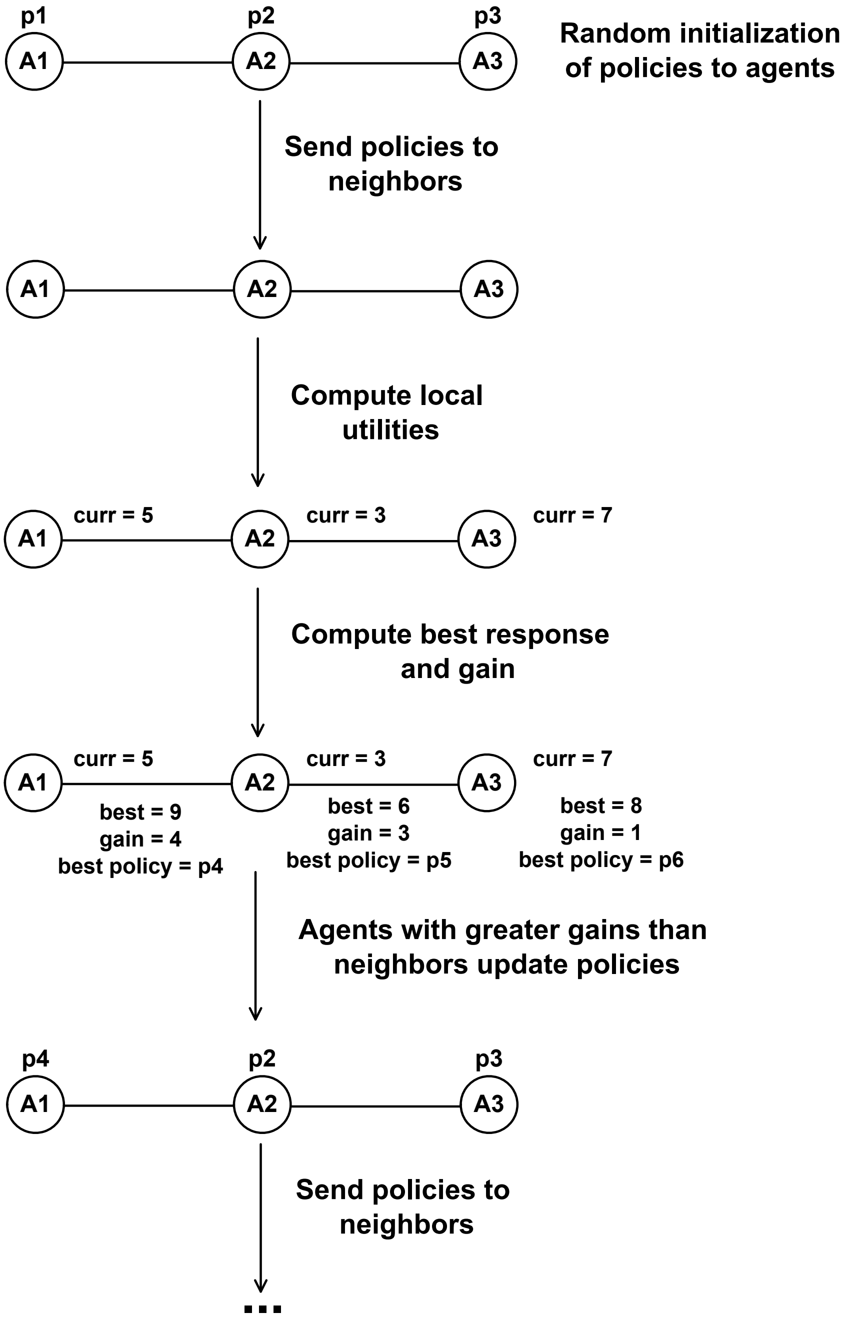


Figure 6.1: Sample execution trace of LID-JESP for a 3-agent chain

Algorithm 6 $\textsc{Evaluate}(i, s_i^t, s_u^t, s_{N_i}^t, \pi_i, \pi_{N_i}, \vec{\omega}_i^t, \vec{\omega}_{N_i}^t, t, T)$

```
$a_i \leftarrow \pi_i(\vec{\omega}_i^t)$, $a_{N_i} \leftarrow \pi_{N_i}(\vec{\omega}_{N_i}^t)$
$val \leftarrow \sum_{l \in E} R_l\left(s_{l1}^t, \ldots, s_{lk}^t, s_u^t, a_{l1}, \ldots, a_{lk}\right)$
**if** $t < T - 1$ **then**
    **for all** $s_i^{t+1}, s_{N_i}^{t+1}, s_u^{t+1}$ **do**
        **for all** $\omega_i^{t+1}, \omega_{N_i}^{t+1}$ **do**
            $val \overset{+}{\leftarrow} P_u(s_u^t, s_u^{t+1}) \cdot P_i(s_i^t, s_u^t, a_i, s_i^{t+1}) \cdot P_{N_i}(s_{N_i}^t, s_u^t, a_{N_i}, s_{N_i}^{t+1}) \cdot O_i(s_i^{t+1}, s_u^{t+1}, a_i, \omega_i^{t+1}) \cdot O_{N_i}(s_{N_i}^{t+1}, s_u^{t+1}, a_{N_i}, \omega_{N_i}^{t+1}) \cdot \textsc{Evaluate}(i, s_i^{t+1}, s_u^{t+1}, s_{N_i}^{t+1}, \pi_i, \pi_{N_i}, \left\langle \vec{\omega}_i^t, \omega_i^{t+1} \right\rangle, \left\langle \vec{\omega}_{N_i}^t, \omega_{N_i}^{t+1} \right\rangle, t+1, T)$
        **end for**
    **end for**
**end if**
**return** $val$
```

A multiagent belief state for an agent i given the distribution over the initial state, $b(s)$ is defined as:

$$B_i^t(e_i^t) = \Pr(s_u^t, s_i^t, s_{N_i}^t, \vec{\omega}_{N_i}^t | \vec{\omega}_i^t, \vec{a}_i^{t-1}, b)$$

The initial multiagent belief state for agent i, B_i^0, can be computed from b as follows:

$$B_i^0(\langle s_u, s_i, s_{N_i}, \langle\rangle\rangle) \leftarrow b_u(s_u) \cdot b_i(s_i) \cdot b_{N_i}(s_{N_i})$$

We can now compute the value of the best response policy via GETVALUE using the following equation (see Algorithm 7):

$$V_i^t(B_i^t) = \max_{a_i \in A_i} V_i^{a_i,t}(B_i^t) \tag{6.3}$$

Algorithm 7 $\textsc{GetValue}(i, B_i^t, \pi_{N_i}, t, T)$

```
**if** $t \geq T$ **then return** 0
**if** $V_i^t(B_i^t)$ is already recorded **then return** $V_i^t(B_i^t)$
$best \leftarrow -\infty$
**for all** $a_i \in A_i$ **do**
    $value \leftarrow \textsc{GetValueAction}(i, B_i^t, a_i, \pi_{N_i}, t, T)$
    record $value$ as $V_i^{a_i,t}(B_i^t)$
    **if** $value > best$ **then** $best \leftarrow value$
**end for**
record $best$ as $V_i^t(B_i^t)$
**return** $best$
```

The function, $V_i^{a_i,t}$, can be computed using GETVALUEACTION(see Algorithm 8) as follows:

$$V_i^{a_i,t}(B_i^t) = \sum_{e_i^t} B_i^t(e_i^t) \sum_{l \in E \ s.t. \ i \in l} R_l(s_{l1}, \ldots, s_{lk}, s_u, \langle a_{l1}, \ldots, a_{lk} \rangle) \\ + \sum_{\omega_i^{t+1} \in \Omega_1} \Pr(\omega_i^{t+1} | B_i^t, a_i) \cdot V_i^{t+1}\left(B_i^{t+1}\right) \tag{6.4}$$

B_i^{t+1} is the belief state updated after performing action a_i and observing ω_i^{t+1} and is computed using the function UPDATE (see Algorithm 9). Agent i's policy is determined from its value function $V_i^{a_i,t}$ using the function FINDPOLICY (see Algorithm 10).

Algorithm 8 GETVALUEACTION$(i, B_i^t, a_i, \pi_{N_i}, t, T)$

```
$value \leftarrow 0$
for all $e_i^t = \langle s_u^t, s_i^t, s_{N_i}^t, \vec{\omega}_{N_i}^t \rangle$ s.t. $B_i^t(e_i^t) > 0$ do
    $a_{N_i} \leftarrow \pi_{N_i}(\vec{\omega}_{N_i}^t)$
    $reward \leftarrow \sum_{l \in E} R_l\left(s_{l1}^t, \ldots, s_{lk}^t, s_u^t, a_{l1}, \ldots, a_{lk}\right)$
    $value \overset{+}{\leftarrow} B_i^t(e_i^t) \cdot reward$
end for
if $t < T - 1$ then
    for all $\omega_i^{t+1} \in \Omega_i$ do
        $B_i^{t+1} \leftarrow$ UPDATE$(i, B_i^t, a_i, \omega_i^{t+1}, \pi_{N_i})$
        $prob \leftarrow 0$
        for all $s_u^t, s_i^t, s_{N_i}^t$ do
            for all $e_i^{t+1} = \langle s_u^{t+1}, s_i^{t+1}, s_{N_i}^{t+1}, \langle \vec{\omega}_{N_i}^t, \omega_{N_i}^{t+1} \rangle \rangle$ s.t. $B_i^{t+1}(e_i^{t+1}) > 0$ do
                $a_{N_i} \leftarrow \pi_{N_i}(\vec{\omega}_{N_i}^t)$
                $prob \overset{+}{\leftarrow} B_i^t(e_i^t) \cdot P_u(s_u^t, s_u^{t+1}) \cdot P_i(s_i^t, s_u^t, a_i, s_i^{t+1}) \cdot P_{N_i}(s_{N_i}^t, s_u^t, a_{N_i}, s_{N_i}^{t+1}) \cdot O_i(s_i^{t+1}, s_u^{t+1}, a_i, \omega_i^{t+1}) \cdot O_{N_i}(s_{N_i}^{t+1}, s_u^{t+1}, a_{N_i}, \omega_{N_i}^{t+1})$
            end for
        end for
        $value \overset{+}{\leftarrow} prob \cdot$ GETVALUE$(i, B_i^{t+1}, \pi_{N_i}, t+1, T)$
    end for
end if
return $value$
```

Algorithm 9 UPDATE$(i, B_i^t, a_i, \omega_i^{t+1}, \pi_{N_i})$

```
for all $e_i^{t+1} = \langle s_u^{t+1}, s_i^{t+1}, s_{N_i}^{t+1}, \langle \vec{\omega}_{N_i}^t, \omega_{N_i}^{t+1} \rangle \rangle$ do
    $B_i^{t+1}(e_i^{t+1}) \leftarrow 0$, $a_{N_i} \leftarrow \pi_{N_i}(\vec{\omega}_{N_i}^t)$
    for all $s_u^t, s_i^t, s_{N_i}^t$ do
        $B_i^{t+1}(e_i^{t+1}) \overset{+}{\leftarrow} B_i^t(e_i^t) \cdot P_u(s_u^t, s_u^{t+1}) \cdot P_i(s_i^t, s_u^t, a_i, s_i^{t+1}) \cdot P_{N_i}(s_{N_i}^t, s_u^t, a_{N_i}, s_{N_i}^{t+1}) \cdot O_i(s_i^{t+1}, s_u^{t+1}, a_i, \omega_i^{t+1}) \cdot O_{N_i}(s_{N_i}^{t+1}, s_u^{t+1}, a_{N_i}, \omega_{N_i}^{t+1})$
    end for
end for
normalize $B_i^{t+1}$
return $B_i^{t+1}$
```

Algorithm 10 FINDPOLICY$(i, B_i^t, \vec{\omega_i}^t, \pi_{N_i}, t, T)$

```
$a_i^* \leftarrow \text{argmax}_{a_i} V_i^{a_i,t}(B_i^t)$, $\pi_i(\vec{\omega_i}^t) \leftarrow a_i^*$
**if** $t < T - 1$ **then**
    **for all** $\omega_i^{t+1} \in \Omega_i$ **do**
        $B_i^{t+1} \leftarrow$ UPDATE$(i, B_i^t, a_i^*, \omega_i^{t+1}, \pi_{N_i})$
        FINDPOLICY$(i, B_i^{t+1}, \langle \vec{\omega_i}^t, \omega_i^{t+1} \rangle, \pi_{N_i}, t+1, T)$
    **end for**
**end if**
**return**
```

6.2.2 Correctness Results

Proposition 7 *When applying LID-JESP, the global utility is strictly increasing until local optimum is reached.*

Proof sketch By construction, only non-neighboring agents can modify their policies in the same cycle. Agent i chooses to change its policy if it can improve upon its local neighborhood utility $V_\pi[N_i]$. From Equation 6.2, increasing $V_\pi[N_i]$ results in an increase in global utility. By locality of interaction, if an agent $j \notin i \cup N_i$ changes its policy to improve its local neighborhood utility, it will not affect $V_\pi[N_i]$ but will increase global utility. Thus with each cycle global utility is strictly increasing until local optimum is reached. □

Proposition 8 *LID-JESP will terminate within d (= diameter) cycles iff agent are in a local optimum.*

Proof: Assume that in cycle c, agent i terminates ($terminationCtr_i = d$) but agents are not in a local optimum. In cycle $c - d$, there must be at least one agent j who can improve, i.e., $gain_j > 0$ (otherwise, agents are in a local optimum in cycle $c - d$ and no agent can improve later). Let d_{ij} refer to the shortest path distance between agents i and j. Then, in cycle $c - d + d_{ij}$ ($\leq c$), $terminationCtr_i$ must have been set to 0. However, $terminationCtr_i$ increases by at most one in each cycle. Thus, in cycle c, $terminationCtr_i \leq d - d_{ij}$. If $d_{ij} \geq 1$, in cycle c, $terminationCtr_i < d$. Also, if $d_{ij} = 0$, i.e., in cycle $c - d$, $gain_i > 0$, then in cycle $c - d + 1$, $terminationCtr_i = 0$, thus, in cycle c, $terminationCtr_i < d$. In either case, $terminationCtr_i \neq d$. By contradiction, if LID-JESP terminates then agents must be in a local optimum.

In the reverse direction, if agents reach a local optimum, $gain_i = 0$ henceforth. Thus, $terminationCtr_i$ is never reset to 0 and is incremented by 1 in every cycle. Hence, after d cycles, $terminationCtr_i = d$ and agents terminate. □

Proposition 7 shows that the agents will eventually reach a local optimum and Proposition 8 shows that the LID-JESP will terminate if and only if agents are in a local optimum. Thus, LID-JESP will correctly find a locally optimum and will terminate.

6.3 Stochastic LID-JESP (SLID-JESP)

One of the criticisms of LID-JESP is that if an agent is the winner (maximum reward among its neighbors), then its precludes its neighbors from changing their policies too in that cycle. In addition, it will sometimes prevent its neighbor's neighbors (and may be their neighbors and so on) from changing their policies in that cycle even if though they are actually independent. For example, consider the execution trace from Figure 6.2, where $gain_{A1} > gain_{A2} > gain_{A3}$. In this situation, only $A1$ changes its policy in that cycle. However, $A3$ should have been able to changed its policy too because it does not depend on $A1$. This realization that LID-JESP allows limited parallelism led us to come up with a stochastic version of LID-JESP, SLID-JESP (Algorithm 11).

The key difference between LID-JESP and SLID-JESP is that in SLID-JESP is that if an agent i can improve its local neighborhood utility (i.e. $gain_i > 0$), it will do so with probability p, a predefined threshold probability (see lines 14-17). Note, that unlike LID-JESP, an agent's decision to change its policy does not depend on its neighbors' gain messages. However, we still agents continue to communicate their gain messages to their neighbors to determine whether the algorithm has terminated.

Since there has been no change to the termination detection approach and the way gain is computed, the following propositions from LID-JESP hold for SLID-JESP as well.

Proposition 9 *When applying SLID-JESP, the global utility is strictly increasing until local optimum is reached.*

Proposition 10 *SLID-JESP will terminate within d (= diameter) cycles iff agent are in a local optimum.*

Proposition 7 shows that the agents will eventually reach a local optimum and Proposition 8 shows that the SLID-JESP will terminate if and only if agents are in a local optimum. Thus, SLID-JESP will correctly find a locally optimum and will terminate.

Algorithm 11 SLID-JESP(i, ND-POMDP, p)

```
0: {lines 1-4 same a LID-JESP}
5: while $terminationCtr_i < d$ do {lines 6-13 same as LID-JESP}
14:    if RANDOM() $< p$ and $gain_i > 0$ then
15:       FINDPOLICY($i, b, \langle\rangle, \pi_{N_i}, 0, T$)
16:       Communicate $\pi_i$ with $N_i$
17:    end if
18:    Receive $\pi_j$ from all $j \in N_i$ that changed their policies
19: end while
20: return $\pi_i$
```

6.4 Hyper-link-based Decomposition (HLD)

Proposition 5 and Equation 6.2 indicate that the value function and the local neighborhood utility function can both be decomposed into components for each hyper-link in the *interaction hypergraph*. We developed the Hyper-link-based Decomposition (HLD) technique as a means to exploit this decomposability, in order to speedup the algorithms EVALUATE and GETVALUE.

We introduce the following definitions to ease the description of hyper-link-based decomposition. Let $E_i = \{l | l \in E \wedge i \in l\}$ be the subset of hyper-links that contain agent i. Note that $N_i = \cup_{l \in E_i} l - \{i\}$, i.e. the neighborhood of i contains all the agents in E_i except agent i itself. We define $S_l = \times_{j \in l} S_j$ refers to the states of agents in link l. Similarly we define $A_l = \times_{j \in l} A_j$, $\Omega_l = \times_{j \in l} \Omega_l$, $P_l(s_l, a_l, s'_l) = \prod_{j \in l} P_j(s_j, a_j, s'_j)$, and $O_l(s_l, a_l, \omega_l) = \prod_{j \in l} O_j(s_j, a_j, \omega_j)$. Further, we define $b_l = \prod_{j \in l} b_j(s_j)$, where b_j is the distribution over agent j's initial state.

Using the above definitions, we can rewrite Equation 6.2 as

$$V_\pi[N_i] = \sum_{l \in E_i} \sum_{s_l, s_u} b_u(s_u) \cdot b_l(s_l) \cdot V^0_{\pi_l}(s_l, s_u, \langle\rangle, \ldots, \langle\rangle) \tag{6.5}$$

EVALUATE-HLD (Algorithm 13) is used to compute the local neighborhood utility of a hyperlink l (inner loop of Equation 6.8). When the joint policy is completely specified, the expected reward from each hyper-link can be computed independently (as in EVALUATE-HLD). However, when trying to find the optimal best response, we cannot optimize on each hyper-link separately since in any belief state, an agent can perform only one action. The optimal best response in any belief state is the action that maximizes the sum of the expected rewards on each of its hyper-links.

The algorithm, GETVALUE-HLD, for computing the best response is a modification of the GETVALUE function that attempts to exploit the decomposability of the value function without violating the constraint that the same action must be applied to all the hyper-links in a particular belief state. Here, we define an *episode* of agent i for a hyper-link l at time t as

$e_{il}^t = \left\langle s_u^t, s_l^t, \vec{\omega}_{l-\{i\}}^t \right\rangle$. Treating episode as the state, the transition function and observation function can be defined as:

$$\begin{aligned} P'_{il}(e_{il}^t, a_i^t, e_{il}^{t+1}) =& P_u(s_u^t, s_u^{t+1}) \cdot P_l(s_l^t, s_u^t, a_l^t, s_l^{t+1}) \\ & \cdot O_{l-\{i\}}(s_{l-\{i\}}^{t+1}, s_u^{t+1}, a_{l-\{i\}}^t, \omega_{l-\{i\}}^{t+1}) \\ O'_{il}(e_i^{t+1}, a_i^t, \omega_i^{t+1}) =& O_i(s_i^{t+1}, s_u^{t+1}, a_i^t, \omega_i^{t+1}) \end{aligned}$$

where $a_{l-\{i\}}^t = \pi_{l-\{i\}}(\vec{\omega}_{l-\{i\}}^t)$. We can now define the multiagent belief state for an agent i with respect to hyper-link $l \in E_i$ as:

$$B_{il}^t(e_{il}^t) = \Pr(s_u^t, s_l^t, \vec{\omega}_{l-\{i\}}^t | \vec{\omega}_i^t, \vec{a}_i^{t-1}, b)$$

We redefine the multiagent belief state of agent i as :

$$B_i^t(e_i^t) = \{B_{il}^t(e_{il}^t) | l \in E_i\}$$

We can now compute the value of the best response policy using the following equation:

$$V_i^t(B_i^t) = \max_{a_i \in A_i} \left(\sum_{l \in E_i} V_{il}^{a_i,t}(B_i^t) \right) \tag{6.6}$$

The value of the best response policy for the link l can be computed as follows:

$$V_{il}^t(B_i^t) = V_{il}^{a_i^*,t}(B_i^t) \tag{6.7}$$

where $a_i^* = \arg\max_{a_i \in A_i} \left(\sum_{l \in E_i} V_{il}^{a_i,t}(B_i^t) \right)$. The function GETVALUE-HLD (see Algorithm 14) computes the term $V_{il}^t(B_i^t$ for all links $l \in E_i$.

The function, $V_{il}^{a_i,t}$, can be computed as follows:

$$\begin{aligned} V_{il}^{a_i,t}(B_i^t) =& \sum_{e_{il}^t} B_{il}^t(e_{il}^t) \cdot R_l(s_l, s_u, a_l) \\ & + \sum_{\omega_i^{t+1} \in \Omega_1} \Pr(\omega_i^{t+1} | B_i^t, a_i) \cdot V_{il}^{t+1}\left(B_i^{t+1}\right) \end{aligned} \tag{6.8}$$

The function GETVALUEACTION-HLD(see Algorithm 15) computes the above value for all links l. B_i^{t+1} is the belief state updated after performing action a_i and observing ω_i^{t+1} and is computed using the function UPDATE (see Algorithm 16). Agent i's policy is determined from its value function $V_i^{a_i,t}$ using the function FINDPOLICY (see Algorithm 17).

The reason why HLD will reduce the run time for finding the best response is that the optimal value function is computed for each linkly separately. This reduction in runtime is borne out by our complexity analysis and experimental results as well.

Algorithm 12 LID-JESP-HLD(i, ND-POMDP)

```
 0: {lines 1-4 same a LID-JESP}
 5: while $terminationCtr_i < d$ do
 6:   for all $s_u$ do
 7:     for all $l \in E_i$ do
 8:       for all $s_l \in S_l$ do
 9:         $B_{il}^0(\langle s_u, s_l, \langle\rangle\rangle) \leftarrow b_u(s_u) \cdot b_l(s_l)$
10:         $prevVal \overset{+}{\leftarrow} B_{il}^0(\langle s_u, s_l, \langle\rangle\rangle) \cdot$ EVALUATE-HLD$(l, s_l, s_u, \pi_l, \langle\rangle, 0, T)$
11:       end for
12:     end for
13:   end for
14:   $gain_i \leftarrow$ GETVALUE-HLD$(i, B_i^0, \pi_{N_i}, 0, T) - prevVal$
15:   if $gain_i > 0$ then $terminationCtr_i \leftarrow 0$
16:   else $terminationCtr_i \overset{+}{\leftarrow} 1$
17:   Exchange $gain_i$,$terminationCtr_i$ with $N_i$
18:   $terminationCtr_i \leftarrow \mathbf{min}_{j \in N_i \cup \{i\}} terminationCtr_j$
19:   $maxGain \leftarrow \max_{j \in N_i \cup \{i\}} gain_j$
20:   $winner \leftarrow$ argmax$_{j \in N_i \cup \{i\}} gain_j$
21:   if $maxGain > 0$ and $i = winner$ then
22:     FINDPOLICY-HLD$(i, B_i^0, \langle\rangle, \pi_{N_i}, 0, T)$
23:     Communicate $\pi_i$ with $N_i$
24:   else if $maxGain > 0$ then
25:     Receive $\pi_{winner}$ from $winner$ and update $\pi_{N_i}$
26:   end if
27: end while
28: return $\pi_i$
```

6.5 Complexity Results

The complexity of the finding the optimal best response for agent i for JESP (using the dynamic programming [Nair et al., 2003a]) is $O(|S|^2 \cdot |A_i|^T \cdot \prod_{j \in \{1 \ldots n\}} |\Omega_j|^T)$. Note that the complexity depends on the number world states $|S|$ and the number of possible observation histories of all the agents.

In contrast, the complexity of finding the optimal best response for i for LID-JESP (and SLID-JESP) is $O(\prod_{l \in E_i} [|S_u \times S_l|^2 \cdot |A_i|^T \cdot |\Omega_l|^T])$. It should be noted that in this case, the complexity depends on the number of states $|S_u|$, $|S_i|$ and $|S_{N_i}|$ and not on the number of states of any

Algorithm 13 EVALUATE-HLD$(l, s_l^t, s_u^t, \pi_l, \vec{\omega}_l^t, t, T)$

```
$a_l \leftarrow \pi_l(\vec{\omega}_l^t)$
$val \leftarrow R_l\,(s_l^t, s_u^t, a_l)$
if $t < T - 1$ then
    for all $s_l^{t+1}, s_u^{t+1}$ do
        for all $\omega_l^{t+1}$ do
            $val \overset{+}{\leftarrow} P_u(s_u^t, s_u^{t+1}) \cdot P_l(s_l^t, s_u^t, a_l, s_l^{t+1}) \cdot O_l(s_l^{t+1}, s_u^{t+1}, a_l, \omega_l^{t+1}) \cdot$ EVALUATE-HLD $\left(l, s_l^{t+1}, s_u^{t+1}, \pi_l, \left\langle \vec{\omega}_l^t, \omega_l^{t+1} \right\rangle, t+1, T\right)$
        end for
    end for
end if
return $val$
```

Algorithm 14 GETVALUE-HLD$(i, B_i^t, \pi_{N_i}, t, T)$

```
if $t \geq T$ then return $0$
if $V_{il}^t(B_i^t)$ is already recorded $\forall l \in E_i$ then return $[V_{il}^t(B_i^t)]_{l \in E_i}$
$bestSum \leftarrow -\infty$
for all $a_i \in A_i$ do
    $value \leftarrow$ GETVALUEACTION-HLD$(i, B_i^t, a_i, \pi_{N_i}, t, T)$
    $valueSum \leftarrow \sum_{l \in E_i} value[l]$
    record $valueSum$ as $V_i^{a_i,t}(B_i^t)$
    if $valueSum > bestSum$ then $best \leftarrow value, bestSum \leftarrow valueSum$
end for
for all $l \in E_i$ do
    record $best[l]$ as $V_{il}^t(B_i^t)$
end for
return $best$
```

Algorithm 15 GETVALUEACTION-HLD$(i, B_i^t, a_i, \pi_{N_i}, t, T)$

```
for all $l \in E_i$ do
    $value[l] \leftarrow 0$
    for all $e_{il}^t = \left\langle s_u^t, s_l^t, \vec{\omega}_{l-\{i\}}^t \right\rangle$ s.t. $B_{il}^t(e_{il}^t) > 0$ do
        $a_{l-\{i\}} \leftarrow \pi_{l-\{i\}}(\vec{\omega}_{l-\{i\}}^t)$
        $value[l] \overset{+}{\leftarrow} B_{il}^t(e_{il}^t) \cdot R_l\left(s_l^t, s_u^t, a_l\right)$
    end for
end for
if $t < T - 1$ then
    for all $\omega_i^{t+1} \in \Omega_i$ do
        for all $l \in E_i$ do
            $B_{il}^{t+1} \leftarrow$ UPDATE-HLD$(i, l, B_{il}^t, a_i, \omega_i^{t+1}, \pi_{l-\{i\}})$
            $prob[l] \leftarrow 0$
            for all $s_u^t, s_l^t$ do
                for all $e_{il}^{t+1} = \left\langle s_u^{t+1}, s_l^{t+1}, \left\langle \vec{\omega}_{l-\{i\}}^t, \omega_{l-\{i\}}^{t+1} \right\rangle \right\rangle$ s.t. $B_{il}^{t+1}(e_{il}^{t+1}) > 0$ do
                    $a_{l-\{i\}} \leftarrow \pi_{l-\{i\}}(\vec{\omega}_{l-\{i\}}^t)$
                    $prob[l] \overset{+}{\leftarrow} B_{il}^t(e_{il}^t) \cdot P_u(s_u^t, s_u^{t+1}) \cdot P_l(s_l^t, s_u^t, a_l, s_l^{t+1}) \cdot O_l(s_l^{t+1}, s_u^{t+1}, a_l, \omega_l^{t+1})$
                end for
            end for
        end for
        $futureValue \leftarrow$GETVALUE-HLD$(i, B_i^{t+1}, \pi_{N_i}, t+1, T)$
        for all $l \in E_i$ do
            $value[l] \overset{+}{\leftarrow} prob[l] \cdot futureValue[l]$
        end for
    end for
end if
return $value$
```

Algorithm 16 UPDATE-HLD$(i, l, B_{il}^t, a_i, \omega_i^{t+1}, \pi_{l-\{i\}})$

```
for all $e_{il}^{t+1} = \left\langle s_u^{t+1}, s_l^{t+1}, \left\langle \vec{\omega}_{l-\{i\}}^t, \omega_{l-\{i\}}^{t+1} \right\rangle \right\rangle$ do
    $B_{il}^{t+1}(e_{il}^{t+1}) \leftarrow 0$
    $a_{l-\{i\}} \leftarrow \pi_{l-\{i\}}(\vec{\omega}_{l-\{i\}}^t)$
    for all $s_u^t, s_l^t$ do
        $B_{il}^{t+1}(e_{il}^{t+1}) \overset{+}{\leftarrow} B_{il}^t(e_{il}^t) \cdot P_u(s_u^t, s_u^{t+1}) \cdot P_l(s_l^t, s_u^t, a_l, s_l^{t+1}) \cdot O_l(s_l^{t+1}, s_u^{t+1}, a_l, \omega_l^{t+1})$
    end for
end for
normalize $B_{il}^{t+1}$
return $B_{il}^{t+1}$
```

Algorithm 17 FINDPOLICY-HLD$(i, B_i^t, \vec{\omega_i}^t, \pi_{N_i}, t, T)$

```
$a_i^* \leftarrow \text{argmax}_{a_i} V_i^{a_i,t}(B_i^t)$
$\pi_i(\vec{\omega_i}^t) \leftarrow a_i^*$
if $t < T - 1$ then
    for all $\omega_i^{t+1} \in \Omega_i$ do
        for all $l \in E_i$ do
            $B_{il}^{t+1} \leftarrow$ UPDATE-HLD$(i, l, B_{il}^t, a_i^*, \omega_i^{t+1}, \pi_{l-\{i\}})$
        end for
        FINDPOLICY-HLD$(i, B_i^{t+1}, \langle \vec{\omega_i}^t, \omega_i^{t+1} \rangle, \pi_{N_i}, t+1, T)$
    end for
end if
return
```

non-neighboring agent. Similarly, the complexity depends on only the number of observation histories of i and its neighbors and not those of all the agents. This highlights the reason for why LID-JESP and SLID-JESP are superior to JESP for problems where locality of interaction can be exploited.

The complexity for computing optimal best response for i in LID-JESP with HLD (and SLID-JESP with HLD) is $O(\Sigma_{l \in E_i}[|S_u \times S_l|^2 \cdot |A_i|^T \cdot |\Omega_l|^T])$. Key difference of note compared to the complexity expression for LID-JESP, is the replacement of product,$\prod$ with a sum,Σ. Thus, as number of neighbors increases, difference between the two approaches increases.

Since JESP is a centralized algorithm, the best response function is performed for each agent serially. LID-JESP and SLID-JESP (with and without HLD), in contrast, are distributed algorithms, where each agent can be run in parallel on a different processor, further alleviating the large complexity of finding the optimal best response.

6.6 Locally Interacting - Global Optimal Algorithm (GOA)

GOA like the above algorithms also borrows from a DCOP algorithm. As opposed to the locally optimal DCOP algorithms used in LID-JESP and SLID-JESP, GOA borrows from an exact algorithm, DPOP (Distributed Pseudotree Optimization Procedure) and at present works only with binary interactions, i.e. edges linking two nodes. We start with a description of GOA applied to tree-structured interaction graphs, and then discuss its application to graphs with cycles.

DPOP dictates the functioning of message passing between the agents. The first phase is the UTIL propagation, where the utility messages, in this case values of policies, are passed up from the leaves to the root. Value for a policy at an agent is defined as the sum of best response values from its children and the joint policy reward associated with the parent policy. Thus, given a fixed policy for a parent node, GOA requires an agent to iterate through all its policies, finding the best response policy and returning the value to the parent — where to find the best policy, an agent requires its children to return their best responses to each of its policies. An agent stores the sum of best response values from its children, to avoid recalculation at the children. This UTIL propagation process is repeated at each level in the tree, until the root exhausts all its policies. In the second phase of VALUE propagation, where the optimal policies are passed down from the root till the leaves.

GOA takes advantage of the local interactions in the interaction graph, by pruning out unnecessary joint policy evaluations (associated with nodes not connected directly in the tree). Since the interaction graph captures all the reward interactions among agents and as this algorithm iterates through all the joint policy evaluations possible with the interaction graph, this algorithm yields an optimal solution.

Algorithm 18 provides the pseudo code for the global optimal algorithm at each agent. This algorithm is invoked with the procedure call GO-JOINTPOLICY$(root, \langle\rangle, no)$. Lines 8-21 represent the UTIL propagation, while Lines 1-4 and 22-24 represent the VALUE propagation phase of DPOP. Line 8 iterates through all the possible policies, where as lines 20-21 work towards calculating the best policy over this entire set of policies using the value of the policies calculated in Lines 9-19. Line 21 stores the values of best response policies obtained from the children. Lines 22-24 starts the termination of the algorithm after all the policies are exhausted at the root. Lines 1-4 propagate the termination message to lower levels in the tree, while recording the best policy, π_i^*.

By using cycle-cutset algorithms [Dechter, 2003], GOA can be applied to interaction graphs containing cycles. These algorithms are used to identify a cycle-cutset, i.e., a subset of agents, whose deletion makes the remaining interaction graph acyclic. After identifying the cutset, joint policies for the cutset agents are enumerated, and then for each of them, we find the best policies of remaining agents using GOA.

6.7 Experimental Results

In this section we provide two sets of experiments. The first set of experiments provide performance comparisons of the locally optimal algorithm, LID-JESP to globally optimal algorithm, GOA and other benchmark algorithms (JESP, LID-JESP no network). Second set of experiments provides comparisons of LID-JESP and its enhancements (SLID-JESP, LID-JESP+HLD, SLID-JESP+HLD). All the experiments were performed on the sensor domain explained in Section 2.1.2.

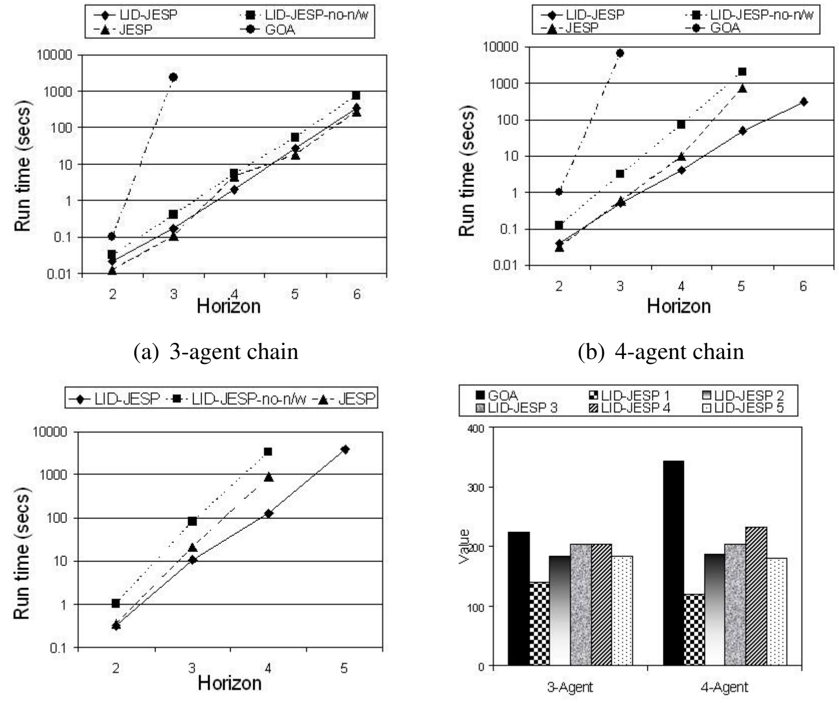
Algorithm 18 GO-JOINTPOLICY$(i, \pi_j, terminate)$

```
if $terminate$ = yes then
    $\pi_i^* \leftarrow bestResponse\{\pi_j\}$
    for all $k \in children_i$ do
        GO-JOINTPOLICY($k, \pi_i^*$,yes)
    end for
    return
end if
$\Pi_i \leftarrow$ enumerate all possible policies
$bestPolicyVal \leftarrow$ -$\infty$,$j \leftarrow$ parent($i$)
for all $\pi_i \in \Pi_i$ do
    $jointPolicyVal \leftarrow 0$, $childVal \leftarrow 0$
    if $i \neq$ root then
        for all $s_i, s_j, s_u$ do
            $jointPolicyVal \overset{+}{\leftarrow} b_i(s_i) \cdot b_{N_i}(s_{N_i}) \cdot b_u(s_u) \cdot$ EVALUATE$(i, s_i, s_u, s_j, \pi_i, \pi_j, \langle\rangle, \langle\rangle, 0, T)$
        end for
    end if
    if $bestChildValMap\{\pi_i\} \neq$ null then
        $jointPolicyVal \overset{+}{\leftarrow} bestChildValMap\{\pi_i\}$
    else
        for all $k \in children_i$ do
            $childVal \overset{+}{\leftarrow}$ GO-JOINTPOLICY($k, \pi_i$,no)
        end for
        $bestChildValMap\{\pi_i\} \leftarrow childVal$
        $jointPolicyVal \overset{+}{\leftarrow} childVal$
    end if
    if $jointPolicyVal > bestPolicyVal$ then
        $bestPolicyVal \leftarrow jointPolicyVal$, $\pi_i^* \leftarrow \pi_i$
    end if
end for
if $i = root$ then
    for all $k \in children_i$ do
        GO-JOINTPOLICY($k, \pi_i^*$,yes)
    end for
end if
if $i \neq root$ then $bestResponse\{\pi_j\} = \pi_i^*$
return $bestPolicyVal$
```

In the first set of experiments, we consider three different sensor network configurations of increasing complexity. In the following text, Loc1-1, Loc2-1 and Loc2-2 are the same regions as in Figure 2.2. The first configuration is a chain with 3 agents (sensors 1-3). Here target1 is either absent or in Loc1-1 and target2 is either absent or in Loc2-1 (4 unaffectable states). Each agent can perform either turnOff, scanEast or scanWest. Agents receive an observation, targetPresent or targetAbsent, based on the unaffectable state and its last action. The second configuration is a 4 agent chain (sensors 1-4). Here, target2 has an additional possible location, Loc2-2, giving rise to 6 unaffectable states. The number of individual actions and observations are unchanged. The third configuration is the 5 agent P-configuration (named for the P shape of the sensor net) and is identical to Figure 2.2. Here, target1 can have two additional locations, Loc1-2 and Loc1-3, giving rise to 12 unaffectable states. We add a new action called scanVert for each agent to scan North and South. For each of these scenarios, we ran the LID-JESP algorithm. Our first benchmark, JESP, uses a centralized policy generator to find a locally optimal joint policy and does not consider the network structure of the interaction, while our second benchmark (LID-JESP-no-nw) is LID-JESP with a fully connected *interaction graph*. For 3 and 4 agent chains, we also ran the GOA algorithm.

Figure 6.2 compares the performance of the various algorithms for 3 and 4 agent chains and 5 agent P-configuration. Graphs (a), (b), (c) show the run time in seconds on a logscale on Y-axis for increasing finite horizon T on X-axis. Run times for LID-JESP, JESP and LID-JESP-no-nw are averaged over 5 runs, each run with a different randomly chosen starting policy . For a particular run, all algorithms use the same starting policies. All three locally optimal algorithms show significant improvement over GOA in terms of run time with LID-JESP outperforming LID-JESP-no-nw and JESP by an order of magnitude (for high T) by exploiting *locality of interaction*. In graph (d), the values obtained using GOA for 3 and 4-Agent case ($T = 3$) are compared to the ones obtained using LID-JESP over 5 runs (each with a different starting policy) for $T = 3$. In this bar graph, the first bar represents value obtained using GOA, while other bars correspond to LID-JESP. This graph emphasizes the fact that with random restarts, LID-JESP converges to a higher local optima — such restarts are afforded given that GOA is orders of magnitude slower compared to LID-JESP.

Table 6.1 helps to better explain the reasons for the speed up of LID-JESP over JESP and LID-JESP-no-nw. LID-JESP allows more than one (non-neighboring) agent to change its policy within a cycle (W), LID-JESP-no-nw allows exactly one agent to change its policy in a cycle and in JESP, there are several cycles where no agent changes its policy. This allows LID-JESP to converge in fewer cycles (C) than LID-JESP-no-nw. Although LID-JESP takes fewer cycles than JESP to converge, it required more calls to GETVALUE (G). However, each such call is cheaper owing to the locality of interaction. LID-JESP will out-perform JESP even more on multi-processor machines owing to its distributedness.



(a) 3-agent chain

(b) 4-agent chain

(c) 5-agent P

(d)

Figure 6.2: Run times (a, b, c), and value (d).

In the second set of experiments, we performed comparison of LID-JESP with the enhancements – SLID-JESP, LID-JESP with HLD and SLID-JESP with HLD – in terms of value and runtime for some complex network structures (2x3 and cross) as well. We used four different topologies of sensors, shown in Figure 6.3, each with a different target movement scenario. With two targets moving in the environment, possible positions of targets are increased as the network grows and the number of unaffected states are increased accordingly. Figure 6.3(a) shows the example where there are 3 sensors arranged in a chain and the number of possible positions for each target is 1. In the cross topology, as in Figure 6.3(b), we considered 5 sensors with one sensor in the center surrounded by 4 sensors and 2 locations are possible for each target. In the example in Figure 6.3(c) with 5 sensors arranged in P shape, target1 and target2 can be at 2 and 3 locations respectively, thus leading to a total of 12 states. There are total 20 states for six sensors in example of Figure 6.3(d) with 4 and 3 locations for target1 and target2, respectively. As we assumed earlier, each target is independent of each other. Thus, total number of unaffected

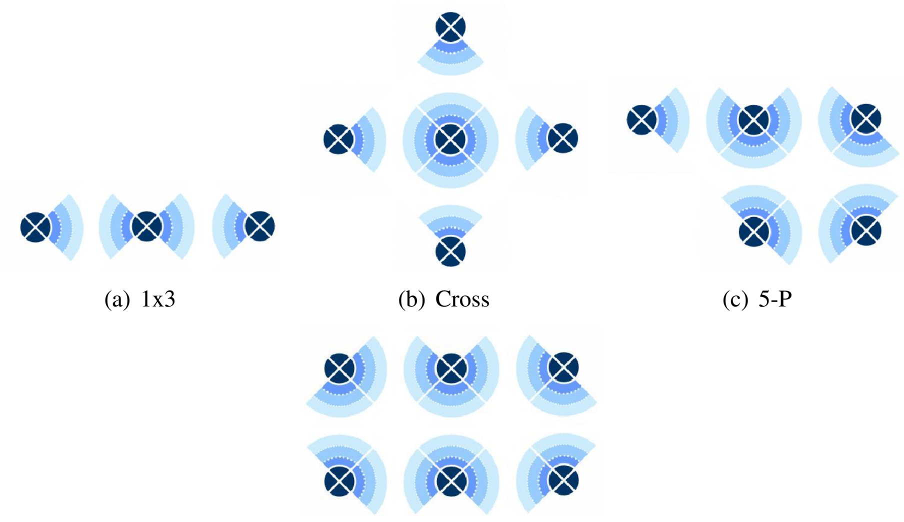
Config.	Algorithm	C	G	W
4-chain	LID-JESP	3.4	13.6	1.412
	LID-JESP-no-nw	4.8	19.2	1
	JESP	7.8	7.8	0.436
5-P	LID-JESP	4.2	21	1.19
	LID-JESP-no-nw	5.8	29	1
	JESP	10.6	10.6	0.472

Table 6.1: Reasons for speed up. C: no. of cycles, G: no. of GETVALUE calls, W: no. of winners per cycle, for T=2.

states are ($\prod_{targets}$(number of possible positions of each target $+ 1$)). Due to the exponentially increasing runtime, the size of the network and time horizon is limited but is still significantly larger than those which have previously been demonstrated in distributed POMDPs. All experiments are started at random initial policies and averaged over five runs for each algorithm. We chose 0.9 as the threshold probability (p) for SLID-JESP which empirically gave a good result for most cases.

Figure 6.4 shows performance improvement of SLID-JESP and HLD in terms of runtime. In Figure 6.4, X-axis shows the time horizon T, while Y-axis shows the runtime in milliseconds on a logarithmic scale. In all cases of Figure 6.4, the line of SLID-JESP is lower than that of LID-JESP with and without HLD where the difference of two grows as the network grows. As in Figure 6.4(c) and Figure 6.4(d) the difference in runtime between LID-JESP and SLID-JESP is bigger than that in smaller network examples. The result that SLID-JESP always takes less time than LID-JESP is because in SLID-JESP, more agents change their policy in one cycle, and hence SLID-JESP tends to converge to a local optimum quickly. As for HLD, all the graphs shows that the use of Hyper-link-based decomposition clearly improved LID-JESP and SLID-JESP in terms of runtime. The improvement is more visible when the number of neighbors increases where HLD takes advantage of decomposition. For example, in Figure 6.4(b), by using HLD the runtime reduced by more than an order of magnitude for $T = 4$. In cross topology, the computation for the agent in the center which has 4 neighbors is a main bottleneck and HLD significantly reduces the computation by decomposition.

Figure 6.5 shows the values of each algorithm for different topologies. In Figure 6.5, X-axis shows the time horizon T, while Y-axis shows the value of team reward. There are only two lines in each graph because the values of the algorithm with HLD and without HLD are always the same because HLD only exploits independence between neighbors and doesn't affect the value of the resulting joint policy. The reward of LID-JESP is larger than that of SLID-JESP in three out of the four topologies that we tried. This suggests SLID-JESP's greedy approach to changing the joint policy causes it to converge to lower local optima than LID-JESP in some cases. However, note that in Figure 6.5(a) SLID-JESP converges to a higher local optima than LID-JESP. This suggests that network topology greatly impacts the choice of whether to use LID-JESP or SLID-JESP. Furthermore, the results of SLID-JESP vary in value for different threshold probabilities. However, there is a consistent trend that the result is better when the threshold probability (p) is large. This trend means that in our domain, it is generally better to change policy if there is a visible gain.



(a) 1x3

(b) Cross

(c) 5-P

(d) 2x3

Figure 6.3: Different sensor net configurations.

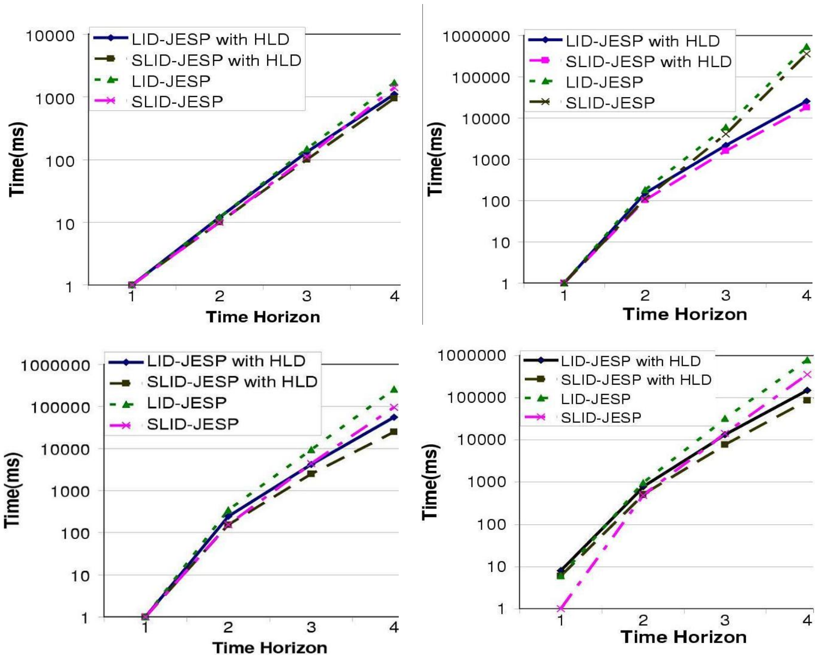


Figure 6.4: Runtime (ms) for (a) 1x3, (b) cross, (c) 5-P and (d) 2x3.

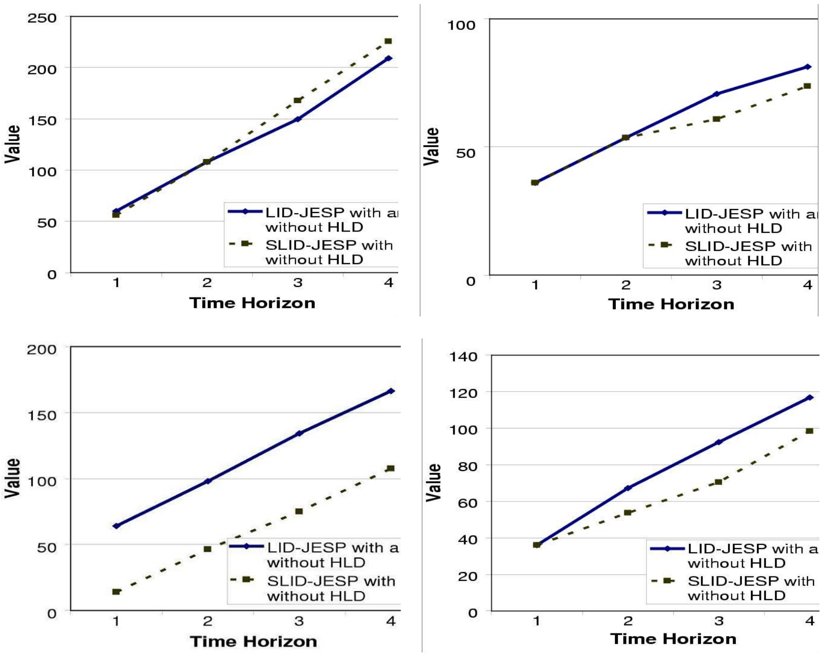


Figure 6.5: Value for (a) 1x3, (b) cross, (c) 5-P and (d) 2x3.

Chapter 7

Direct value approximation and exploiting interaction structure (Distributed POMDPs)

Whereas previous chapter illustrated exploitation of structure for efficient computation of approximate solutions, this chapter exploits structure for exact algorithms. In addition, I also present a direct value approximation enhancement for Distributed POMDPs. Thus, the technique introduced in this chapter not only provides guarantees on solution quality, but also exploits the network structure to compute solutions efficiently for a network of agents.

In particular, this chapter introduces the exact algorithm SPIDER (Search for Policies In Distributed EnviRonments), before presenting the approximation technique algorithm. SPIDER is a branch and bound heuristic search technique that uses a MDP-based heuristic function to search for an optimal joint policy. This MDP-based heuristic approximates the distributed POMDP as a single agent centralized MDP and computes the value corresponding to the optimal policy of this MDP. In a similar vein to the structure exploitation presented in Section 6.6, SPIDER also exploits network structure of agents by organizing agents into a DFS tree (Depth First Search) or pseudo tree [Petcu and Faltings, 2005] and exploiting independence in the different branches of the tree (while constructing joint policies). Furthermore, the MDP-based heuristic function is also computed efficiently by utilizing the interaction structure.

I then provide three enhancements to improve the efficiency of the basic SPIDER algorithm while providing guarantees on the quality of the solution. The first enhancement is an exact one, based on the idea of initially performing branch and bound search on abstract policies (representing a group of complete policies) and then extending to the complete policies. Second enhancement bounds the search approximately given a parameter that provides the tolerable expected value difference from the optimal solution. The third enhancement is again based on bounding the search approximately, however with a tolerance parameter that is provided as a percentage of optimal.

We experimented with the sensor network domain presented in Section 2.1.2, while the model used to represent the domain is the Network Distributed POMDP model (presented in Section 6.1). In our experimental results, we show that SPIDER dominates an existing global optimal approach, GOA presented in Section 6.6. GOA is the only known global optimal algorithm that works with more than two agents. Furthermore, we demonstrate that the idea of abstraction improves the performance of SPIDER significantly while providing optimal solutions and also that by utilizing the approximation enhancements, SPIDER provides significant improvements in run-time performance while not losing significantly on quality.

7.1 Search for Policies In Distributed EnviRonments (SPIDER)

As mentioned in Section 6.1, an ND-POMDP can be treated as a DCOP, where the goal is to compute a joint policy that maximizes the overall joint reward. The bruteforce technique for computing an optimal policy would be to examine the expected values for all possible joint policies. The key idea in SPIDER is to avoid computation of expected values for the entire space of joint policies, by utilizing upperbounds on the expected values of policies and the interaction structure of the agents.

Akin to some of the algorithms for DCOP [Modi et al., 2003a; Petcu and Faltings, 2005], SPIDER has a pre-processing step that constructs a DFS tree corresponding to the given interaction structure. We employ the Maximum Constrained Node (MCN) heuristic used in ADOPT [Modi et al., 2003a], however other heuristics (such as MLSP heuristic from [Maheswaran et al., 2004]) can also be employed. MCN heuristic tries to place agents with more number of constraints at the top of the tree. This tree governs how the search for the optimal joint policy proceeds in SPIDER. The algorithms presented in this paper are easily extendable to hyper-trees, however for expository purposes, we assume a binary tree.

SPIDER is an algorithm for centralized planning and distributed execution in distributed POMDPs. Though the explanation is presented from the perspective of individual agents, the algorithm is centralized. In this paper, we employ the following notation to denote policies and expected values of joint policies:

$Ancestors(i)$ ⇒ agents from i to the $root$ (not including i).

$Tree(i)$ ⇒ agents in the sub-tree (not including i) for which i is the root.

π^{root+} ⇒ joint policy of all agents.

π^{i+} ⇒ joint policy of all agents in the sub-tree for which i is the root.

π^{i-} ⇒ joint policy of agents that are ancestors to agents in the sub-tree for which i is the root.

π_i ⇒ policy of the ith agent.

$\hat{v}[\pi_i, \pi^{i-}] \Rightarrow$ upper bound on the expected value for π^{i+} given π_i and policies of ancestor agents i.e. π^{i-}.

$\hat{v}_j[\pi_i, \pi^{i-}] \Rightarrow$ upper bound on the expected value for π^{i+} from the jth child.

$v[\pi_i, \pi^{i-}] \Rightarrow$ expected value for π_i given policies of ancestor agents i.e. π^{i-}.

$v[\pi^{i+}, \pi^{i-}] \Rightarrow$ expected value for π^{i+} given policies of ancestor agents π^{i-}.

$v_j[\pi^{i+}, \pi^{i-}] \Rightarrow$ expected value for π^{i+} from the jth child.

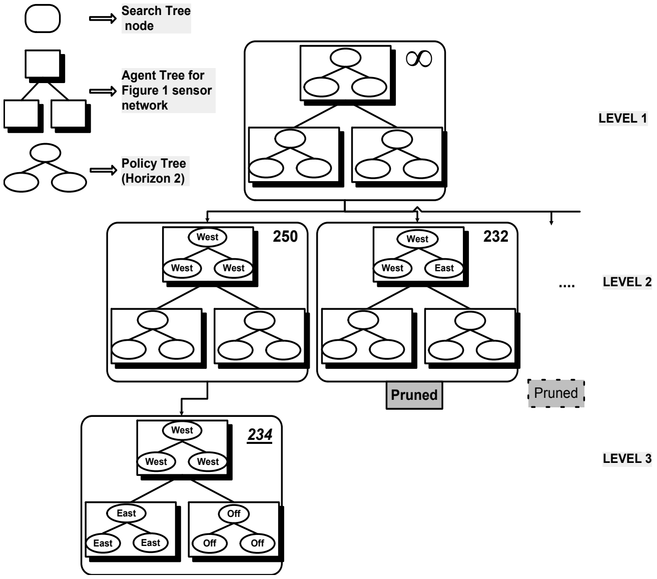


Figure 7.1: Execution of SPIDER, an example

7.1.1 Outline of SPIDER

SPIDER is based on the idea of branch and bound search, where the nodes in the search tree represent the joint policies, π^{root+}. Figure 7.1 shows an example search tree for the SPIDER algorithm, using an example of the three agent chain. We create a tree from this chain, with the middle agent as the root of the tree. Note that in our example figure each agent is assigned a policy with T=2. Each rounded rectange (search tree node) indicates a partial/complete joint policy and a rectangle indicates an agent. Heuristic or actual expected value for a joint policy is indicated in the top right corner of the rounded rectangle. If the number is italicized and underlined, it implies that the actual expected value of the joint policy is provided. SPIDER begins with no policy assigned to any of the agents (shown in the level 1 of the search tree). Level 2 of the search tree indicates that the joint policies are sorted based on upper bounds computed for root agent's policies. Level 3 contains a node with a complete joint policy (a policy assigned to each of the agents). The expected value for this joint policy is used to prune out the nodes in level 2 (the ones with upper bounds < 234)

When creating policies for each non-leaf agent i, SPIDER potentially performs two steps:

1. Obtaining upper bounds and sorting: In this step, agent i computes upper bounds on the expected values, $\hat{v}[\pi_i, \pi^{i-}]$ of the joint policies π^{i+} corresponding to each of its policy π_i and fixed ancestor policies. A MDP based heuristic is used to compute these upper bounds on the expected values. Detailed description about this MDP heuristic and other possible heuristics is provided in Section 7.1.2. All policies of agent i, Π_i are then sorted based on these upper bounds (also referred to as heuristic values henceforth) in descending order. Exploration of these policies (in step 2 below) are performed in this descending order. As indicated in the level 2 of the search tree of Figure 7.1, all the joint policies are sorted based on the heuristic values, indicated in the top right corner of each joint policy. The intuition behind sorting and then exploring policies in descending order of upper bounds, is that the policies with higher upper bounds could yield joint policies with higher expected values.

2. Exploration and Pruning: Exploration here implies computing the best response joint policy $\pi^{i+,*}$ corresponding to fixed ancestor policies of agent i, π^{i-}. This is performed by iterating through all policies of agent i i.e. Π_i and then for each policy, computing and summing two quantities: (i) compute the best response for each of i's children (obtained by performing steps 1 and 2 at each of the child nodes); (ii) compute the expected value obtained by i for fixed policies of ancestors. Thus, exploration of a policy π_i yields actual expected value of a joint policy, π^{i+} represented as $v[\pi^{i+}, \pi^{i-}]$. The policy with the highest expected value is the best response policy.

   Pruning refers to the process of avoiding exploring policies (or computing expected values) at agent i by using the maximum expected value, $v^{max}[\pi^{i+}, \pi^{i-}]$ encountered until this juncture. Henceforth, this $v^{max}[\pi^{i+}, \pi^{i-}]$ will be referred to as *threshold*. A policy, π_i need not be explored if the upper bound for that policy, $\hat{v}[\pi_i, \pi^{i-}]$ is less than the *threshold*. This is because the best joint policy that can be obtained from that policy will have an expected value that is less than the expected value of the current best joint policy.

On the other hand, when considering a leaf agent, SPIDER computes the best response policy (and consequently its expected value) corresponding to fixed policies of its ancestors, π^{i-}. This is accomplished by computing expected values for each of the policies (corresponding to fixed policies of ancestors) and selecting the policy with the highest expected value. Going back to

Algorithm 19 SPIDER$(i, \pi^{i-}, threshold)$

```
1:  $\pi^{i+,*} \leftarrow null$
2:  $\Pi_i \leftarrow$ GET-ALL-POLICIES $(horizon, A_i, \Omega_i)$
3:  **if** IS-LEAF(i) **then**
4:      **for all** $\pi_i \in \Pi_i$ **do**
5:          $v[\pi_i, \pi^{i-}] \leftarrow$ JOINT-REWARD $(\pi_i, \pi^{i-})$
6:          **if** $v[\pi_i, \pi^{i-}] > threshold$ **then**
7:              $\pi^{i+,*} \leftarrow \pi_i$
8:              $threshold \leftarrow v[\pi_i, \pi^{i-}]$
9:          **end if**
10:     **end for**
11: **else**
12:     $children \leftarrow$ CHILDREN $(i)$
13:     $\hat{\Pi}_i \leftarrow$ UPPER-BOUND-SORT$(i, \Pi_i, \pi^{i-})$
14:     **for all** $\pi_i \in \hat{\Pi}_i$ **do**
15:         $\tilde{\pi}^{i+} \leftarrow \pi_i$
16:         **if** $\hat{v}[\pi_i, \pi^{i-}] < threshold$ **then**
17:             Go to line 12
18:         **end if**
19:         **for all** $j \in children$ **do**
20:             $jThres \leftarrow threshold - v[\pi_i, \pi^{i-}] - \Sigma_{k \in children, k \neq j} \hat{v}_k[\pi_i, \pi^{i-}]$
21:             $\pi^{j+,*} \leftarrow$ SPIDER$(j, \pi_i \parallel \pi^{i-}, jThres)$
22:             $\tilde{\pi}^{i+} \leftarrow \tilde{\pi}^{i+} \parallel \pi^{j+,*}$
23:             $\hat{v}_j[\pi_i, \pi^{i-}] \leftarrow v[\pi^{j+,*}, \pi_i \parallel \pi^{i-}]$
24:         **end for**
25:         **if** $v[\tilde{\pi}^{i+}, \pi^{i-}] > threshold$ **then**
26:             $threshold \leftarrow v[\tilde{\pi}^{i+}, \pi^{i-}]$
27:             $\pi^{i+,*} \leftarrow \tilde{\pi}^{i+}$
28:         **end if**
29:     **end for**
30: **end if**
31: **return** $\pi^{i+,*}$
```

Algorithm 20 UPPER-BOUND-SORT(i, Π_i, π^{i-})

```
$children \leftarrow$ CHILDREN $(i)$
$\hat{\Pi}_i \leftarrow null$ /* Stores the sorted list */
**for all** $\pi_i \in \Pi_i$ **do**
    $\hat{v}[\pi_i, \pi^{i-}] \leftarrow$ JOINT-REWARD $(\pi_i, \pi^{i-})$
    **for all** $j \in children$ **do**
        $\hat{v}_j[\pi_i, \pi^{i-}] \leftarrow$ UPPER-BOUND$(j, \pi_i \parallel \pi^{i-})$
        $\hat{v}[\pi_i, \pi^{i-}] \overset{+}{\leftarrow} \hat{v}_j[\pi_i, \pi^{i-}]$
    **end for**
    $\hat{\Pi}_i \leftarrow$ INSERT-INTO-SORTED $(\pi_i, \hat{\Pi}_i)$
**end for**
**return** $\hat{\Pi}_i$
```

Figure 7.1, SPIDER assigns best response policies to leaf agents at level 3. The policy for the left leaf agent is to perform action East at each time step in the policy, while the policy for the right leaf agent is to perform ”Off” at each time step. This best response policies from the leaf agents yield an actual expected value of 234 for the complete joint policy.

Algorithm 19 provides the pseudo code for SPIDER. This algorithm outputs the best joint policy, $\pi^{i+,*}$ (with an expected value greater than *threshold*) for the agents in the sub-tree with agent i as the root. Lines 3-8 compute the best response policy of a leaf agent i by iterating through all the policies (line 4) and finding the policy with the highest expected value (lines 5-8). Lines 9-23 computes the best response joint policy for agents in the sub-tree with i as the root. Sorting of policies (in descending order) based on heuristic policies is done on line 11.

Exploration of a policy i.e. computing best response joint policy corresponding to fixed ancestor policies is done in lines 12-23. This includes computation of best joint policies for each of the child sub-trees (lines 16-23). This computation in turn involves distributing the *threshold* (line 17), recursively calling the SPIDER algorithm (line 18) for each of the children and maintaining the best expected value, joint policy (lines 21-23). **Pruning** of policies is performed in lines 14-15 by comparing the upper bound on the expected value against the *threshold*.

Algorithm 20 provides the algorithm for sorting policies based on the upper bounds on the expected values of joint policies. Expected value for an agent i consists of two parts: value obtained from ancestors and value obtained from its children. Line 4 computes the value obtained from (fixed policies of) ancestors of the agent (by using the JOINT-REWARD function), while lines 5-7 compute the heuristic value (upper-bounds) from the children. Thus the sum of these two parts yields an upper bound on the expected value for agent i, and line 8 of the algorithm is used for sorting the policies based on these upper bounds.

7.1.2 MDP based heuristic function

The job of the heuristic function is to quickly provide an upper bound on the expected value obtainable from the sub-tree for which i is the root. The sub-tree of agents is a distributed POMDP in itself and the idea here is to construct a centralized MDP corresponding to the (sub-tree) distributed POMDP and obtain the expected value of the optimal policy for this centralized MDP. To reiterate this in terms of the agents in DFS tree interaction structure, we assume full observability for the agents in the $Tree(i)$ and for fixed policies of the agents in the set $\{Ancestors(i) \cup i\}$, we compute the joint value $\hat{v}[\pi^{i+}, \pi^{i-}]$.

We use the following notation for presenting the equations for computing upper bounds/heuristic values (for agents i and k):

Let E^{i-} denote the set of links between agents in $Ancestors(i)$ and $Tree(i) \cup i$ and E^{i+} denote the set of links between agents in $Tree(i) \cup i$. Also, if $l \in E^{i-}$, then l_1 denotes the agent in $Ancestors(i)$ and l_2 denotes the agent in $Tree(i)$.

$$\begin{aligned}
o_k^t &\triangleq O_k(s_k^{t+1}, s_u^{t+1}, \pi_k(\vec{\omega}_k^t), \omega_k^{t+1}) && (7.1)\\
p_k^t &\triangleq P_k(s_k^t, s_u^t, \pi_k(\vec{\omega}_k^t), s_k^{t+1}) \cdot o_k^t\\
\hat{p}_k^t &\triangleq p_k^t, if\ \ k \in \{Ancestors(i) \cup i\}\\
&\triangleq P_k(s_k^t, s_u^t, \pi_k(\vec{\omega}_k^t), s_k^{t+1}), if\ \ k \in Tree(i) && (7.2)\\
p_u^t &\triangleq P(s_u^t, s_u^{t+1})\\
s_l^t &= \langle s_{l_1}^t, s_{l_2}^t, s_u^t \rangle\\
\omega_l^t &= \langle \omega_{l_1}^t, \omega_{l_2}^t \rangle\\
r_l^t &\triangleq R_l(s_l^t, \pi_{l_1}(\vec{\omega}_{l_1}^t), \pi_{l_2}(\vec{\omega}_{l_2}^t))\\
\text{IF }\ l \in E^{i-}, \hat{r}_l^t &\triangleq \max_{\{a_{l_2}\}} R_l(s_l^t, \pi_{l_1}(\vec{\omega}_{l_1}^t), a_{l_2})\\
\text{IF }\ l \in E^{i+}, \hat{r}_l^t &\triangleq \max_{\{a_{l_1}, a_{l_2}\}} R_l(s_l^t, a_{l_1}, a_{l_2})\\
v_l^t &\triangleq V_{\pi_l}^t(s_l^t, s_u^t, \vec{\omega}_{l_1}^t, \vec{\omega}_{l_2}^t)
\end{aligned}$$

The value function for an agent i executing the joint policy π^{i+} at time $\eta - 1$ is provided by the equation:

$$\begin{aligned}
V_{\pi^{i+}}^{\eta-1}(s^{\eta-1}, \vec{\omega}^{\eta-1}) &= \sum_{l \in E^{i-}} v_l^{\eta-1} + \sum_{l \in E^{i+}} v_l^{\eta-1}\\
where\ \ v_l^{\eta-1} &= r_l^{\eta-1} + \sum_{\omega_l^\eta, s^\eta} p_{l_1}^{\eta-1} p_{l_2}^{\eta-1} p_u^{\eta-1} v_l^\eta && (7.3)
\end{aligned}$$

Algorithm 21 UPPER-BOUND (j, π^{j-})

```
$val \leftarrow 0$
for all $s_l^0$ do
    $val \overset{+}{\leftarrow} startingBelief[s_l^0] \cdot$ UPPER-BOUND-TIME $(s_l^0, j, \{\}, \langle\rangle, \langle\rangle)$
end for
return $val$
```

Upper bound on the expected value for a link is computed by modifying the equation 7.3 to reflect the full observability assumption. This involves removing the observational probability

Algorithm 22 UPPER-BOUND-TIME $(s_l^t, j, \pi_{l_1}, \vec{\omega}_{l_1}^t)$

```
$val \leftarrow$ GET-REWARD$(s_l^t, a_{l_1}, a_{l_2})$
**if** $t < \pi_i.horizon - 1$ **then**
    **for all** $s_l^{t+1}, \omega_{l_1}^{t+1}$ **do**
        $futVal \leftarrow p_u^t \hat{p}_{l_1}^t \hat{p}_{l_2}^t$
        $futVal \overset{*}{\leftarrow}$ UPPER-BOUND-TIME$(s_l^{t+1}, j, \pi_{l_1}, \vec{\omega}_{l_1}^t \parallel \omega_{l_1}^{t+1})$
    **end for**
    $val \overset{+}{\leftarrow} futVal$
**end if**
**return** $val$
```

term for agents in $Tree(i)$ and maximizing the future value $\hat{v}_l^\eta$ over the actions of those agents (in $Tree(i)$). Thus, the equation for the computation of the upper bound will be as follows:

$$\text{IF } l \in E^{i-}, \hat{v}_l^{\eta-1} = \hat{r}_l^{\eta-1} + \max_{a_{l_2}} \sum_{\omega_{l_1}^\eta, s_l^\eta} \hat{p}_{l_1}^{\eta-1} \hat{p}_{l_2}^{\eta-1} p_u^{\eta-1} \hat{v}_l^\eta$$

$$\text{IF } l \in E^{i+}, \hat{v}_l^{\eta-1} = \hat{r}_l^{\eta-1} + \max_{a_{l_1}, a_{l_2}} \sum_{s_l^\eta} \hat{p}_{l_1}^{\eta-1} \hat{p}_{l_2}^{\eta-1} p_u^{\eta-1} \hat{v}_l^\eta$$

Algorithm 21 and Algorithm 22 provide the algorithm for computing upper bound for child j of agent i using the equations above. Algorithm 21 maximizes over all possible combinations of actions for agents in $Tree(j) \cup j$. The value for a combination terates over all links associated with an agent While Algorithm 22 computes the upper bound on a link, l

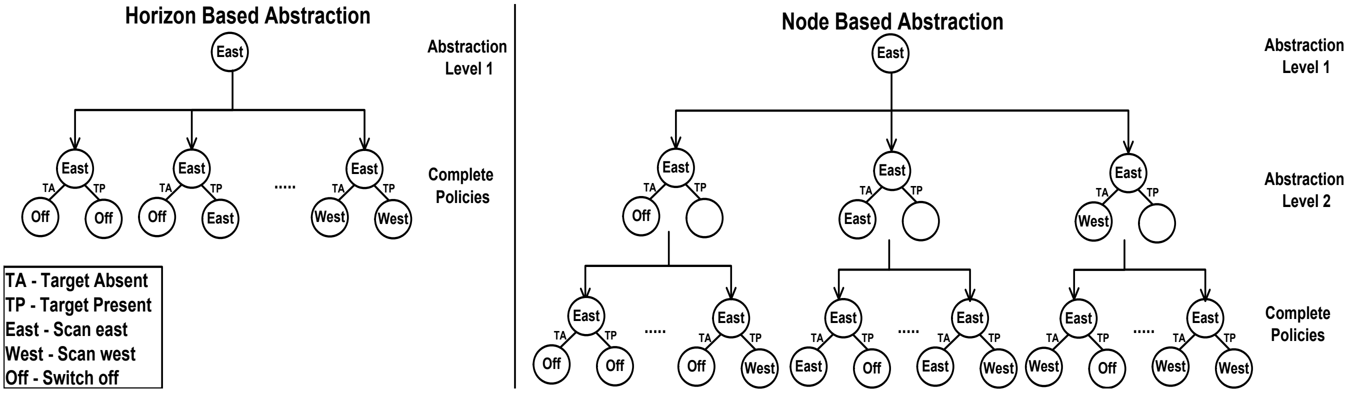


Figure 7.2: Example of abstraction for (a) HBA (Horizon Based Abstraction) and (b) NBA (Node Based Abstraction)

7.1.3 Abstraction

In SPIDER, the exploration/pruning phase can only begin after the heuristic (or upper bound) computation and sorting for the policies has finished. We provide an approach of interleaving

Algorithm 23 SPIDER-ABS$(i, \pi^{i-}, threshold)$

```
 1: $\pi^{i+,*} \leftarrow null$
 2: $\Pi_i \leftarrow$ GET-POLICIES $(<>, 1)$
 3: **if** IS-LEAF(i) **then**
 4:   **for all** $\pi_i \in \Pi_i$ **do**
 5:     $absHeuristic \leftarrow$ GET-ABS-HEURISTIC $(\pi_i, \pi^{i-})$
 6:     $absHeuristic \overset{*}{\leftarrow} (timeHorizon - \pi_i.horizon)$
 7:     **if** $\pi_i.horizon = timeHorizon$ and $\pi_i.absNodes = 0$ **then**
 8:       $v[\pi_i, \pi^{i-}] \leftarrow$ JOINT-REWARD $(\pi_i, \pi^{i-})$
 9:       **if** $v[\pi_i, \pi^{i-}] > threshold$ **then**
10:         $\pi^{i+,*} \leftarrow \pi_i; threshold \leftarrow v[\pi_i, \pi^{i-}]$
11:       **end if**
12:     **else if** $v[\pi_i, \pi^{i-}] + absHeuristic > threshold$ **then**
13:       $absNodes \leftarrow \pi_i.absNodes + 1$
14:       $\hat{\Pi}_i \leftarrow$ GET-POLICIES $(\pi_i, \pi_i.horizon + 1, absNodes)$
15:       /* Insert policies in the beginning of $\Pi_i$ in sorted order*/
16:       $\Pi_i \overset{+}{\leftarrow}$ INSERT-SORTED-POLICIES $(\hat{\Pi}_i)$
17:     **end if**
18:     REMOVE$(\pi_i)$
19:   **end for**
20: **else**
21:   $children \leftarrow$ CHILDREN $(i)$
22:   $\Pi_i \leftarrow$ UPPER-BOUND-SORT$(i, \Pi_i, \pi^{i-})$
23:   **for all** $\pi_i \in \Pi_i$ **do**
24:     $\tilde{\pi}^{i+} \leftarrow \pi_i$
25:     $absHeuristic \leftarrow$ GET-ABS-HEURISTIC $(\pi_i, \pi^{i-})$
26:     $absHeuristic \overset{*}{\leftarrow} (timeHorizon - \pi_i.horizon)$
27:     **if** $\pi_i.horizon = timeHorizon$ and $\pi_i.absNodes = 0$ **then**
28:       **if** $\hat{v}[\pi_i, \pi^{i-}] < threshold$ and $\pi_i.absNodes = 0$ **then**
29:         Go to line 19
30:       **end if**
31:       **for all** $j \in children$ **do**
32:         $jThres \leftarrow threshold - v[\pi_i, \pi^{i-}] - \Sigma_{k \in children, k \neq j} \hat{v}_k[\pi_i, \pi^{i-}]$
33:         $\pi^{j+,*} \leftarrow$ SPIDER$(j, \pi_i \parallel \pi^{i-}, jThres)$
34:         $\tilde{\pi}^{i+} \leftarrow \tilde{\pi}^{i+} \parallel \pi^{j+,*}; \hat{v}_j[\pi_i, \pi^{i-}] \leftarrow v[\pi^{j+,*}, \pi_i \parallel \pi^{i-}]$
35:       **end for**
36:       **if** $v[\tilde{\pi}^{i+}, \pi^{i-}] > threshold$ **then**
37:         $threshold \leftarrow v[\tilde{\pi}^{i+}, \pi^{i-}]; \pi^{i+,*} \leftarrow \tilde{\pi}^{i+}$
38:       **end if**
39:     **else if** $\hat{v}[\pi^{i+}, \pi^{i-}] + absHeuristic > threshold$ **then**
40:       $absNodes \leftarrow \pi_i.absNodes + 1$
41:       $\hat{\Pi}_i \leftarrow$ GET-POLICIES $(\pi_i, \pi_i.horizon, absNodes)$
42:       /* Insert policies in the beginning of $\Pi_i$ in sorted order*/
43:       $\Pi_i \overset{+}{\leftarrow}$ INSERT-SORTED-POLICIES $(\hat{\Pi}_i)$
44:     **end if**
45:   **end for**
46:   REMOVE$(\pi_i)$
47: **end if**
48: **return** $\pi^{i+,*}$
```

exploration/pruning phase with the heuristic computation and sorting phase. This thus possibly circumvents the exploration of a group of policies based on heuristic computation for one abstract policy. The type of abstraction used dictates the amount of interleaving of exploration/pruning phase with heuristic computation phase. The important steps in this technique are defining the abstract policy and how heuristic values are computated for the abstract policies. In this paper, we propose two types of abstraction:

1. **Horizon Based Abstraction (HBA)**: In this type of abstraction, the abstract policy is defined as a shorter horizon policy. It represents a group of longer horizon policies that have the same actions as the abstract policy for times less than or equal to the horizon of the abstract policy. This is illustrated in Figure 7.2(a).

   For HBA, there are two parts to heuristic computation:

   (a) Computing the upper bound for the horizon of the abstract policy. This is same as the heuristic computation defined by the GET-HEURISTIC() algorithm for SPIDER, however with a shorter time horizon (horizon of the abstract policy).

   (b) Computing the maximum possible reward that can be accumulated in one time step and multiplying it by the number of time steps to time horizon. This maximum possible reward in turn is obtained by iterating through all the actions of all the agents involved (agents in the sub-tree with i as the root) and computing the maximum joint reward for any joint action.

   The sum of (a) and (b) above is the heuristic value for a HBA abstract policy.

2. **Node Based Abstraction (NBA)**: Abstraction of this type is performed by not associating actions to certain nodes of the policy tree, i.e. incomplete policies. Unlike abstraction (a) above, this implies multiple levels of abstraction. This is illustrated in Figure 7.2(b), where there is a T=1 policy that is an abstract policy for T=2 policies that do not contain an action for the case where TP is observed. These incomplete T=2 policies are further abstractions for T=2 complete policies. Increased levels of abstraction leads to faster computation of a complete joint policy, π^{root+} and also to shorter heuristic computation and exploration/pruning phases. For NBA, the heuristic computation is similar to that of a normal policy except in cases where there is no action associated with certain policy nodes. In cases where such nodes are encountered, the immediate reward is taken as R_{max} (maximum reward possible for any action).

We combine both the abstraction techniques mentioned above into one technique, SPIDER-ABS. Algorithm 23 provides the algorithm for this abstraction technique. For computing optimal joint policy with SPIDER-ABS, a non-leaf agent i initially examines all T=1 policies and sorts them based on abstract policy heuristic computations. This is performed on lines 2, 19 of Algorithm 23. These T=1 policies are then *explored* in descending order of heuristic values and ones that have heuristic values less than the *threshold* are pruned (lines 25-26). *Exploration* in SPIDER-ABS has the same definition as in SPIDER if the policy being *explored* has a horizon of policy computation which is equal to the actual time horizon and if all the nodes of the policy have an action associated with them (lines 27-30). However, if those conditions are not met, then it is substituted by a group of policies that it represents (referred to as *extension* henceforth) (lines 33-35). Before substituting the abstract policy, this group of policies are again sorted based on the heuristic values (line 37). At this juncture, if all the substituted policies have horizon of policy computation equal to the time horizon and all the nodes of these policies have actions associated with them, then the *exploration/pruning* phase akin to the one in SPIDER ensues (line 24). In case of partial policies, further *extension* of policies occurs. Similar type of abstraction based computation of best response is adopted at leaf agents in SPIDER-ABS (lines 3-16).

7.1.4 Value ApproXimation (VAX)

In this section, we present an approximate enhancement to SPIDER called VAX. The input to this technique is an approximation parameter ϵ, which determines the difference between the optimal solution and the approximate solution. This approximation parameter is used at each agent for pruning out joint policies. The pruning mechanism in SPIDER and SPIDER-Abs dictates that a joint policy be pruned only if the threshold is exactly greater than the heuristic value. However, the idea in this technique is to prune out joint policies even if *threshold* plus the approximation parameter, ϵ is greater than the heuristic value.

In the example of Figure 7.1, if the heuristic value for the second joint policy (or second search tree node) in level 2 were 238 instead of 232, then that policy could not be be pruned using SPIDER or SPIDER-Abs. However, in VAX with an approximation parameter of 5, the joint policy in consideration would also be pruned. This is because the *threshold* (234) at that juncture plus the approximation parameter (5), i.e. 239 would have been greater than the heuristic value for that joint policy (238). It can be noted from the example (just discussed) that this kind of *pruning* can lead to fewer *explorations* and hence lead to an improvement in the overall run-time performance. However, this can entail a sacrifice in the quality of the solution because this technique can prune out a candidate optimal solution. A bound on the error introduced by this approximate algorithm as a function of ϵ, is provided by Proposition 13.

7.1.5 Percentage ApproXimation (PAX)

In this section, we present the second approximation enhancement over SPIDER called PAX. Input to this technique is a parameter, δ that represents the percentage of the optimal solution quality that is tolerable. Output of this technique is a policy with an expected value that is at least $\frac{\delta}{100}$ of the optimal solution quality. As with VAX, this parameter is also used at each agent in the pruning phase. A policy is pruned if $\frac{\delta}{100}$ of its heuristic value is not greater than the *threshold*. Again in Figure 7.1, if the heuristic value for the second search tree node in level 2 were 238 instead of 232, then PAX with an input parameter of 98% would be able to prune that search tree node (since $\frac{98}{100} * 238 < 234$). Like in VAX, this leads to fewer explorations and hence an improvement in run-time performance, while potentially leading to a loss in quality of the solution. As shown in Proposition 14, this loss is again bounded and the bound is $\delta\%$ of the optimal solution quality.

7.1.6 Theoretical Results

Proposition 11 *Heuristic provided using the centralized MDP heuristic is admissible.*

Proof. For the value provided by the heuristic to be admissible, it should be an over estimate of the expected value for a joint policy. Thus, we need to show that:

For $l \in E^{i+} \cup E^{i-}$: $\hat{v}_l^t \geq v_l^t$.

We use mathematical induction on t to prove this.

Base case: $t = T - 1$. Irrespective of whether $l \in E^{i-}$ or $l \in E^{i+}$, $\hat{r}_l^t$ is computed by maximizing over all actions of the agents in the sub-tree for which i is the root, while r_l^t is computed for fixed policies of the same agents. Hence, $\hat{r}_l^t \geq r_l^t$ and also $\hat{v}_l^t \geq v_l^t$.

Assumption: Proposition holds for $t = \eta$, where $1 \leq \eta < T - 1$. Thus, $\hat{v}_l^\eta \geq v_l^\eta$, for $l \in E^{i-}$ or $l \in E^{i+}$.

We now have to prove that the proposition holds for $t = \eta - 1$ i.e. $\hat{v}_l^{\eta-1} \geq v_l^{\eta-1}$.

We initially prove that the above holds for $l \in E^{i-}$ and similar reasoning can be adopted to prove for $l \in E^{i+}$. The heuristic value function for $l \in E^{i-}$ is provided by the following equation:

$$\hat{v}_l^{\eta-1} = \hat{r}_l^{\eta-1} + \max_{a_{l_2}} \sum_{\omega_{l_1}^\eta, s_l^\eta} \hat{p}_{l_1}^{\eta-1} \hat{p}_{l_2}^{\eta-1} p_u^{\eta-1} \hat{v}_l^\eta$$

Rewriting the RHS and using Eqn 7.2

$$
\begin{aligned}
&= \hat{r}_l^{\eta-1} + \max_{a_{l_2}} \sum_{\omega_{l_1}^{\eta}, s_l^{\eta}} p_u^{\eta-1} p_{l_1}^{\eta-1} \hat{p}_{l_2}^{\eta-1} \hat{v}_l^{\eta} \\
&= \hat{r}_l^{\eta-1} + \sum_{\omega_{l_1}^{\eta}, s_l^{\eta}} p_u^{\eta-1} p_{l_1}^{\eta-1} \max_{a_{l_2}} \hat{p}_{l_2}^{\eta-1} \hat{v}_l^{\eta}
\end{aligned}
$$

Since $\max_{a_{l_2}} \hat{p}_{l_2}^{\eta-1} \hat{v}_l^{\eta} \geq \sum_{\omega_{l_2}} o_{l_2}^{\eta-1} \hat{p}_{l_2}^{\eta-1} \hat{v}_l^{\eta}$ and $p_{l_2}^{\eta-1} = o_{l_2}^{\eta-1} \hat{p}_{l_2}^{\eta-1}$

$$
\geq \hat{r}_l^{\eta-1} + \sum_{\omega_{l_1}^{\eta}, s_l^{\eta}} p_u^{\eta-1} p_{l_1}^{\eta-1} \sum_{\omega_{l_2}} p_{l_2}^{\eta-1} \hat{v}_l^{\eta}
$$

Since $\hat{v}_l^{\eta} \geq v_l^{\eta}$ (from the assumption)

$$
\begin{aligned}
&\geq \hat{r}_l^{\eta-1} + \sum_{\omega_{l_1}^{\eta}, s_l^{\eta}} p_u^{\eta-1} p_{l_1}^{\eta-1} \sum_{\omega_{l_2}} p_{l_2}^{\eta-1} v_l^{\eta} \\
&\geq \hat{r}_l^{\eta-1} + \sum_{(\omega_{l_1}^{\eta}, s_l^{\eta})} \sum_{\omega_{l_2}} p_u^{\eta-1} p_{l_1}^{\eta-1} p_{l_2}^{\eta-1} v_l^{\eta} \\
&\geq r_l^{\eta-1} + \sum_{(\omega_l^{\eta}, s_l^{\eta})} p_u^{\eta-1} p_{l_1}^{\eta-1} p_{l_2}^{\eta-1} v_l^{\eta} \\
&\geq v_l^{\eta-1}
\end{aligned}
$$

Thus proved. ■

Proposition 12 *SPIDER provides an optimal solution.*

Proof. SPIDER examines all possible joint policies given the interaction structure of the agents. The only exception being when a joint policy is *pruned* based on the heuristic value. Thus, as long as a candidate optimal policy is not pruned, SPIDER will return an optimal policy. As proved in Proposition 11, the expected value for a joint policy is always an upper bound. Hence when a joint policy is pruned, it cannot be an optimal solution.

Proposition 13 *Error bound on the solution quality for VAX (implemented over SPIDER-ABS) with an approximation parameter of ϵ is given by $\rho\epsilon$, where ρ indicates the number of leaf nodes in the DFS agent tree.*

Proof. We prove this proposition using mathematical induction on the $depth$ of the DFS tree.

Base case: depth $=$ 1 (i.e. one node). Best response is computed by iterating through all policies, Π_k. A policy,π_k is pruned if $\hat{v}[\pi_k, \pi^{k-}] < threshold + \epsilon$. Thus the best response policy computed by VAX would be at most ϵ away from the optimal best response. Hence the proposition holds for the base case.

Assumption: Proposition holds for a tree of depth d, where $1 \leq$ depth $\leq d$.

We now have to prove that the proposition holds for a tree of depth $d + 1$.

Without loss of generality, lets assume that the root node of this tree has k children. Each of this children is of depth $\leq d$, and hence from the assumption above the error introduced in kth child is $\rho_k \epsilon$, where ρ_k is the number of leaf nodes in kth child of the root. Therefore, $\rho = \sum_k \rho_k$, where ρ is the number of leaf nodes in the tree.

Hence, with VAX the pruning condition at the root agent will be $\hat{v}[\pi_k, \pi^{k-}] < (threshold - \sum_k \rho_k \epsilon) + \epsilon$. However, with SPIDER-ABS the pruning condition would have been $\hat{v}[\pi_k, \pi^{k-}] < threshold$. As long as $\sum_k \rho_k \geq 1$, the root agent in VAX does not prune a policy that was not pruned in SPIDER-ABS. Hence the root agent does not introduce any error in the solution. All the error is thus introduced by children of the root agent, which is $\sum_k \rho_k \epsilon = (\sum_k \rho_k)\epsilon = \rho\epsilon$.

Hence proved. ■

Proposition 14 *For PAX (implemented over SPIDER-ABS) with an input parameter of δ, the solution quality is at least $\frac{\delta}{100} v[\pi^{root+,*}]$, where $v[\pi^{root+,*}]$ denotes the optimal solution quality.*

Proof. We prove this proposition using mathematical induction on the $depth$ of the DFS tree.

Base case: depth $=$ 1 (i.e. one node). Best response is computed by iterating through all policies, Π_k. A policy,π_k is pruned if $\frac{\delta}{100}\hat{v}[\pi_k, \pi^{k-}] < threshold$. Thus the best response policy computed by PAX would be at least $\frac{\delta}{100}$ times the optimal best response. Hence the proposition holds for the base case.

Assumption: Proposition holds for a tree of depth d, where $1 \leq$ depth $\leq d$.

We now have to prove that the proposition holds for a tree of depth $d + 1$.

Without loss of generality, lets assume that the root node of this tree has k children. Each of this children is of depth $\leq d$, and hence from the assumption above the solution quality in the kth child is at least $\frac{\delta}{100} v[\pi^{k+,*}, \pi^{k-}]$ for PAX.

With SPIDER-ABS the pruning condition would have been:
$\hat{v}[\pi_{root}, \pi^{root-}] < \sum_k v[\pi^{k+,*}, \pi^{k-}]$. With PAX, the pruning condition at the root agent will be $\frac{\delta}{100}\hat{v}[\pi_{root}, \pi^{root-}] < \sum_k \frac{\delta}{100} v[\pi^{k+,*}, \pi^{k-}] \Rightarrow \hat{v}[\pi_{root}, \pi^{root-}] < \sum_k v[\pi^{k+,*}, \pi^{k-}]$. Since the pruning condition at the root agent in PAX is the same as the one in SPIDER-ABS, a joint policy that is not pruned in SPIDER-ABS will not be pruned in PAX. Hence there is no error introduced at the root agent and all the error is introduced in the children. Thus the overall solution quality is at least $\frac{\delta}{100}$ of the optimal solution.

Hence proved. ■

7.2 Experimental Results

All our experiments were conducted on the sensor network domain provided in Section 2.1.2. Network configurations presented in Figure 7.3 were used in these experiments. Algorithms that we experimented with as part of this paper include GOA, SPIDER, SPIDER-ABS, PAX and VAX. We compare against GOA because it is the only global optimal algorithm that exploits network structure and considers more than two agents. We performed two sets of experiments: (i) firstly, we compared the run-time performance of the algorithms mentioned above and (ii) secondly, we further experimented with PAX and VAX to study the tradeoff between run-time and solution quality. Experiments were terminated if they exceeded the time limit of 10000 seconds[1].

Figure 7.4(a) provides the run-time comparisons between the optimal algorithms GOA, SPIDER, SPIDER-Abs and the approximate algorithm, VAX with varying epsilons. X-axis denotes the type of sensor network configuration used, while Y-axis indicates the amount of time taken (on a log scale) to compute the optimal solution. The time horizon of policy computation for all the configurations was 3. For each configuration (3-chain, 4-chain, 4-star and 5-star), there are five bars indicating the time taken by GOA, SPIDER, SPIDER-Abs and VAX with 2 different epsilons. GOA did not terminate within the time limit for 4-star and 5-star configurations. SPIDER-Abs dominated the other two optimal algorithms for all the configurations. For instance, for the 3-chain configuration, SPIDER-ABS provides 230-fold speedup over GOA and 2-fold speedup over SPIDER and for the 4-chain configuration it provides 58-fold speedup over GOA and 2-fold speedup over SPIDER. The two approximation approaches, VAX (with ϵ of 10) and PAX (with δ of 80) provided a further improvement in performance over SPIDER-Abs. For instance, for the 5-star configuration VAX provides a 15-fold speedup and PAX provides a 8-fold speedup over SPIDER-Abs.

[1] Machine specs for all experiments: Intel Xeon 3.6 GHZ processor, 2GB RAM

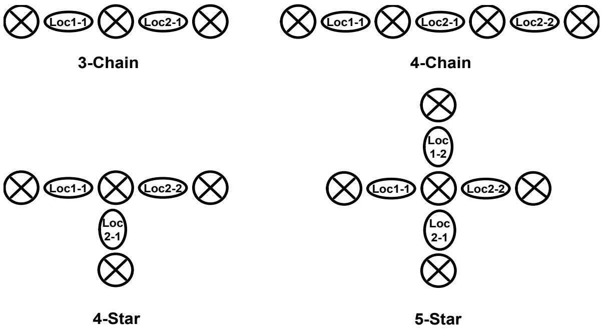


Figure 7.3: Sensor network configurations

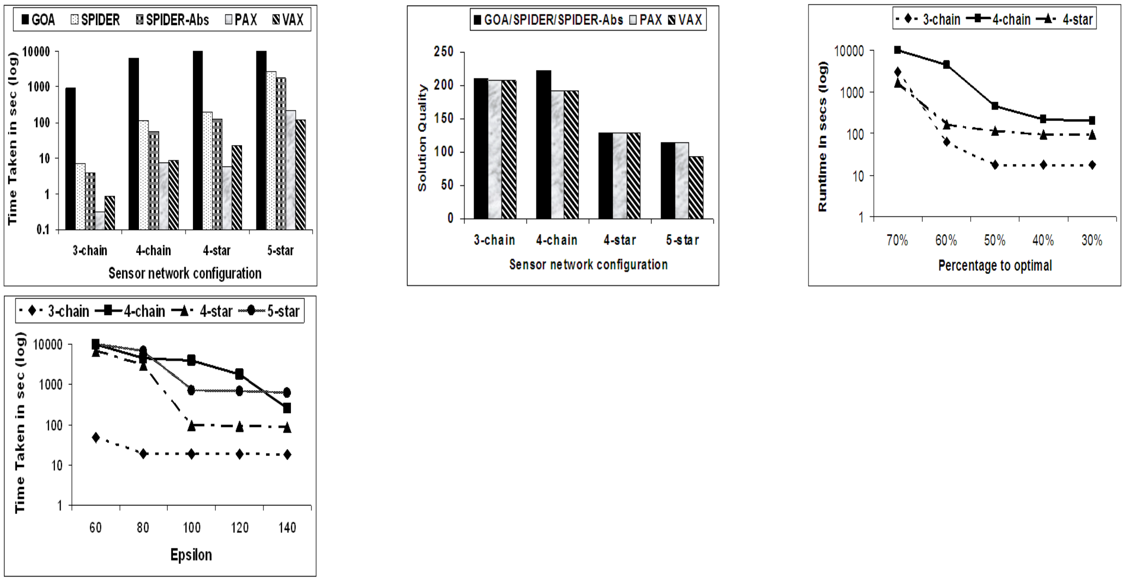


Figure 7.4: Comparison of GOA, SPIDER, SPIDER-Abs and VAX for T = 3 on (a) Runtime and (b) Solution quality; (c) Time to solution for PAX with varying percentage to optimal for T=4 (d) Time to solution for VAX with varying epsilon for T=4

Figures 7.4(b) provides a comparison of the solution quality obtained using the different algorithms for the problems tested in Figure 7.4(a). X-axis denotes the sensor network configuration while Y-axis indicates the solution quality. Since GOA, SPIDER, and SPIDER-Abs are all global optimal algorithms, the solution quality is the same for all those algorithms. With both the approximations, we obtained a solution quality that was close to the optimal solution quality. In 3-chain and 4-star configurations, it is remarkable that both PAX and VAX obtained almost the same quality as the global optimal algorithms. For other configurations as well, the loss in quality was less than 15% of the optimal solution quality.

Figure 7.4(c) provides the time to solution with PAX (for varying epsilons). X-axis denotes the approximation parameter, δ (percentage to optimal) used, while Y-axis denotes the time taken to compute the solution (on a log-scale). The time horizon for all the configurations was 4. As δ was decreased from 70 to 30, the time to solution decreased drastically. For instance, in the 3-chain case there was a total speedup of 170-fold when the δ was changed from 70 to 30. However, the variance in actual solution quality was zero.

Figure 7.4(d) provides the time to solution for all the configurations with VAX (for varying epsilons). X-axis denotes the approximation parameter, ϵ used, while Y-axis denotes the time taken to compute the solution (on a log-scale). The time horizon for all the configurations was 4. As ϵ was increased, the time to solution decreased drastically. For instance, in the 4-star case there was a total speedup of 73-fold when the ϵ was changed from 60 to 140. Again, the actual solution quality did not change with varying epsilon.

Chapter 8

Exploiting structure in dynamics for Distributed POMDPs

In this chapter, I propose a novel technique that exploits structure in dynamics for distributed POMDPs, while planning over a continuous initial belief space. This algorithm builds on the technique proposed in Chapter 3 for exploiting structure in single agent POMDPs and on the "Joint Equilibrium-based Search for Policies" (JESP) algorithm [Nair et al., 2003a] which finds locally optimal policies from an unrestricted set of possible policies, with a finite planning horizon. Not only does this technique exploits structure to improve efficiency, it also addresses a major shortcoming in existing research in Distributed POMDPs: planning for a continuous starting belief region.

In particular, whereas the original JESP performed iterative best-response computations from a single starting belief state, the combined algorithm exploits the single-agent POMDP techniques to perform best-response computations over continuous regions of the belief space. The new algorithm, CS-JESP (Continuous Space JESP) allows for generation of a piece-wise linear and convex value function over continuous belief spaces for the optimal policy of one agent in the distributed POMDP, given fixed policies of other agents — the familiar cup-like shape of this value function [Kaelbling et al., 1998]. The cup-shape implies that when dealing with a continuous starting belief space, agents usually have more than one policy, each of which dominates in a different region of the belief space.

This region-wise dominance highlights the three important challenges addressed in CS-JESP. First, CS-JESP requires computation of best response policies for one agent, given that different policies dominate over different regions of the belief space for the second agent. To efficiently compute best response policies per belief region, it is critical to employ techniques that prune out unreachable future belief states. To that end, we illustrate application of the belief bound techniques [Varakantham et al., 2005] for improved efficiency. Second, owing to these best response calculations for different belief regions, often the policies for contiguous belief regions can be identical. To address this inefficiency, we implement a merging method that combines such adjacent regions with equivalent policies. Third, to improve the performance of the algorithm, we implement region-based convergence, i.e. once policies have converged for a region, these are not considered for subsequent best response computations.

8.1 Continuous Space JESP (CS-JESP)

One of the key insights in CS-JESP is the synergistic interaction between the JESP algorithm for distributed POMDPs and the DB-GIP technique of single agent POMDPs. We illustrate these interactions with a two-agent example in Section 8.1.1, and present key ideas in Section 8.1.2. Further, we describe the algorithm for n agents in Section 8.1.3 and some theoretical guarantees in Section 8.1.4.

Unlike previous work, our work focuses on continuous starting belief spaces and thus requires modifications for policy representation that is traditionally used in distributed POMDP literature. In particular, because different policies may be dominant over different regions in the belief space, we introduce the notion of a *general policy*. A *general policy*, Π_i for an agent i is defined as a mapping from belief regions to policies. Π_i is represented as the set $\{(B_0^1,\pi_i^1),...(B_0^m,\pi_i^m)\}$, where $B_0^1,..,B_0^m$ are belief regions in the starting belief space B_0 and $\pi_i^1,..\pi_i^m$ are the policies that will be executed starting from those regions. Henceforth we refer π_i^k as *specialized policies*. Thus, given a starting belief point $b_0^k \in B_0^k$, agent i on receiving observations $\omega_i^1,\ldots,\omega_i^t$ will perform the action $\pi_i^k(\vec{\omega}_i^t)$ where $\vec{\omega}_i^t = \omega_i^1,\ldots,\omega_i^t$. $\Pi = \langle\Pi_1,\ldots,\Pi_n\rangle$ refers to the joint general policy of the team of agents.

8.1.1 Illustrative Example

For ease of explanation, initially the algorithm is explained with two agents, Agent1 and Agent2. However, as we will show in Section 8.1.3, this algorithm is easily extendable to n agents. Initially, each agent selects a random general policy, Π_i, which will be a singleton set, $\{(B_0, \pi_i)\}$, i.e. a single *specialized policy*,π_i, over the entire starting belief space, B_0. While for expository purposes this example describes policy computations by individual agents, in reality in CS-JESP these computations are performed by a centralized policy generator. CS-JESP begins when one agent, say Agent2, fixes its *general policy* Π_2, and other agent, Agent1, finds the best response for Agent2's *general policy*. Fixing Agent2's *specialized policy*, π_2, Agent1 creates a single agent POMDP with an extended state space, as explained in Section 2.3.3. Agent1 solves this POMDP using DB-GIP technique, explained in Section 3, with starting belief space as B_0, and obtains a new general policy Π_1, containing a set $\{(B_0^1, \pi_1^1), ...(B_0^m, \pi_1^m)\}$. Each $B_0^k \in B_0$ is a belief region and is represented by a minimum and maximum value for each of the $|S| - 1$ dimensions that represent the belief space. Now, Agent1 freezes its *general policy*, Π_1, and Agent2 solves a POMDP for each $\pi_1^j \in \Pi_1$, with the starting belief region as B^j. Thus, Agent2 solves m POMDPs, and obtains a new *general policy* Π_2. At this point, bordering regions in Π_2 that have identical policies are merged. This process continues until the solutions converge, and a local optimal is reached, i.e. no agent can improve its value vectors in any belief region.

Figure 8.1 illustrates the working of the algorithm with the multiagent tiger scenario (Section 8.1.1) for time horizon, $T = 2$. Each tree in Figure 8.1 represents a *specialized policy*. All the trees on the left side of the figure are part of the *general policies* of Agent1, and trees on the right are part of the *general policies* of Agent2. For instance, at the end of iteration 3, both agents contain two specialized policies in their *general policy*. Within each tree (specialized policy), the letter inside each node indicates the action, and edges indicate the observation received. Thus, for the highlighted tree in the top left corner, the root node indicates the Listen(L) action, and upon either observing TL or TR, the *specialized policy* requires the agent to take a Listen(L) action. In this example, belief region over which a *specialized policy* dominates, consists of two numbers, namely the minimum and maximum belief probability of the state SL. These belief regions are indicated below each *specialized policy* in the figure. For instance, for the highlighted tree it is [0,1], but for other trees, regions such as [0.18,0.85] are shown.

The algorithm begins with both agents randomly selecting a *specialized policy* for the entire belief space [0,1]. In iteration 1, Agent2 fixes its *general policy*, and Agent1 comes up with its best response *general policy*. For calculating the best response, the Agent1 solves a POMDP with the starting belief range as [0,1], since Agent2's general policy is defined over this range. After the first iteration, Agent1 contains three *specialized policies* as part of its *general policy*, dominating over ranges [0,0.15], [0.15,0.85], [0.85,1]. In iteration 2, Agent1 fixes its *general policy*, and Agent2 begins its best response calculation with region [0,0.15]. For this range [0,0.15], Agent2 has only one dominant *specialized policy* and same is the case for [0.85,1]. However, for the range [0.15,0.85], Agent2 has two dominant *specialized policies*, one that dominates in the range [0.15,0.5], and the other that dominates in the range [0.5,0.85]. Thus after iteration 2, Agent2 has four *specialized policies* as part of its best response *general policy*. However, regions highlighted (with dotted rectangular boxes) have identical policies and thus after merging we are left with only two *specialized policies*. This algorithm continues with Agent2 fixing its *general policy* at iteration 3. Finally at convergence, each agent contains two *specialized policies* as part of their *general policies*.

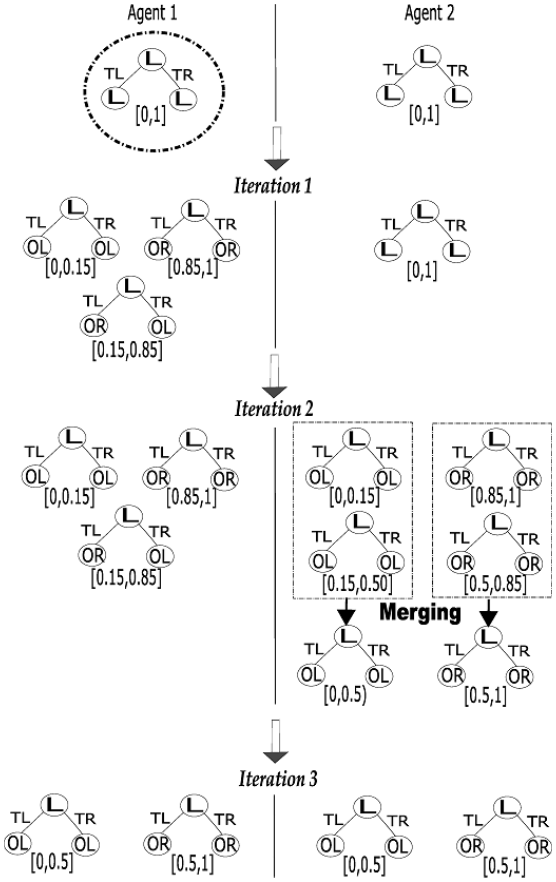


Figure 8.1: Trace of the algorithm for T=2 in Multi Agent tiger example with a specific starting joint policy

8.1.2 Key Ideas

In this section, we explain in detail the key ideas in the CS-JESP algorithm, namely: (a) JESP and DB-GIP synergy; (b) Calculation of dominanant belief regions for *specialized policies*; (c) Region-based convergence; and (d) Merging of adjacent regions with identical *specialized policies*.

JESP and DB-GIP synergy: Both the DS and DB techniques of DB-GIP can provide significant performance improvements in CS-JESP. First, with respect to DS, JESP's state space is dynamic, where the set of states reachable at time t, e_i^t differ from the set of states at $t+1$, e_i^{t+1}. DS can exploit this dynamism by computing dominant policies at time t over the belief space generated by the states in e_i^t thus reducing the dimensionality of the state space considered. For instance, in Figure 2.4, we have two initial states e_1^1= SL or SR, while there are four states

e_1^2, e.g.$SL(TL), SL(TR)$ etc. Given a time horizon of T=2, instead of constructing a belief space over (2+4=) 6 dimensions, DS will lead to constructing a belief space over two states at the first time step and four states over the second time step. Such dimensionality reduction leads to significant speedups in CS-JESP. Second, with respect to DB, each agent solves a POMDP over the belief regions in the *general policies* of the other agents. DB is able to exploit forward projections of such starting belief regions to bound the maximum probabilities over states, and thus again restrict the belief space over which dominant policies are planned per belief region, obtaining additional speedups. For instance in Figure 8.1, at iteration2, Agent2 solves three POMDPs — these POMDPs are defined over extended states given three separate fixed policies of Agent1— one with the starting belief region as [0,0.15], another with [0.15,0.85], and a third with [0.85,1]. Thus, in solving the POMDP starting with the belief range [0,0.15], DB helps prune all the unreachable portions of the belief space given that the starting range is [0,0.15]. In all three POMDPs, the belief region is narrower compared to [0,1].

Region-based convergence: Given continuous initial belief space, we obtain value vectors (vector containing values for all the states) for all the belief regions in the *general policy*. Thus, convergence is attained when for all agents the value vectors at the current iteration for all the belief regions are equal to those in the previous iteration. For instance, in Figure 8.1, the convergence is attained in the fourth iteration, with the *general policy* of Agent2 containing the two exact same *specialized policies* from iteration 3. However, once one region has converged — the value vectors for all agents do not change from one iteration to the next for that region — CS-JESP will not test that region further for convergence, but only continue changing policies in regions that have failed to converge.

Merging of adjacent regions with identical *specialized policies*: Merging such regions can be important as the other agent would have to solve fewer number of POMDPs in the next iteration. For instance, in the *general policy* of Agent2 before merging at iteration 2, belief regions [0,0.15] and [0.15,0.5] have identical specialized policies. Similarly, regions [0.5, 0.85] and [0.85,1] have identical specialized policies. Thus Agent2 has only two *specialized policies* after merging (instead of four before merging) and this leads to agent1 solving two instead of four POMDPs at iteration 3.

Merging requires identifying regions adjacent to each other. In the Tiger domain, this is done by doing adjacency check for regions along one dimension. However, finding bordering regions in a $|S|$ dimensional state space requires comparisons along $|S| - 1$ dimensional space.

Calculation of dominant belief regions for *specialized policies*: One standard way of representing solutions in single agent POMDPs is through value vectors. In this representation, the best policy for a belief point, b, is computed by testing for a vector that provides the maximum expected value for that belief point.
$\pi_1^* \leftarrow argmax_{\pi \in \{\pi_1\}} v_\pi \cdot b.$

However, in CS-JESP, one agent uses the belief regions of the other agent to calculate the best responses over each of those belief regions. We develop a linear program to address the dominant belief region computation for each policy. Algorithm 24 computes the maximum belief probability of a state, s_j, where a policy or value vector,v dominates all the other policies or value vectors, $\mathcal{V} - v$ in the final policy. Constraint 1 in Algorithm 24, computes points where v dominates all the other vectors in $\mathcal{V}$. Objective function of the algorithm is a maximization over $b(s_j)$, thus finding highest possible belief probability for state s_j amongst all those dominating points. In a similar way, the minimum for s_j can be found by doing a minimize, instead of maximize, in line 1 of the LP. The belief region is calculated by solving these max, min LPs for each state $s_j \in S$. Thus, requiring $2 * |\mathcal{V}| * |S|$ number of LPs to be solved for the computation of an entire belief region.

```
Algorithm 24 MAXIMUMBELIEF($s_j, v, \mathcal{V}, B^{min}, B^{max}$)
  Maximize $b(s_j)$
  subject to constraints
    1. $b.(v - v') > 0, \forall v' \in \mathcal{V} - v$
    2. $\Sigma_{s \in S} b(s) = 1$
    3. $B^{min}(s) < b(s) < B^{max}(s), \forall s \in S$
```

8.1.3 Algorithm for n agents

In this section, we present the CS-JESP algorithm (Algorithm 25) for n agents. In the initialization stage (lines 1-4), each agent i has only one belief region that corresponds to its entire belief space (Π'_i.*beliefPartition*). Also, each agent has a single randomly selected *specialized policy*, $\Pi'_i.\pi[\langle [0,1], \ldots, [0,1] \rangle]$ (i.e. π is the *specialized policy*), for the entire belief space (line 3). Every *general policy* has "*count*" for each belief region, to track the convergence of policies in that belief region (region-based convergence) — if the *count* reaches n then the region has converged, because no agent will change any further. The flag "*converged*" monitors if joint *general policies* in all the regions have converged.

In each iteration (one execution of lines 6-23) of Algorithm 25, we choose an agent i and find its optimal response to the fixed *general policies* of the remaining agents by calling OPTIMALBESTRESPONSE(). This is repeated until no agent acting alone can improve upon the joint expected reward by changing its own *general policy*.

Although each agent i starts off with the same belief set partition, Π'_i.*beliefPartition*, this will not be true after calling OPTIMALBESTRESPONSE() as seen in Figure 8.1. The function UPDATEPARTITION() (Algorithm 26) is responsible for creating a new belief set partition for an agent i, depending on the belief regions of the other $n-1$ agents. This new belief set partition is obtained by splitting the overlapping belief regions of the $n-1$ agents, in a way that no two resulting belief regions, which now belong to this partition, overlap. Furthermore, this function computes the Π_i.*count* for all the new regions, from the $count$ values for the regions in Π'_j, where j was the free agent in the last iteration (i.e the agent who computed the best response in the last iteration).

FINDNEWPARTITION() (Algorithm 27) takes two arguments, (i) *partition* and (ii) a belief region, *br*, and it generates all feasible partitions from the two arguments. To illustrate the working of this function, we provide an example with three states $\{s_1, s_2, s_3\}$. Belief regions in the corresponding belief space can be represented with minimum and maximum belief probabilities for just s_1 and s_2, i.e. $\{(b^{min}[s_1], b^{max}[s_1])$, $(b^{min}[s_2], b^{max}[s_2])\}$. For example, let *partition* $=\{\langle[0, 0.8], [0.5, 0.9]\rangle\}$ (has only one region) and *br* $= \{[0.4, 0.9], [0.3, 0.6]\}$. In the first step (line 3), partitions are found for each state, s_i separately. Thus, for the first state, s_1, $[0, 0.8]$ and $[0.4, 0.9]$ yields partitions, $[0, 0.4]$, $[0.4, 0.8]$, $[0.8, 0.9]$. Similarly for the second state, s_2, the partitions found are $[0.3, 0.5]$, $[0.5, 0.6]$,$[0.6, 0.9]$. In the second step (line 4), we compute the cross product of these individual dimension partitions. This gives rise to nine belief regions, viz. $\{[0, 0.4], [0.3, 0.5]\}$, ..., $\{[0.8, 0.9]$, $[0.4, 0.8]\}$, $\{[0.8, 0.9], [0.8, 0.9]\}$. Finally, in the third step (line 5), we prune regions which do not contain any valid points, i.e. $\sum_{s \leq |S|-1} b^{min}[s] > 1$. For instance, the region {[0.8,0.9],[0.4,0.8]} can be pruned, because a belief point in this region has probability of atleast 1.2 $(= 0.8 + 0.4)$.

The function OPTIMALBESTRESPONSE() (Algorithm 28) is then called separately for each belief region in agent i's belief set partition. It returns a new partitioniong of the initial belief space and the optimal policy for each belief region in this partition. CONSTRUCTEXTENDEDPOMDP() constructs a POMDP with extended state space, as explained in Section 2.3.3, while the function CALCULATEBELIEFREGION() computes the belief regions where each vector v $(\in \mathcal{V})$ dominates.

After computing best responses, CS-JESP() ensures that the number of belief partitions obtained are finite (in lines 14-16) and that Π.*count* is updated correctly for each belief region (in lines 18-21).

It is possible that an agent's best response in adjacent belief regions is the same policy. The function MERGEBELIEFREGIONS() (Algorithm 8.1.3) is responsible for merging such kind of regions (lines 4-7). Further, once the policies in a belief region have converged, that region is not considered for subsequent merging phases (first part of the condition on line 4).

Algorithm 25 CS-JESP()

```
 1: for $i \leftarrow 1$to $n$ do
 2:   $\Pi'_i.\textit{beliefPartition} \leftarrow \{\langle [0,1], \ldots, [0,1] \rangle\}$
 3:   $\Pi'_i.\pi[\langle [0,1], \ldots, [0,1] \rangle] \leftarrow$ random specialized policy
 4:   $\Pi'_i.\textit{count}[\langle (0,1), \ldots, (0,1) \rangle] \leftarrow 0$
 5: end for
 6: $\textit{converged} \leftarrow$ false; $i \leftarrow n$;
 7: while $\textit{converged} =$ false do
 8:   $i \leftarrow (i\ \textit{MOD}\ n) + 1$; $\textit{converged} \leftarrow$ true
 9:   UPDATEPARTITION$(i, \Pi_i, \Pi')$
10:   for all $br$ in $\Pi_i.\textit{beliefPartition}$ do
11:     if $\Pi_i.\textit{count}[br] < n$ then
12:       $\textit{converged} \leftarrow$false
13:       $\{\Pi_i, \textit{regions}\} \leftarrow$ OPTIMALBESTRESPONSE$(i, \Pi', br)$
14:       for all $br_1$ in $\textit{regions}$ do
15:         $\pi \leftarrow \Pi_i.\pi[br_1]$; REMOVE$(\Pi_i, br_1)$
16:         for $\textit{dim} \leftarrow 1$ to $|S| - 1$ do
17:           $br_1[\textit{dim}] \leftarrow$ ROUNDOFF$(br_1[\textit{dim}], \textit{precision})$
18:         end for
19:         if VOLUME$(br_1) > 0$ then
20:           ADD$(\Pi_i.\textit{beliefPartition}, br_1, \pi)$
21:           if $\Pi_i.\pi[br_1] = \Pi'_i.\pi[br]$ then
22:             $\Pi_i.\textit{count}[br_1] \leftarrow \Pi'_i.\textit{count}[br] + 1$
23:           else
24:             $\Pi_i.\textit{count}[br_1] \leftarrow 1$
25:           end if
26:         end if
27:       end for
28:     end if
29:   end for
30:   MERGEBELIEFREGIONS$(\Pi_i)$
31:   $\Pi'_i \leftarrow \Pi_i$
32: end while
33: return $\Pi$
```

Algorithm 26 updatePartition(i, Π)

```
$\Pi_i \leftarrow \Pi_{(i \text{ MOD } n)+1}$
for all $j$ in$\{1, \ldots, n\} - \{i, (i\ MOD\ n) + 1\}$ do
    for all $br_1$ in $\Pi_j.beliefPartition$ do
        if $\Pi_i.count[br_1] < n$ then
            $\Pi_i.beliefPartition \leftarrow$ findNewPartition($\Pi_i.beliefPartition, br_1$)
        end if
    end for
end for
if $i = 1$ then $j \leftarrow n$else $j \leftarrow i - 1$
for all $br_2$ in $\Pi_i.beliefPartition$ do
    $br_3 \leftarrow$ overlappingRegion($\Pi_j.beliefPartition, br_2$)
    $\Pi_i.count[br_2] \leftarrow \Pi_j.count[br_3]$
end for
return
```

Algorithm 27 findNewPartition(($partition, br$))

```
$newPartition \leftarrow \emptyset$
for $dim \leftarrow 1$ to $|S| - 1$ do
    $1DPartition \leftarrow$ splitDimension($dim, br, partition$)
    $newPartition \leftarrow$ crossProduct($newPartition$, $1DPartition$)
    $newPartition \leftarrow$ prune($newPartition$)
end for
return $newPartition$
```

Algorithm 28 optimalBestResponse(i, Π', br)

```
$k \leftarrow 0$
$extendedPOMDP \leftarrow$ constructExtendedPOMDP($i, \Pi', br$)
$\{\mathcal{V}, \pi^{new}\} \leftarrow$ DB-GIP($extendedPomdp, br$)
for $j \leftarrow 1$ to $\mathcal{V}.size$ do
    $v \leftarrow \mathcal{V}[j]; \mathcal{V}' \leftarrow \mathcal{V} - v$
    $beliefPartition[k] \leftarrow$ calculateBeliefRegion($v, \mathcal{V}', br$)
    $\Pi_i.beliefPartition[k] \leftarrow beliefPartition[k]$
    $\Pi_i.\pi[beliefPartition[k]] \leftarrow \pi^{new}[j]; k \overset{+}{\leftarrow} 1$
end for
return $\{\Pi_i, beliefPartition\}$
```

Algorithm 29 MERGEBELIEFREGIONS(Π_i)

```
for each $b_1$ in $\Pi_i$.beliefPartition do
  if $\Pi_i$.count($b_1$) $< n$ then
    for each $b_2$ in $\Pi_i$.beliefPartition do
      if $\Pi_i$.count($b_2$) $< n \wedge \Pi_i.\pi[b_1] = \Pi_i.\pi[b_2]$ then
        if ISADJACENT($b_1, b_2$) then
          $b \leftarrow$MERGEREGIONS($b_1, b_2$)
          ADD($\Pi_i$.beliefPartition, $b, \Pi_i.\pi[b_1]$)
          $\Pi_i$.count[$b$] $\leftarrow$ min($\Pi_i$.count[$b_1$], $\Pi_i$.count[$b_2$])
          REMOVE($\Pi_i, b_1$); REMOVE($\Pi_i, b_2$)
        end if
      end if
    end for
  end if
end for
return $\Pi_i$
```

8.1.4 Theoretical Results

In the following proofs, we use "iteration" to mean one execution of the "while" loop (lines 5-23) of Algorithm 25, n for the number of agents, and "free agent" to denote the i^{th} agent for that iteration.

Proposition 15 *In CS-JESP, the joint expected reward for all starting belief points is monotonically increasing with each iteration.*

Proof Sketch. In every iteration, each starting belief point must belong to one of the regions in the belief partition of the free agent. Each such belief region corresponds to one of the value vectors, calculated by a call to OPTIMALBESTRESPONSE(Algorithm 28). Since DB-GIP is optimal, these vectors should either equal or dominate the vectors at the previous iteration, in all belief regions.■

Proposition 16 *CS-JESP will terminate iff the joint policy has converged in all the free agent's belief regions.*

Proof Sketch. By construction, CS-JESP (Algorithm 25) terminates iff *converged* $=$ true, which will happen iff Π_i.*count*$[br] \geq n$, for all belief regions *br* of the free agent i. Π_i.*count*$[br] \geq n$ iff the joint policy for the region *br* remains constant for n iterations. In order for the joint *general policy* to remain constant for n iterations, OPTIMALBESTRESPONSE() should return identical *specialized policies* (to those in previous iteration) for all the belief regions , for $n - 1$ free agents. This happens when no one agent can improve the global value by altering its *general* *policy*, i.e. when local optima is attained. Furthermore, we round off each dimension of a belief region to *precision* decimal spaces, and hence the number of possible belief regions cannot grow indefinitely.■

From Propositions 15 and 16, we can conclude that CS-JESP will always terminate. At termination, the joint policy will be locally optimal as long as none of the belief regions returned by OPTIMALBESTRESPONSE were eliminated by the ROUNDOFF procedure.

8.2 Experimental Results

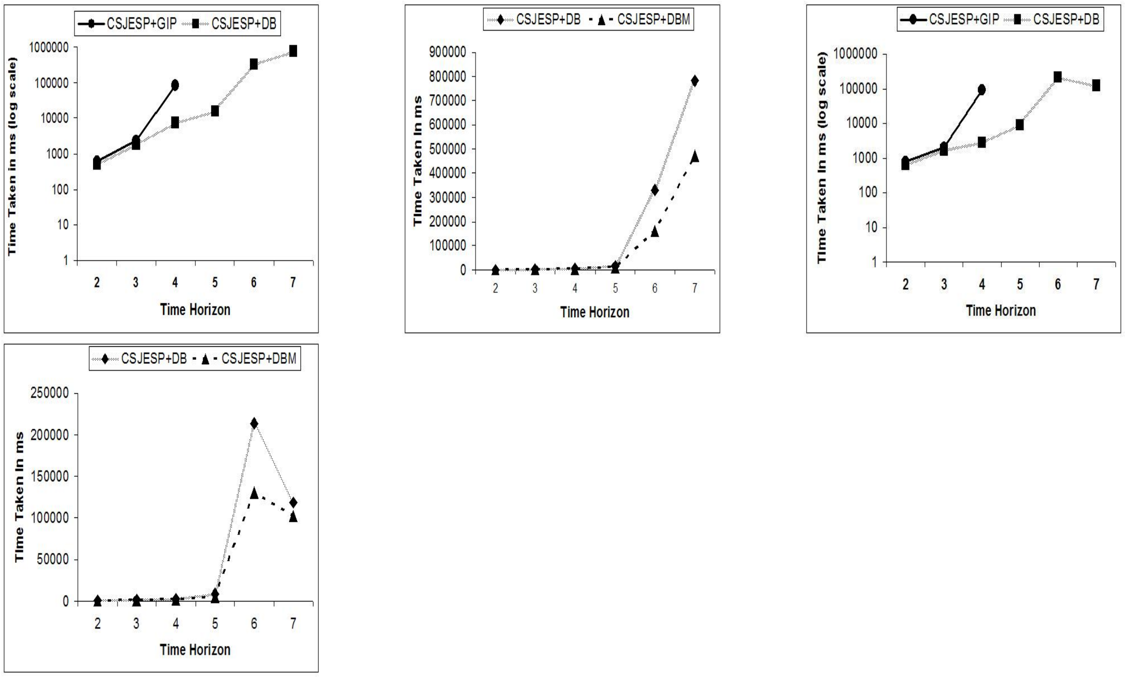


Figure 8.2: Comparison of (a) CSJESP+GIP, and CSJESP+DB for reward structure 1 (b) CSJESP+DB, and CSJESP+DBM for reward structure 1 (c) CSJESP+GIP, and CSJESP+DB for reward structure 2 (d) CSJESP+DB, and CSJESP+DBM for reward structure 2

This section provides three types of evaluations for CS-JESP using the multiagent tiger domain [Nair et al., 2003a]. The first experiment focuses on run-time evaluations. We provide a comparison of three techniques: (i) CS-JESP+GIP: is the basic version of the combination of the JESP and the value iteration algorithm, GIP of single agent POMDPs. (ii) CS-JESP+DB, is JESP with DB-GIP. (iii)CS-JESP+DBM is CS-JESP+DB with the merging enhancement. Results of this experiment are shown in Figure 8.2. We experiment with two separate reward structures (presented in [Nair et al., 2003a]). Figure 8.2(a) and Figure 8.2(b) focus on reward structure1, while

Figure 8.2(c) and Figure 8.2(d) focus on reward structure2. In Figure 8.2(a), x-axis plots varying time horizon while y-axis plots run-time in milliseconds on log-scale[1]. In Figure 8.2(b), x-axis again plots time horizon, but the y-axis plots run-time in milliseconds (no log-scale is used). Time limit for the problems was set at 7,500,000 ms, after which they were terminated. Figure 8.2(a) and Figure 8.2(c) refer to comparisons between CS-JESP+GIP and CS-JESP+DB, while Figure 8.2(b) and Figure 8.2(d) refer to comparisons between CS-JESP+DB and CS-JESP+DBM for the two reward structures.

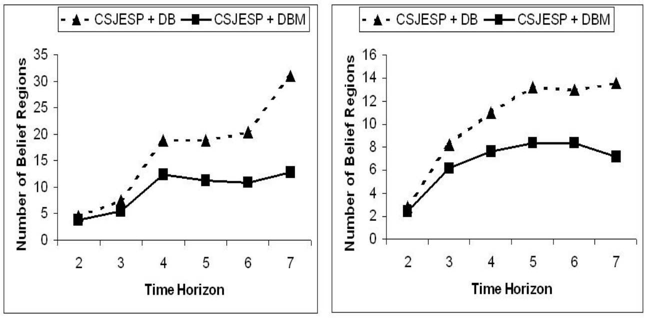


Figure 8.3: Comparison of the number of belief regions created in CS-JESP+DB and CS-JESP+DBM for reward structures 1 and 2

Figure 8.2(a) shows that CS-JESP+GIP did not terminate within the specified time limit after T=4. However, CS-JESP+DB converged to the solution even for T = 7, within the specified time limit. Even in cases where CS-JESP+GIP terminates, CS-JESP+DB provides significant speedups. For instance, in Figure 8.2(a), at T = 4, while CS-JESP+GIP takes in 83717.8 ms, CS-JESP+DB takes only 7345.2 ms leading to a speedup of 11.4 fold. Similar conclusions can be drawn from Figure 8.2(c). These results illustrate the synergy of JESP and DB-GIP, and the suitability of CS-JESP to take advantage of DB-GIP.

Figure 8.2(b) shows that CS-JESP+DBM provides further speedups over CS-JESP+DB, as time horizon increases. For instance at T=7 in Figure 8.2(b), merging in CS-JESP+DBM provided 1.66 fold speedup over CS-JESP+DB. Similar results are obtained with reward structure 2 in Figure 8.2(d), thus establishing the utility of merging contiguous regions with identical policies. In Figure 8.2(d) T=7 post merging show a faster execution compared to the T=6 results post merging. This occurs because the number of iterations of CS-JESP required for convergence at T=7 are lower (6) compared with iterations at T=6 (11).

Our second evaluation in Figure 8.3 focuses on understanding the speedups due to merging in CS-JESP+DBM. The number of belief regions present in the final solution is an indicator of the

[1] Machine specs for all experiments: Intel Xeon 3.6 GHZ processor, 2GB RAM

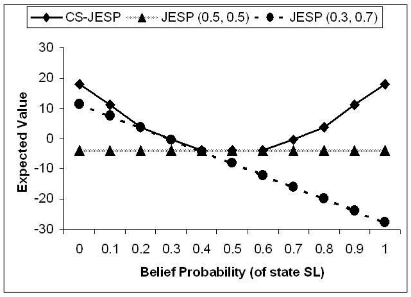


Figure 8.4: Comparison of the expected values obtained with JESP for specific belief points and CS-JESP

number of single agent POMDPs getting solved at each iteration. The x-axis in the figures represents the time horizon, while the y-axis is the number of belief regions. Thus in Figure 8.3, for a time-horizon of 7, using CSJESP+DB led to 31 belief regions, whereas using CSJESP+DBM led to 13 belief regions, a 2.39-fold reduction in the number of belief regions considered. Furthermore, we see that increasing the time horizon leads to increasing reduction in the number of belief regions with CS-JESP+DBM when compared to the number with CS-JESP+DB. Effect of the number of belief regions on the time taken increases with time horizon, because the single agent POMDPs expand in size with the time horizon. This provides the explanation for the timing results for CS-JESP+DBM in Figure 8.2(b) and Figure 8.2(d).

Our third evaluation focused on illustrating that CS-JESP achieves what it set out to do — generating policies over continuous initial belief space as shown in Figure 8.4. Belief space (in this domain belief probability of SL) is denoted on the x-axis, while the expected value of the policy is depicted on the y-axis. CS-JESP provides a general policy where the expected value is represented by a “CUP”-shape. There are five different policies represented in the cup, each dominant over a single belief region. The figure also indicates that if we were to approximate this entire general policy with a single policy over a single starting belief state, e.g. with JESP, then results may be arbitrarily worse. For instance with JESP (0.3, 0.7), the value at (1, 0) is -27, while the value generated with CS-JESP is 18, a difference of 45. With JESP (0.5, 0.5), the value at (1,0) is -4, where CS-JESP attains a value of 18, a difference of 22.

Of course, we may sample several belief points with JESP and then for a new belief point provide a policy from the nearest sample. Such a proposed heuristic approach naturally leads to our fourth evaluation comparing CS-JESP runtime to an approach that samples the belief space. This evaluation is not meant to be a precise comparison of JESP and CS-JESP, instead the aim is to show that the run-time results for sampled JESP would be comparable to the run times of CS-JESP. In Table 8.1, we show the run times of JESP and CS-JESP for T = 6, and T=7 for reward structure1. To replicate the policy obtained with CS-JESP, JESP would have to sample atleast as many times as the number of belief regions in the final policy of CS-JESP. For instance, for T=7, the number of samples required for JESP would be thirteen (from Figure 8.3). Table 8.1 shows an estimate of such a sampled JESP technique, given runtime results from [Nair et al., 2004]. We see that CS-JESP run-times are comparable, yet CS-JESP provides guarantees on these results that are unavailable with sampling.

	CS-JESP	**JESP**	**Sampled Regions**	**Sampled JESP**
T = 6	160336	15000	11	165000
T = 7	470398	73000	13	949000

Table 8.1: Comparison of run times (in ms) for JESP and CS-JESP

Chapter 9

Related Work

There are three major areas of related work. The first is speeding up POMDP policy computation, and in particular value iteration algorithms, while the second is related work where agents are deployed in monitoring and assisting humans, and must plan in the presence of uncertainty to assist individual humans or teams. The third area of related work is distributed POMDPs.

9.1 Related work in POMDPs

There are a wide variety of techniques for generating policies for POMDPs. These techniques can be categorized into off-line and on-line techniques. Whereas we consider off-line approaches as planning for any belief state within a given range (without knowledge of an agent's current belief state), on-line approaches focus on exploring reachable belief states starting only from an agent's current belief state. Off-line techniques can be further categorized into exact and approximate algorithms; although some approximate techniques may also be converted into on-line techniques. We first focus on offline, exact algorithms and then on approximate algorithms, and finally discuss on-line algorithms.

Generalized Incremental Pruning (GIP) [Cassandra et al., 1997a] has been one of the efficient exact baseline algorithms, that was experimentally shown to be superior to other exact algorithms [Kaelbling et al., 1998]. We have already presented GIP in detail in the background section. Recent enhancements to the GIP algorithm, particularly the Region Based Incremental Pruning (RBIP) [Feng and Zilberstein, 2004a, 2005] provides significant speedups. The key idea in RBIP is the use of witness regions (earlier idea of witness was presented in [Cassandra et al., 1997b]) for cross sums. While these exact algorithms have improved the basic value iteration algorithm considerably, as discussed earlier, they are unable to scale to the problems of interest in key domains. Indeed, as shown in our experimental analysis, we could not generate policies with GIP within our cutoff for most of our problems. This problem stems in part because these algorithms plan for unreachable parts of the belief space. Our work complements these existing algorithms by proving "wrapping" techniques, thus complementing the strengths of current approaches. Indeed, the advantages of our "wrappers" (DS, DB, DDB) can be combined with these existing algorithms, as we illustrated by adding our techniques to both the GIP and RBIP algorithms. Our approximation technique can also be used to enhance these algorithms.

Other exact and approximate algorithms have also attempted to exploit properties of the domain to speedup POMDPs, e.g. [Boutilier and Poole, 1996] focus on compactly representing dynamics of a domain. These compact representations however do not seem to have advantages in terms of speedups [Kaelbling et al., 1998]. A hybrid framework that combines MDP-POMDP problem solving techniques to take advantage of perfectly and partially observable components of the model and subsequent value function decomposition was proposed by [Hauskrecht and Fraser, 2000]. This method of separating perfectly and partially observable components of a state does reachability analysis on belief states. However: (i) their analysis does not capture dynamic changes in belief space reachability; (ii) their analysis is limited to factored POMDPs; (iii) no speedup measurements are shown. This contrasts with our work which focuses on dynamic changes in belief space reachability and its application to both flat and factored state POMDPs. [Feng and Hansen, 2004] provide approaches to reduce the dimensionality of the α-vectors based on the equality of values of states. This method does not provide speedups in the TMP domain, as there are very few instances where there are alpha vectors with states having equal values.

Because of the slowness of exact algorithms at solving even small problems, significant amounts of research in POMDPs has focussed on approximate algorithms. While there is an entire space of algorithms to report in this arena but point-based [Smith and Simmons, 2005; Pineau et al., 2003], policy search [Braziunas and Boutilier, 2004; Poupart and Boutilier, 2004; Menleau et al., 1999], and grid [Hauskrecht, 2000b; Zhou and Hansen, 2001] approaches dominate other algorithms. Since discussion about point-based approaches has already been presented in Section 2, here we concentrate on other approaches. Policy-search approaches typically employ a finite-state controller, to represent the policy, that is updated until convergence to a stable controller. By restricting the size of these finite state controllers, performance improvements are obtained in these algorithms. Grid-based methods are similar to point-based approaches, with the difference that they maintain "values" at belief points, as opposed to "value gradients" in point-based techniques. Though these approaches can solve larger problems, many of them provide loose (or no) quality guarantees on the solution, which is a critical weakness in domains of interest in our work. For example, quality guarantees are important for agent assistants to gain the trust of a human user.

Another approximate approach that attacks scalability is the dimensionality reduction technique, which fundamentally alters the belief space itself [Roy and Gordon, 2002]. This work applies E-PCA (an improvement to Principal Component Analysis) on a set of belief vectors, to obtain a low dimensional representation of the original state space. Though this work provides huge reduction of dimension (state space), it does not provide any guarantees on the quality of solutions. A more crucial issue is the dynamic evolution of the reachable regions of the belief space. E-PCA does not capture this dynamic evolution, while our work focuses on and capture such evolution in the reachable regions of the belief space.

Turning now to on-line algorithms for POMDPs, algorithms such as Real-time Belief Space Search (RTBSS) [Paquet et al., 2005] are offered as on-line approaches for solving POMDPs, which explore reachable belief states starting only from an agent's current belief state. On-line approaches clearly save effort by avoiding computation of policies for every possible situation an agent could encounter. For instance, starting with an initial belief state, the RTBSS algorithm does a branch-and-bound search over belief-states, finding the best action at each cycle. However, in order to cut down time to find an action online, RTBSS must cut-down the depth of its search — the deeper the search in belief states, the more expensive it is online. Unfortunately, such shallow search leads to lower quality solutions; while deeper searches consume precious time on-line.In domains such as disaster rescue (including disaster rescue simulation domains), it would appear that such on-line planning may not provide an appropriate tradeoff. In particular, since quality may be related to crucial aspects of the domain, such as saving civilians, obtaining lower quality solutions just to avoid off-line computation may not be appropriate. Furthermore, spending time on-line may waste critical moments particularly when civilians are injured, and time is of the essence in saving such civilians. Also, because by definition these on-line techniques require knowledge of the belief state, Indeed, in such domains, there may be sufficient time available off-line to generate a policy of high enough quality.

9.2 Related work in Software Personal Assistants

Several recent research projects have focused on deploying personal assistant agents to monitor and assist humans, and must plan in the presence of uncertainty to assist individual humans or teams [Scerri et al., 2002; Magni et al., 1998; Leong and Cao, 1998]. For instance [Scerri et al., 2002] have focused on software assistants that assist humans in offices, in rescheduling meetings or deciding presenters for research meetings. [Magni et al., 1998] focuses on therapy planning, considering the dynamic evolution of a therapy for patients. However, these research efforts have often used MDPs rather than POMDPs, thus assuming away observational uncertainty, that is a key factor in realistic domains.

Among software personal assistants that have relied on POMDPs, [Hauskrecht and Fraser] apply POMDPs for medical therapy planning for patients with heart disease. They note that MDPs fail to capture the situation in their domain where the underlying disease is hidden, can only be observed indirectly via a series of imperfect observations. POMDPs provide a useful tool to overcome this difficulty by enabling us to model the observational uncertainty, but at a high computational cost. To overcome this challenge, [Hauskrecht and Fraser] rely on on several approximation techniques improve the computational complexity of these POMDPs. We have discussed the relationship of our work to this approximation technique in the previous section.

Similarly, [Pollack et al., 2003a] apply POMDPs in mobile robotic assistant, developed to assist elderly individual. The high-level control architecture of the robotic assistant is modeled as a POMDP. Once again the authors, via experiments, illustrate the need to take into account observational uncertainty during planning, and hence the need for POMDPs, e.g an MDP controller in similar circumstances leads to more errors. However, given the large state space encountered, exact algorithms for the POMDP are ruled out. Instead, a hierarchical version of the POMDP is actually used to generate an approximation to the optimal policy. The techniques introduced in our article is in essence complementary to the research reported here, providing techniques to speedup POMDP policy computation, potentially even in the hierarchical context.

9.3 Related work on Distributed POMDPs

Here we have two categories of related work:

Related work for generating policies for distributed POMDPs given a single initial belief point: as mentioned earlier our work is related to key DCOP and distributed POMDP algorithms, i.e., we synthesize new algorithms by exploiting their synergies. Here we discuss some other recent algorithms for locally and globally optimal policy generation for distributed POMDPs. For instance, [Hansen et al., 2004a] present an exact algorithm for partially observable stochastic games (POSGs) based on dynamic programming and iterated elimination of dominant policies. [Montemerlo et al., 2004] approximate POSGs as a series of one-step Bayesian games using heuristics to find the future discounted value for actions. We have earlier discussed [Nair et al., 2003a]'s JESP algorithm that uses dynamic programming to reach a local optimal. Another technique that computes local optimal policies in distributed POMDPs is Paruchur *et al.* Paruchuri et al. [2006]'s Rolling Down Randomisation (RDR) algorithm. However, Paruchur *et al.* have studied this in the context of generating randomized policies.

In addition, [Becker et al., 2004]'s work on transition-independent distributed MDPs is related to our assumptions about transition and observability independence in ND-POMDPs. These are all centralized policy generation algorithms that could benefit from the key ideas in ND-POMDPs — that of exploiting local interaction structure among agents to (i) enable distributed policy generation; (ii) limit policy generation complexity by considering only interactions with "neighboring" agents. [Guestrin et al., 2002], present "coordination graphs" which have similarities to constraint graphs. The key difference in their approach is that the "coordination graph" is obtained from the value function which is computed in a centralized manner. The agents then use a distributed procedure for online action selection based on the coordination graph. In our approach, the value function is computed in a distributed manner. [Dolgov and Durfee, 2004] exploit network structure in multiagent MDPs (not POMDPs) but assume that each agent tried to optimize its individual utility instead of the team's utility.

Related work for a continuous initial belief space: [Becker et al., 2003] present an exact globally optimal algorithm – the coverage set algorithm for transition-independent distributed MDPs. However unlike CS-JESP, this algorithm starts from a particular known initial state distribution. Hansen *et al.* Hansen et al. [2004b] and Szer *et al.* Szer et al. [2005] are techniques that compute optimal solutions without making any assumptions about the domain. Hansen *et al.* present an algorithm for solving partially observable stochastic games (POSGs) based on dynamic programming and iterated elimination of dominant policies. Though this technique provides a set of equillibrium strategies in the context of POSGs, it is shown to provide exact optimal solutions for decentralized POMDPs. Szer *et al.* Szer et al. [2005] provide an optimal heuristic search method for solving Decentralized POMDPs with finite horizon (given a starting belief point). This algorithm is based on the combination of classical heuristic search algorithm, A^* and decentralized control theory. Heuristic functions (upper bounds) required in A^* are obtained by approximating a decentralized POMDP as a single agent POMDP and computing the value function quickly. This algorithm are important from a theoretical standpoint, but because of the inherent complexity of finding an exact solution for general distributed POMDPs, this algorithm does not scale well. Another approach that computes global optimal solutions is presented in Ranjit *et al.* Nair et al. [2003b]. However this approach computes optimal policy in the context of a given BDI (Belief Desire Intention) team plan.

Among locally optimal approaches, [Peshkin et al., 2000b] use gradient descent search to find local optimum finite-controllers with bounded memory. Their algorithm finds locally optimal policies from a limited subset of policies, with an infinite planning horizon. Their work does not consider a continuous belief space and starts from a fixed belief point. We have earlier discussed [Nair et al., 2003a]'s JESP algorithm that uses dynamic programming to reach a local optimal. [Hansen; and Zilberstein, 2005] present a locally optimal bounded policy iteration algorithm for infinite-horizon distributed algorithms. This algorithm has been theoretically shown to work in a continuous belief space from an unknown initial belief distribution. While this is an important contribution, the use of finite-state controllers restricts the policy representation. Also, their experimental results are for a single initial belief. Further, unlike our algorithm they use a *correlation device* in order to ensure coordination among the various agents.

In other related models to distributed POMDPs, there has been an interesting model called the Interactive POMDP (I-POMDP) model by Piotr *et al* Gmytrasiewicz and Doshi [2005]. This model extends the POMDP model to multi-agent settings by incorporating the notion of agent models into the state space. Agents maintain beliefs over physical states of the environment and over models of other agents, and they use Bayesian update to maintain their beliefs over time. In I-POMDPs, an agent is primarily concerned about its own welfare, while in distributed POMDPs, an agent is concerned about the team welfare.

Chapter 10

Conclusion

This thesis presents techniques to build agents/teams of agents that make sequence of decisions, while operating in real world uncertain environments. A need for such systems has been shown in many facets of human life, such as software personal assistants, therapy planning, space mission planning, sensor webs for monitoring weather phenomena and others. However for such systems to be a reality, these agents need to handle the uncertainty arising at various levels in these domains: unknown initial state, non-deterministic outcome of actions, and noisy observations. While Partially Observable Markov Decision Processes (POMDPs) and Distributed POMDPs provide powerful models to address uncertainties in real-world domains, solving these models is computationally expensive. Due to this significant computational complexity of these models, existing approaches that provide exact solutions do not scale, while approximate solutions do not provide any usable guarantees on quality.

Towards addressing the above challenges, the following key ideas have been proposed in this thesis: (a) Exploiting structure to improve efficiency of POMDPs and Distributed POMDPs. This technique exploits structure in dynamics to solve POMDPs faster, while the second exploits interaction structure of agents to solve distributed POMDPs. (b) An approximate technique for POMDPs and Distributed POMDPs that approximates directly in the value space. This technique provides quality bounds that are easily computable and operationalizable, while providing comparable performance to fastest existing solvers.

For agents and multiagent systems to finally break out in the real-world, in a very fundamental sense, they must conquer uncertainty. In the future, I would like to build upon the work in my thesis towards understanding the reasoning process in ever more realistic environments.

- *Environments with cooperation and competition*: Previous work in distributed POMDPs and multiagent systems in general has categorized agents as either fully adversarial or completely collaborative. However, in many real-world applications, such stark categorization may not be appropriate; agents' motivations thus themselves become sources of uncertainty. Modeling such uncertainties in Distributed Markov Decision Problems and Distributed POMDPs is an open question in the field.

- *Unknown environments*: These are domains where there is no model available or there is uncertainty about the model itself, thus requiring a learning phase to reduce the uncertainty about the model.

- *Bounded resource environments*: These domains are constrained by the limited availability of resources. There is uncertainty introduced in such domains because actions result in non-deterministic consumption of resources. Decision process in such domains becomes complicated due to this underlying uncertainty and the constraints imposed by the resource availability.

I believe that understanding the process of decision making in these critical settings and utilizing this knowledge towards building intelligent agent/multi-agent systems will result in a smooth transition of intelligent systems into our daily life.

Bibliography

R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-Independent Decentralized Markov Decision Processes. In *AAMAS*, 2003.

R. Becker, S. Zilberstein, V. Lesser, and C.V. Goldman. Solving transition independent decentralized Markov decision processes. *JAIR*, 22:423–455, 2004.

D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of MDPs. In *UAI*, 2000.

C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *AAAI*, 1996.

M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *AIJ*, 2002.

D. Braziunas and Craig Boutilier. Stochastic local search for POMDP controllers. In *AAAI*, 2004.

A. R. Cassandra, M. L. Littman, and N. L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *UAI*, 1997a.

A. R. Cassandra, M. L. Littman, and N. L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *UAI*, 1997b.

I. Chadès, B. Scherrer, and F. Charpillet. A heuristic approach for solving decentralized-pomdp: Assessment on the pursuit problem. In *SAC*, 2002.

K. Chintalapudi, E. A. Johnson, and R. Govindan. Structural damage detection using wireless sensor-actuator networks. In *Proceedings of the thirteenth Mediterranean Conference on Control and Automation*, 2005.

Rina Dechter. *Constraint Processing*. Morgan Kaufman, 2003.

D. Dolgov and E. Durfee. Graphical models in local, asymmetric multi-agent markov decision processes. In *AAMAS*, 2004.

Z. Feng and E. Hansen. An approach to state aggregation for pomdps. In *AAAI-04 Workshop on Learning and Planning in Markov Processes – Advances and Challenges*, 2004.

Z. Feng and S. Zilberstein. Efficient maximization in solving POMDPs. In *AAAI*, 2005.

Z. Feng and S. Zilberstein. Region based incremental pruning for POMDPs. In *UAI*, 2004a.

Z. Feng and S. Zilberstein. Region based incremental pruning for POMDPs. In *UAI*, 2004b.

P. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

C.V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *JAIR*, 22:143–174, 2004.

C. Guestrin, S. Venkataraman, and D. Koller. Context specific multiagent coordination and planning with factored MDPs. In *AAAI*, 2002.

D. Bernstein; E. Hansen; and S. Zilberstein. Bounded policy iteration for decentralized pomdps. In *IJCAI*, 2005.

E.A. Hansen, D.S. Bernstein, and S. Zilberstein. Dynamic Programming for Partially Observable Stochastic Games. In *AAAI*, 2004a.

Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, 2004b.

M. Hauskrecht. Value-function approximations for POMDPs. *JAIR*, 13:33–94, 2000a.

M. Hauskrecht. Value-function approximations for POMDPs. *JAIR*, 13:33–94, 2000b.

M. Hauskrecht and H. Fraser. Planning treatment of ischemic heart disease with partially observable markov decision processes. *AI in Medicine*, 18:221–244, 2000.

M. Hauskrecht and H. Fraser. Planning treatment of ischemic heart disease with partially observable markov decision processes. *Artificial Intelligence in Medicine*.

CALO: Cognitive Agent that Learns and Organizes. http://www.ai.sri.com/project/CALO, http://calo.sri.com, 2003.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *AI Journal*, 1998.

H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjoh, and S. Shimada. RoboCup-Rescue: Search and rescue for large scale disasters as a domain for multiagent research. In *IEEE SMC*, 1999.

T. Y. Leong and C. Cao. Modeling medical decisions in DynaMoL: A new general framework of dynamic decision analysis. In *World Congress on Medical Informatics (MEDINFO)*, pages 483–487, 1998.

V. Lesser, C. Ortiz, and M. Tambe. *Distributed sensor nets: A multiagent perspective*. Kluwer, 2003.

P. Magni, R. Bellazzi, and F. Locatelli:. Using uncertainty management techniques in medical therapy planning: A decision-theoretic approach. In *Applications of Uncertainty Formalisms*, pages 38–57, 1998.

R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham. Taking dcop to the real world : Efficient complete solutions for distributed event scheduling. In *AAMAS*, 2004.

R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, 2004a.

R. Mailler and V. Lesser. Using cooperative mediation to solve distributed constraint satisfaction problems. In *AAMAS*, 2004b.

N. Menleau, K. E. Kim, L. P. Kaelbling, and A. R. Cassandra. Solving POMDPs by searching the space of finite policies. In *UAI*, 1999.

P. J. Modi, H. Jung, M. Tambe, W. Shen, and S. Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation [pdf] [ps] (extended version. In *CP*, 2001.

P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *AAMAS*, 2003a.

P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *AAMAS*, 2003b.

R. E. Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *AAMAS*, 2004.

R. Nair, D. Pynadath, M. Yokoo, M. Tambe, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*, 2003a.

R. Nair, M. Tambe, and S. Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS-03)*, pages 552–559, 2003b.

R. Nair, M. Tambe, and S. Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *AAMAS*, 2003c.

R. Nair, M. Roth, M. Yokoo, and M. Tambe. Communication for improving policy computation in distributed pomdps. In *AAMAS*, 2004.

R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *AAAI*, 2005.

S. Paquet, B. Chaib-draa, and Stephane Ross. Rtbss: An online pomdp algorithm for complex environments. In *AAMAS*, 2005.

P. Paruchuri, M. Tambe, R. Ordonez, and S. Kraus. Security in multiagent systems by policy randomization. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS-06)*, 2006.

L. Peshkin, N. Meuleau, K.-E. Kim, and L. Kaelbling. Learning to cooperate via policy search. In *UAI*, 2000a.

L. Peshkin, N. Meuleau, K.-E. Kim, and L. Kaelbling. Learning to cooperate via policy search. In *UAI*, 2000b.

A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, 2005.

J. Pineau and G. Gordon. POMDP planning for robust robot control. In *ISRR*, 2005.

J. Pineau, G. Gordon, and S. Thrun. PBVI: An anytime algorithm for POMDPs. In *IJCAI*, 2003.

M. E. Pollack, L. Brown, D. Colbry, C. E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, and I. Tsamardinos. Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems*, 44:273–282, 2003a.

M. E. Pollack, L. Brown, D. Colbry, C. E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, and I. Tsamardinos. Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems*, 44:273–282, 2003b.

P. Poupart and C. Boutilier. Vdcbpi: an approximate scalable algorithm for large scale POMDPs. In *NIPS*, 2004.

D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *JAIR*, 16:389–423, 2002.

Maayan Roth, Reid G. Simmons, and Manuela M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *AAMAS*, 2005.

N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In *NIPS*, 2002.

M. Paskin S. Funiak, C. Guestrin and R. Sukthankar. Distributed localization of networked cameras. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN-06)*, 2006.

P. Scerri, D. Pynadath, and M. Tambe. Towards adjustable autonomy for the real-world. *JAIR*, 17:171–228, 2002.

D. Schreckenghost, C. Martin, P. Bonasso, D. Kortenkamp, T.Milam, and C.Thronesbery. Supporting group interaction among humans and autonomous agents. In *AAAI*, 2002.

R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

T. Smith and R. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *UAI*, 2005.

D. Szer, F. Charpillet, and S. Zilberstein. Maa*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-05)*, 2005.

P. Varakantham, R. Maheswaran, and M. Tambe. Exploiting belief bounds: Practical pomdps for personal assistant agents. In *AAMAS*, 2005.

M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *ICMAS*, 1996.

R. Zhou and E. Hansen. An improved grid-based approximation algorithm for POMDPs. In *IJCAI*, 2001.

PRADEEP R. VARAKANTHAM

CURRENT POSITION

Graduate Research Assistant
Computer Science Department
Viterbi School of Engineering
University of Southern California
teamcore.usc.edu/pradeep
varakant@usc.edu

USC Powell Hall 516
3737 Watt Way
Los Angeles, CA 90089
Phone: (213)740-9569
Fax: (213)740-7877

RESEARCH INTERESTS

Decision making in uncertain domains; Multiagent systems; Probabilistic reasoning; Reinforcement learning; Distributed constraint reasoning.

EDUCATION

Doctor of Philosophy (in progress)
Computer Science 09/03 - Present
University of Southern California, USA
Thesis: Towards efficient planning for real world partially observable domains
Advisor: Prof. Milind Tambe
Committee: Prof. Milind Tambe, Prof. Manuela Veloso, Prof. Sven Koenig,
Prof. Makoto Yokoo, Prof. Stacy Marsella, Prof. Fernando Ordonez

Master of Science
Computer Science 09/03 - 05/06
University of Southern California, USA

Bachelor of Technology
Computer Science and Information Technology 09/98 - 06/02
International Institute of Information Technology, India
Thesis: XML Storage and Concurrency Manager
Advisor: Prof. Kamalakar Karlapalem

HONORS AND ACTIVITIES

Outstanding Research Assistant in the CS department at USC for the year 2005: There are only two outstanding research assistant awards given out each year in the department for excellence in research.

Awarded **dean's merit scholarship** at IIIT for two semesters: Given to students with a GPA above 9.5(out of 10) in a semester.

Awarded **merit certificate** in state wide Maths Olympiad: One of 60 (out of 10000) high school students who received this merit certificate.

EXPERIENCE

Graduate Research Assistant
Teamcore Research Group (Prof. Milind Tambe) 09/03 - Present
Computer Science Department, USC
Los Angeles, CA

***Theory**: Working on the problems of decision making in partially observable domains, primarily using the POMDP (Partially Observable Markov Decision Problem) and Distributed POMDP models for representing the domains. The key contributions have been in providing efficient techniques to solve these models while providing bounds on the solution quality.*

***Applied**: Working on developing a system that assists users in managing tasks on a project. This system is being developed in the context of CALO (Cognitive Agent that Learns and Organises) and employs POMDPs for making decisions about allocation/reallocation of tasks and adjustable autonomy.*

Teaching Assistant for Advanced Artificial Intelligence
Computer Science Department, USC 08/05 - 12/05
Los Angeles, CA

Was involved with teaching classes, creating and evaluating exams, and addressing students' concerns.

Software Developer
Divine Inc. 07/02 - 07/03
Hyderabad, India

Worked on an Instant Messaging tool called MindAlign. Was one of the few developers who was on the maintenance team for this product with 100 developers.

Summer Intern
Language Technologies Research Center 04/00 - 07/00
IIIT, Hyderabad, India

Developed a natural language querying interface for a relational database system. The interface was developed for two languages English and Hindi.

PUBLICATIONS

Journal Publications

Pradeep Varakantham, Ranjit Nair, Yoonheui Kim, Milind Tambe and Makoto Yokoo. *Exploiting Locality of Interaction in Networked Distributed POMDPs*. **Submitted** to Journal of Artificial Intelligence Research, JAIR.

Rajiv Maheswaran, Jonathan Pearce, Pradeep Varakantham, Emma Bowring and Milind Tambe, *Privacy Loss in Distributed Constraint Reasoning: A Quantitative Framework for Analysis and its Applications*. Journal of the Autonomous Agents and MultiAgent systems (JAAMAS),13:27−60, 2006.

Conference and Poster Publications

Pradeep Varakantham, Janusz Marecki, Makoto Yokoo and Milind Tambe. *Letting loose a SPIDER on a network of POMDPs: Generating quality guaranteed policies*. Accepted as a full paper for presentation at Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS-2007. **Acceptance Rate: 22%**.

Pradeep Varakantham, Rajiv Maheswaran, Tapana Gupta and Milind Tambe. *Towards efficient computation of quality bounded solutions in POMDPs*. Accepted for ORAL PRESENTATION at the Twentieth International Joint Conference on Artificial Intelligence, IJCAI-2007. **Acceptance Rate: 15%**.

Pradeep Varakantham, Ranjit Nair, Milind Tambe and Makoto Yokoo. *Winning back the CUP for Distributed POMDPs: Planning over continuous belief spaces*. Proceedings of the fifth International Conference on Autonomous Agents and Multi Agent Systems, AAMAS-2006. **Acceptance Rate: 23%**.

Pradeep Varakantham, Rajiv Maheswaran, and Milind Tambe. *Exploiting Belief Bounds: Practical POMDPs for Personal Assistant Agents*. Proceedings of the fourth International Conference on Autonomous Agents and Multi Agent Systems, AAMAS-2005. **Acceptance Rate: 24%**.

Ranjit Nair, Pradeep Varakantham, Milind Tambe and Makoto Yokoo. *Network Distributed POMDPs, A Synthesis of Distributed Constraint Optimization and POMDPs*. Proceedings of the Twentieth National Conference on Artificial Intelligence, AAAI-2005. **Acceptance Rate: 18%**.

Ranjit Nair, Pradeep Varakantham, Makoto Yokoo and Milind Tambe. *Networked Distributed POMDPs: A Synergy of Distributed Constraint Optimization and POMDPs*. Poster paper at the proceedings of the International Joint Conference on Artificial Intelligence,IJCAI-2005. **Acceptance Rate: 22%**.

Rajiv Maheswaran, Jonathan Pearce, Pradeep Varakantham, Emma Bowring, and Milind Tambe. *Valuations of Possible States (VPS): A Unifying Quantitative Framework for Evaluating Privacy in Collaboration*. Proceedings of the fourth International Conference on Autonomous Agents and Multi Agent Systems, AAMAS-2005. **Acceptance Rate: 24%**.

Rajiv Maheswaran, Milind Tambe, Emma Bowring, Jonathan Pearce, Pradeep Varakantham. *Taking DCOP to the Real World : Efficient Complete Solutions for Distributed Event Scheduling*. Proceedings of the third International Conference on Autonomous Agents and Multi Agent Systems, AAMAS-2004. **Acceptance Rate: 24%**.

Invited papers

Milind Tambe, Emma Bowring, Pradeep Varakantham, Jonathan Pearce, David Pynadath, Paul Scerri. *Electric Elves: What went wrong and why*. In special issue of AI Magazine on What went wrong and why (to appear).

Praveen Paruchuri, Emma Bowring, Ranjit Nair, Jonathan Pearce, Nathan Schurr, Milind Tambe, Pradeep Varakantham. *Hybrids in Multiagent Teamwork*. Published in Communications of the Computer Society of India.

M. Tambe, E. Bowring, H. Jung, G. Kaminka, R. Maheswaran, J. Marecki, P. J. Modi, R. Nair, J. Pearce, P. Paruchuri, D. Pynadath, P. Scerri, N. Schurr, P. Varakantham. *Conflicts in teamwork: Hybrids to the rescue*. Invited paper at the Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS-2005.

Book Chapters

Pradeep Varakantham, Rajiv Maheswaran, Milind Tambe. *Implementation Techniques for solving POMDPs in Personal Assistant Domains*. Programming Multiagent Systems (PROMAS). Springer Press Book Chapter, 2006.

Rajiv Maheswaran, Milind Tambe, Pradeep Varakantham and Karen Myers. *Adjustable Autonomy challenges in in Personal Assistant Agents: A Position Paper*. Proceedings of Autonomy'03.

Symposium Publications

Yoonheui Kim, Ranjit Nair, Pradeep Varakantham, Milind Tambe and Makoto Yokoo. *Exploiting locality of interaction in Network Distributed POMDPs* . Proceedings of the AAAI Spring Symposium on Distributed Plan and Schedule Management, 2006.

Milind Tambe, Emma Bowring, Jonathan Pearce, Pradeep Varakantham, Paul Scerri, David Pynadath. *Electric Elves: What Went Wrong and Why*. Proceedings of the AAAI Spring Symposium on What Went Wrong and Why, 2006.

Pradeep Varakantham, Rajiv Maheswaran and Milind Tambe. *Practical POMDPs for Personal Assistant Agents*. Proceedings of the AAAI Spring Symposium on Persistent Assistants, 2005.

Rajiv Maheswaran, Jonathan Pearce, Pradeep Varakantham, Emma Bowring and Milind. Tambe. *Valuations of Possible Worlds (VPW): A Quantitative Framework for Analysis of Privacy Loss Among Collaborative Personal Assistant Agents*. Proceedings of the AAAI Spring Symposium on Persistent Assistants, 2005.

Workshop papers

Nathan Schurr, Pradeep Varakantham, Emma Bowring, Milind Tambe, and Barbara Grosz. *Asimovian Multiagents: Applying Laws of Robotics to Teams of Humans and Agents*. Proceedings of the Programming Multi-agent systems (PROMAS) workshop at AAMAS-2006.

Pradeep Varakantham, Rajiv Maheswaran, and Milind Tambe. *Implementation techniques for solving POMDPs in personal assistant domains.* Proceedings of the Programming Multi-agent systems (PROMAS) workshop at AAMAS-2005.

Ranjit Nair, Pradeep Varakantham, Makoto Yokoo and Milind Tambe. *Exploiting Interaction Structure in Networked Distributed POMDPs.* Proceedings of the Planning and Scheduling workshop at ICAPS, 2005.

Pradeep Varakantham, Rajiv Maheswaran and Milind Tambe. *Agent Modeling in Partially Observable Domains.* Proceedings of the AAMAS Workshop on Agent Modeling using Observations, 2004.

PROFESSIONAL ACTIVITIES

Program Committee

National Conference on Artificial Intelligence, AAAI (2007).

Reviewer

Journal of Artificial Intelligence Research (JAIR).
Journal of Autonomous Agents and Multi Agent systems (JAAMAS).
International Joint Conference on Artificial Intelligence, IJCAI (2005, 2007).
Autonomous Agents and Multi Agent Systems, AAMAS (2006).
IEEE Transactions on Systems, Man, and Cybernetics Part B (2005).
Brazilian Symposium on Artificial Intelligence (2004).
Physics of Life Reviews (2004).

Miscellaneous

Maintainer of the distributed POMDP respository ($teamcore.usc.edu/pradeep/dpomdp\_page.html$).

REFERENCES

Milind Tambe
Professor
Computer Science Department
University of Southern California
Powell Hall of Engineering 208
3737 Watt Way
Los Angeles, CA 90089-0781
Phone: (213) 740-6447
Fax: (213) 740-7285
Email: tambe@usc.edu
Web: http://teamcore.usc.edu/tambe

Manuela Veloso
Professor of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
Phone: (412) 268-1474
Fax: (412) 268-4801
Email: veloso@cs.cmu.edu
Web: http://www.cs.cmu.edu/ mmv

Sven Koenig
Associate Professor
Computer Science Department
University of Southern California
300 Henry Salvatori Computer Science Center (SAL)
941 W 37th Street
Los Angeles, CA 90089-0781
Phone: 213-740-6491
Fax: 213-740-7285
Email: skoenig@usc.edu
Web:idm-lab.org/

Makoto Yokoo
Professor
Department of Intelligent Systems
Kyushu University
6-10-1 Hakozaki, Higashi-ku
Fukuoka, 812-8581, Japan
Phone:+81-92-642-4065
Fax:+81-92-632-5204
Email: yokoo@is.kyushu-u.ac.jp
Web: lang.is.kyushu-u.ac.jp/yokoo/