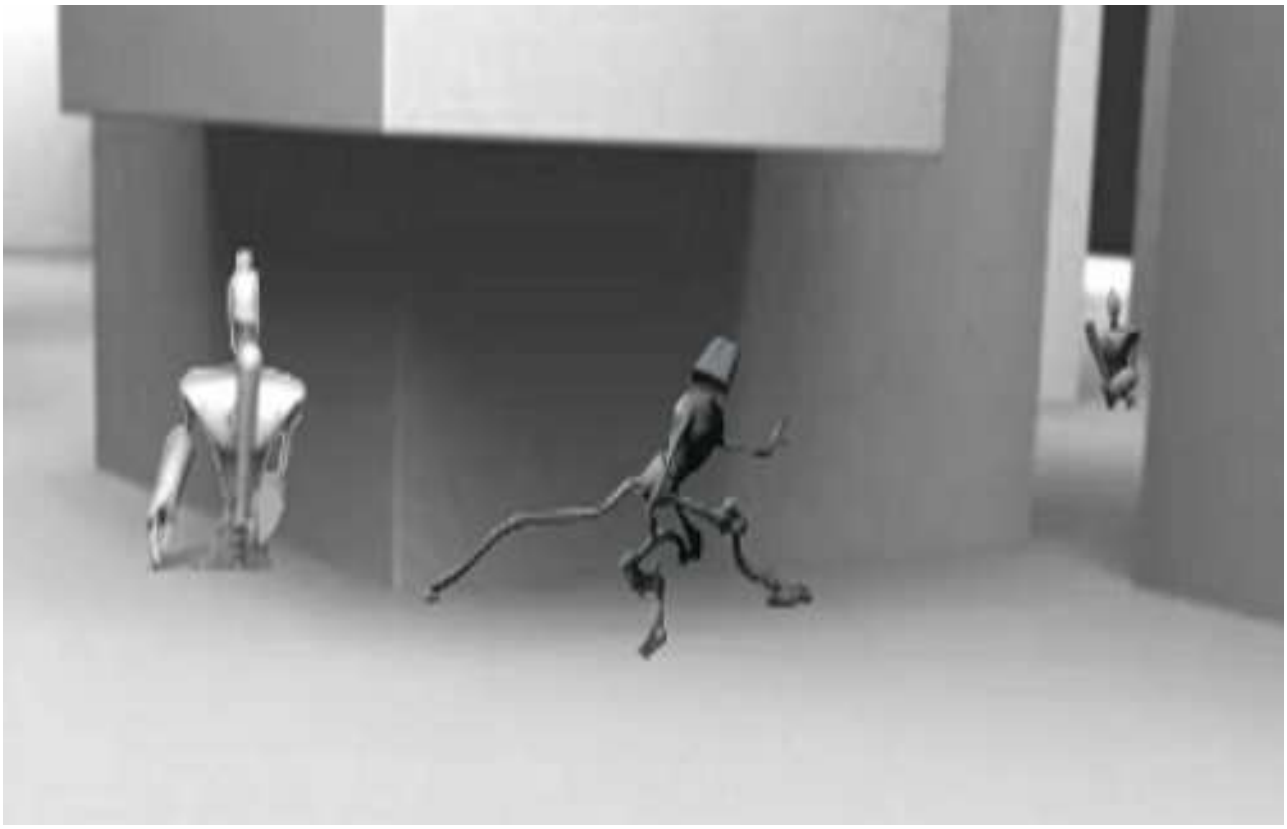
Lossless clustering of multi-agent beliefs to approximate large horizon Dec-POMDPs

Štefan Konečný



University of Amsterdam

Lossless clustering of multi-agent beliefs to approximate large horizon Dec-POMDPs

Štefan Konečný
5765277

Master thesis
Credits: 42 EC

Master Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. Arnoud Visser

Intelligent Systems Lab Amsterdam
Informatics Institute
Faculty of Science
University of Amsterdam
Kruislaan 403
1098 SJ Amsterdam

October 27th, 2011

Abstract

Research of Multi Agent Systems (MAS) is devoted to modeling, analyzing and designing interaction between intelligent autonomous agents operating in an uncertain environment. Usage of MAS in logistics, resource allocation and electronic commerce has been suggested, where software agents interact with humans or act on their behalf. Additionally MAS can be used for solving complex tasks, which can be decomposed into simpler subtasks, each carried out by a single agent. The *Dec-POMDP* formalism is often used to represent this interaction of multiple cooperative agents in an uncertain environment.

This thesis contributes to *Dec-POMDP* research in multiple ways. Many offline *Dynamic Programming* (*DP*, *BDP*, *MBDP*) and online *Bayesian Game Approximation* (*Forward-Sweep Policy Computation*) algorithms have been reimplemented into the *MADP Toolbox*. This increases the reusability of these algorithms among fellow researchers.

Additionally a new *BAGA* variant has been proposed. This algorithm introduces the known *Lossless Clustering* into the *BAGA* online algorithm. For many considered domains it is the best performing *BAGA* variant and obtains solutions close to optimum.

Many algorithms have been evaluated against new benchmarks and thus many new performance values have been obtained. Also the known *Robotic Tag 2* domain has been added into *MADP Toolbox*, which makes it largest problem domain in the toolbox.

The analysis has shown that while the scaling of considered *DP* algorithms is limited, *FSPC* methods scale much better and obtain results close to optimum. They are also faster than considered offline algorithms.

Acknowledgement

First of all, I would like to express my gratitude towards my supervisor Arnoud Visser, who provided me with an exciting thesis topic and had a lot of patience with me while pursuing this research. I would also like to express my thankfulness towards Matthijs Spaan and Frans Oliehoek for creating the MADP Toolbox which I could extend, and more importantly for their readily support and helpfulness whenever I ran into trouble. I also feel indebted to Christoper Amato for providing the source code of Dynamic Programming methods and youtube user simplex409 whose video is the source of the cover page picture.

I also feel grateful to my family, which have supported me in many ways, especially my mother and my sister whom I have robbed of the privilege of having her own room.

And thanks to You God, for all the inspiration and blessings.

List of Algorithms

List of Tables

List of Figures

Contents

Chapter 1

Introduction

In recent years, many new application domains for intelligent autonomous software entities (or simply agents) have been proposed. While the usage of intelligent agents to govern robots in environments hazardous [Kitano et al. (1999)] or hardly accessible [Zilberstein et al. (2002)] to humans has been suggested before, nowadays they are increasingly employed in domains which are too complex or change too rapidly for efficient interaction with a human. Moreover, in many real world settings such as logistics [van der Putten et al. (2006)], resource allocation [Giesen et al. (2009)], smart grid [Block et al. (2009)], electronic commerce [Arunachalam and Sadeh (2005)] these agents interact with other intelligent agents (some of them may be human) which increases the complexity and dynamics of such *Multi Agent Systems* even more. Because of the complexity and the rate of change, it can be quite difficult for a human agent to interact with such a system. Humans may instead delegate a software agent to act on their behalf.

To achieve a complex task may be still too difficult for a single software agent. Such a task can be split into a set of simpler tasks, each of which is performed by a single agent. These agents then have to coordinate their behavior, while often having only a limited knowledge about the state of the environment or the behavior of the other agents. This uncertainty makes the efficient coordination challenging and an area of thriving research. This thesis also focuses on the cooperative setting, where multiple agents have to coordinate their actions to reach a common goal in an uncertain and dynamic environment. Thorough

this thesis the the *Dec-POMDP* [Bernstein et al. (2002)] model is used to formally represent such a setting.

Dec-POMDP

In *Dec-POMDPs* multiple autonomous intelligent agents work towards a common goal. In each of the discrete timesteps, each agent independently selects its individual action. The actions of *all* agents(the *joint action*) *together* with the problem dynamics may alter the state of the environment. Each agent *i* receives an individual observation about this altered state. These observations may be used to infer the effect of the action of agent *i*. Both actions and observations of the other agents are unknown to *i*. All agents also receive the same reward, which depends on the previous state of the environment and *the joint action* they have performed.

While the setting is a purely cooperative one, the agents have to deal with different forms of uncertainty. First of all they have a limited knowledge about the state of the environment, and the dynamics governing the change of the environment can be stochastic. Secondly their sensors are often unreliable (noisy observations) and so are their actuators (failed actions). Lastly they do not know what were the actions and observations of the other agents, and so they have to reason about the possible behavior of the other agents as well.

In the *Dec-POMDP* model the possibility of communication is not considered, although it could be integrated seamlessly. Received messages could be added into the set of observations and transmissions could be handled as additional actions [Emery-Montemerlo (2005)]. The introduction of communication into a noisy domain would further increase the complexity of the problem. Phenomena like (unknown) communication delays and errant messages would have to be considered in most real world domains. This would lead agents to develop some kind of communication policy, and they would also have to anticipate the communication policies of the other agents.

The sequence of joint (individual) actions and observations that occurred during one trial is called the *joint action-observation history* (*JAOH*) (*individual action-observation history* (*IAOH*)). Each agent has to select its actions based on its *IAOH* and the known

problem dynamics only. The solving the *Dec-POMDP* amounts to assigning a policy to each agent (which describes how to act in each situation), or provide it with an algorithm to construct such a policy at runtime. This set of individual policies (forming a joint policy) should attain the maximum average reward in a finite number of time steps (called the horizon, h) for all possible *JAOHs*.

The complexity of finding an optimal solution to a *Dec-POMDP* is discussed in the next section.

Complexity Results

Solving finite horizon *Dec-POMDP* is very hard, in fact it has been proven to be NEXP complete [Bernstein et al. (2002)] and to find ε optimal (bounded approximate solution) history dependent joint policies is still NEXP hard [Rabinovich et al. (2003)]. [Seuken and Zilberstein (2008)] offers a nice overview of models related to *Dec-POMDPs*, complexity results and solution methods (prior to 2008). Despite of the recent advancement in optimal solution methods [Spaan et al. (2011)], these worst-case complexity results motivate research in approximate solution methods.

Approximate Solution Methods

As the complexity of trivial enumeration (and subsequent evaluation) of all possible *policies* is double exponential[1], many algorithms attempt to prune (*Dynamic Programming* ([Hansen (2004)], see Section 2.2.1) and *Bounded Dynamic Programming* ([Amato et al. (2007b)], see Section 2.2.1)), efficiently explore the *joint policy* space (*GMAA** variants [Oliehoek et al. (2008a, 2009)], see Section 2.3.5), optimize their performance on most likely *JAOHs* (*Memory-Bounded Dynamic Programming* [Seuken and Zilberstein (2007b)], see Section 2.2.2) or combine the aforementioned [Carlin and Zilberstein (2008); Amato et al. (2009); Wu et al. (2010)].

[1] $\mathcal{O}(|\mathcal{A}|^{(|\mathcal{O}|)^h})$ where $\mathcal{A}$ is the set of possible joint actions, $\mathcal{O}$ is the set of possible joint observations and h is the considered horizon

Others opted for approximation of the horizon *h Dec-POMDPs* through a series of *Bayesian Games*, each solving a single timestep, referred to as *Forward-Sweep Policy Computation methods* in this work. This approach was first suggested in [Emery-Montemerlo et al. (2004, 2005)] and the idea of *Bayesian Game Approximation* technique is an integral part of some of the best performing exact solution methods [Oliehoek et al. (2009); Spaan et al. (2011)] to date.

In this thesis many aforementioned algorithms have been reimplemented, to provide their comprehensive comparison across a set of standard benchmark problems. Both online and offline planning algorithms have been reimplemented, which allows for a comparison of the two approaches to *Multi Agent Planning*.

Online vs offline planning

Dec-POMDP planning methods can be divided into two broad categories of online and offline planning. This two methodologies differ substantially in the way how they cope with uncertainty and how they use feedback from the environment.

Offline methods keep the planning and execution phase strictly separate. In the planning phase, the best possible *joint policy* π is found, and each agent i is assigned its part π_i (its *individual policy*) to follow during the execution phase. This *joint policy* is designed to maximize the average result achieved on all *JAOHs*, which can occur during following of policy π. During the execution phase, the behavior of agent is not altered based on his *IAOH*. In fact, the *IAOH* is used merely to identify which action it should perform in accordance with π_i. The lack of adaptability is clearly a drawback of this approach. On the other hand, it offers possibility of tighter coordination, as the algorithm has access to both *joint actions* and *joint observations* during the planning phase. It is also likely to have more stable performance, as the π does well (on average) for all possible *JAOHs.* The planning phase may take a long time, but it is done only once and the execution phase is very fast. Most of *Dec-POMDP* solving methods available to date perform offline planning.

Online planning methods attempt to utilize the received runtime data to create and update a belief about the true state of the environment. They typically interleave planning and

execution phase. Therefore the planning phase has to be performed quickly enough to enable for a smooth execution. Each agent i uses its *IAOH* to infer a belief about the true state of the environment, and this belief is updated upon each *individual action* performed and *individual observation* received. As the environment changes depending on the *joint action* a, each agent has to deduce the *individual action* (and the resulting *individual observation*) of all the other agents, relying on a common knowledge assumption (as for instance the distribution over possible *JAOHs* described in Section 2.3). Although the online planning algorithms employ coordination mechanisms, the beliefs of the agents may be different nonetheless, which poses a threat to an effective coordination.

It is difficult to make a compelling case for one approach or the other. In domains I have experimented with they have achieved similar results. In general the *BAGA* [Emery-Montemerlo et al. (2005)] reimplemented online algorithms tend to be much faster than any offline algorithm I am aware of. Online algorithms should also scale better with growing horizon, although the best scaling algorithms known to date are offline.

Contribution of this Thesis

The contribution of this thesis to the *Dec-POMDP* research is many fold.

First of all many *Dynamic Programming* and *Forward-Sweep Policy Computation* algorithms have been reimplemented and added into a software library aimed to support Multi Agent Decision Process research (MADP Toolbox [Spaan and Oliehoek (2008)]). This enables their direct comparison and allows for their easier modification for future researchers.

All experiments have been carried out on a set of benchmark domains, many of which were not in use in the time of publication of the original algorithms. Thus Chapter 3 and Appendix C present many new results for the traditional algorithms, achieved in the new domains. Additionally a larger domain *Robotic Tag 2* [Emery-Montemerlo et al. (2004)] has been added to the MADP Toolbox, which can be solved only by *FSPC* for larger horizons.

A new *BAGA* variant *BAGA-Lossless* has been proposed, which introduces the benefit of Lossless Clustering [Oliehoek et al. (2009)] into the *BAGA* algorithm. This new *BAGA*

variant performs very well (close to optimum in most domains) and it outperforms the previous BAGA versions [Emery-Montemerlo et al. (2005)] for many domains.

Chapter 2

Theory

This chapter provides the theory background and introduces the notation used thorough this thesis. First the *Dec-POMDP* model (Section 2.1) is introduced. Readers familiar with the model and the notation used in [Spaan and Oliehoek (2008); Spaan et al. (2011)] can safely skip this section. The notation for a particular policy representation used in the reimplemented *Dynamic Programming* methods (*policy tree*) is presented in Section 2.1.1. Then the *Dynamic Programming* methods are discussed in Section 2.2. The last Section 2.3 is devoted to methods using *Bayesian Game Approximation* and also introduces the necessary *Bayesian Game* formalism. While the main focus of this section is on the *Forward-Sweep Policy Computation* (variants of the *BAGA* [Emery-Montemerlo et al. (2005)] algorithm), the GMAA*Cluster [Oliehoek et al. (2009)] is also presented in some detail.

2.1 Dec-POMDP framework

All work in this thesis is carried out in the scope of the *Dec-POMDP* framework. *Dec-POMDP (Decentralized Partially Observable Markov Decision Process)* is an extension of *POMDP* into the multiple agents case. The *Dec-POMDP* model is very powerful and enables us to model the interaction of two (or more) autonomous agents, which take individual actions based on their individual observations and *common knowledge* about the domain. These individual observations may not represent the world accurately (a.k.a they

may be noisy) and the carried out actions may fail to have the intended effect. The *Dec-POMDP* model is a purely cooperative setting, and both agents work towards a common goal.

Formally a *Dec-POMDP* is defined as a tuple $\langle \mathscr{D}, S, \mathscr{A}, T, \mathscr{O}, O, R, h, b_0 \rangle$ (the adopted notation is from Spaan and Oliehoek (2008); Spaan et al. (2011)) where

- $\mathscr{D}$ is a finite set of n agents
- S is a finite set of possible states of the environment
- $\mathscr{A} = \times_i \mathscr{A}_i$ is a finite set of possible *joint actions*. $\mathscr{A}_i$ is the finite set of individual actions for agent i
- T is the transition function, a mapping from states and joint actions to probability distributions over next states $T : S \times \mathscr{A} \mapsto \mathscr{P}(S)$
- $\mathscr{O} = \times_i \mathscr{O}_i$ is a finite set of possible joint observations. $\mathscr{O}_i$ is the finite set of individual observations for agent i
- O is the observation function, a mapping from joint actions and successor states to probability distributions over *joint observations* $O : \mathscr{A} \times S \mapsto \mathscr{P}(\mathscr{O})$
- R is the reward function, a mapping from *joint actions* and current state to real number $R : S \times \mathscr{A} \mapsto \mathbb{R}$
- h is the horizon, the number of timesteps considered.
- b_0 is an initial belief (probability distribution over states) $b_0 \in \mathscr{P}(S)$

For the sake of notational simplicity we shall focus on the special case of having only two agents (thus we assume $n = 2$ for the rest of this thesis), however the reimplemented algorithms will work for any (small) number of agents without any modification. The time is divided into discrete timesteps, represented by a superscript t in subsequent notations.

At the beginning ($t = 0$), the environment is at the state s^0. While s^0 is unknown to both agents, it is known to be drawn from b_0, which is *common knowledge*. At each time step t ($t \geq 0$) each agent i performs individually an action a_i^t which is unknown to the other agent.

Given the current state s^t and the *joint action* $a^t = \langle a_0^t, a_1^t \rangle$ the environment changes its state to a new state s^{t+1}, which is drawn from the distribution $T(s^t, a^t) \in \mathscr{P}(S)$. s^{t+1}again remains hidden to both agents, but a *joint observation* $o^{t+1} = \langle o_0^{t+1}, o_1^{t+1} \rangle$[1] is drawn from the distribution $O(a^t, s^{t+1}) \in \mathscr{P}(\mathscr{O})$. Each agent i receives its individual observation o_i^{t+1}, which is a part of o^{t+1}, and is unknown to the other agent. Both agents receive the same reward $r^t = R(s_t, a_t)$[2]. At the next timestep each agent i performs its individual action a_i^{t+1} and the whole process is repeated until timestep $t = h$ is reached. At this stage, the sum

$$r = \sum_{t=0}^{t<h} r^t \tag{2.1}$$

is calculated which represents the total reward obtained in the run of the problem, in which the sequence $(s^0, a^0, s^1, o^1, a^1, s^2, ..., s^{h-1}, o^{h-1})$ has occurred. The states are never revealed to the individual agents, unless the special case of full observability occurs, where $o_i^t = s^t$ is drawn from $O(a^{t-1}, s^t)$ with probability of 1 for $\forall s^t$.

The sequence $\overrightarrow{\theta^t} = (a^0, o^1, a^1, ..., a^{t-1}, o^t)$ denotes the *joint action-observation history* (*JAOH*). This information is not available during runtime (as agents do not know each others individual actions and observations) and each agent i has access only to its own *individual action-observation history (IAOH)* $\overrightarrow{\theta_i^t} = (a_i^0, o_i^1, a_i^1, ..., a_i^{t-1}, o_i^t)$. Set of all possible *JAOHs* of length t is denoted by Θ^t (Θ_i^t for *IAOHs* respectively). In the runtime each agent i at each timestep t has to select an action a_i^t based on his *IAOH* $\overrightarrow{\theta_i^t}$ and this selection is formally represented by a *decision rule* $\delta_i^t : \Theta_i^t \to \mathscr{A}_i$. A sequence of decision rules for each timestep $\varphi_i^t = (\delta_i^0, ..., \delta_i^{t-1})$ is called an *individual partial policy* (if $t = h$ it is a (full) *individual policy* $\pi_i^h = (\delta_i^0, ..., \delta_i^{h-1})$). Alternatively, *individual policy* can be understood as a mapping *from IAOHs* of length at most t to *individual actions* $\pi_i^t : (\bigcup_{0 \le j < t} \Theta_i^j) \to \mathscr{A}_i$. A tuple of individual policies (one for each agent) is called a *joint policy* $\pi = \langle \pi_0, \pi_1 \rangle$. For cases where $t = h$ we omit the superscript, thus $\pi = \pi^h$. If this mapping (and as a consequence each δ_i^t) is deterministic, it is sufficient to condition the actions on *individual observation histories*

[1] for $t = 0$, o^0 is an empty observation and is therefore omitted in the following notations of histories

[2] There is a further generalization of the *Dec-POMDPs* called *POSG*s, in which each agents maintains its own reward function R_i. Consequently, *Dec-POMDPs* may be referred to as *POSGs* with common payoff in some works.

(*IOHs*) $\overrightarrow{o_i^t} = (o_i^1, o_i^2, ..., o_i^{t-1}, o_i^t)$ instead of the *IAOHs* $\overrightarrow{\theta_i^t}$ (as actions can be inferred from past actions and observations).

The goal of the *Dec-POMDP* planning algorithms is to assign to each agent i an individual policy π_i (or provide it with a method to construct π_i in run time), so that the joint policy $\pi = \langle \pi_0, \pi_1 \rangle$ maximizes the sum (2.1) on average for each possible sequence $(s^0, a^0, s^1, o^1, a^1, s^2, ..., s^{h-1}, o^{h-1})$ induced by the following of π.

2.1.1 Policy Trees

In this thesis *policy trees* are used to represent *policies*. In fact many solution methods [Hansen (2004); Seuken and Zilberstein (2007b); Spaan et al. (2011)] require this particular representation to operate on. Every *joint partial policy* $\varphi^t = \langle \varphi_0^t, \varphi_1^t \rangle$ can be represented by a *joint policy tree* $q^t = \langle q_0^t, q_1^t \rangle$ where each *individual policy tree* $q_i^t \in Q_i^t$ corresponds directly to its *individual policy* counterpart φ_i^t. Q_i^t represents the set of all possible *individual policy trees* for agent i, Q^t denotes the set of all *joint policy trees.* We shall focus only on *joint policy trees* in the upcoming description, but the derivation of the individual agent case is straightforward.

In a join policy tree q^t each node is labeled by a *joint action* and each arch is labeled by a *joint observation*. The root node represents the *joint action* which should be performed on receiving the empty observation o^0. Each non-root node is uniquely identified by a path from the root to this node, which corresponds to a *JAOH* $\overrightarrow{\theta^t}$. The joint action to be performed at this node is identified by $\varphi^t(\overrightarrow{\theta^t})$. Each level of the tree represents a timestep, the root the first (0th) and leaf node the last (t-1 th). A subtree rooted in the t_1 stage having all of its leaves in stage t_2 is denoted by $q^{(t_1,t_2)}$.

As a *policy tree* is only a particular representation of a *policy*, the two terms shall be used interchangeably from now on.

2.2 Dynamic Programming Methods

The *Dynamic Programming Methods* perform offline planning to construct the joint policy tree q^h from bottom to top. In fact the whole process operates on the *individual policy trees*

q_i^t and *joint policies* are used only when they are evaluated (see Equations 2.2-2.4). First subtrees of type $q_i^{(h-1,h-1)}$ are constructed, then trees of type $q_i^{(h-2,h-1)}$ and the complete tree $q_i^h = q_i^{(0,h-1)}$ is constructed only in the final step.

Not all possible subtrees have to be considered. The various Dynamic Programming methods differ mainly in the way how the candidate set C_i^t of subpolicies of type $q_i^{(t,h-1)}$ is selected. While *Hansen Dynamic Programming* and *Bounded Dynamic Programming* prune away policies which are suboptimal, *Memory Bound Dynamic Programming* attempts to find policies which perform well for *JAOHs* likely to occur during runtime.

When candidate policy $q_i^{(t-1,h-1)}$ is constructed, each possible *individual action* for the root node is considered. For each possible *individual observation* a subtree from C_i^t (constructed in previous timestep) is assigned. This subpolicy corresponds to an subpolicy to be followed upon observing the given *individual observation*. From this construction procedure follows that any reduction of C_i^t propagates into the reduction of $C_i^{t_1}$ of all $t_1 < t$. All possible combinations of root action and subpolicies for each individual observation (selected from C_i^t) form an initial candidate pool for C_i^{t-1}. Therefore this procedure of generation of the candidate pool C_i^{t-1} is called *exhaustive backup*. This pool is further reduced, depending on the *DP* method. The various *DP* methods shall be discussed in the upcoming sections.

2.2.1 Hansen Dynamic Programming

Dynamic Programming introduced by [Hansen (2004)] was one of the first non trivial solution methods to solve *Dec-POMDPs*. To distinguish this particular algorithm from the general *Dynamic Programming* solution method described earlier, we shall refer to the prior by *Hansen DP*. Outside the scope of this chapter any mention of *DP* concerns *Hansen DP.* More details on *Hansen DP* can be found in Appendix A.

In *Hansen DP,* finding an optimal solution of the *Dec-POMDP* amounts to finding an optimal *individual policy tree* q_i for each agent i. The *resulting joint policy* tree $q = \langle q_0, q_1 \rangle$ is evaluated according to equations:

$$V^h(s, q, \vec{\theta}) = R(s, a) \tag{2.2}$$

$$V^t(s,q,\vec{\theta}) = R(s,a) + \sum_{s' \in S} \left[\mathscr{P}(s'|s,a) \left(\sum_{o \in \mathscr{O}} \mathscr{P}(o|a,s') V^{t+1}(s', q, \theta^{\vec{a},o}) \right) \right] \tag{2.3}$$

$$V(q) = \sum_{s \in S} b_0(s) V^0(s,q,\vec{\theta}) \tag{2.4}$$

Equation 2.2 forms the base of the recursion and describes how the final *joint action a* of *joint policy tree q* ($a = q(\vec{\theta})$) taken after *JAOH* $\vec{\theta}$ has occurred and the environment is in state s should be evaluated. Equation 2.3 represents the recursive step and aggregates the immediate reward $R(s,a)$ with the expected future reward. In the future reward term, the value for each possible future state s' and future *JAOH* $\theta^{\vec{a},o}$ ($\theta^{\vec{a},o}$ stands for *JAOH* $\vec{\theta}$ extended by a and o) is weighted by the probability of occurrence of s' and $\theta^{\vec{a},o}$ given the action a. Action a is uniquely determined by q and past *JAOH* $\vec{\theta}$ as $a = q(\vec{\theta})$. The probability functions in (2.3) are derived from T and O from the *Dec-POMDP* model. Finally (2.4) explains how the value of the whole *joint policy tree q* can be calculated recursively. As $\vec{\theta}$ consist of an empty observation for $t = 0$ it can be omitted and $V^0(s,q,\vec{\theta})$ can be written as

$$V^0(s,q,\vec{\theta}) = V^0(s, \langle q_0, q_1 \rangle) \tag{2.5}$$

The algorithm attempts to restrict the number *individual policy trees* which have to be considered. If an agent i has two candidate individual policy trees q_i, q_i^* and q_i^* does never worse than q_i (no matter what policy the other agent adopts and what is the belief about the state), q_i can be safely ignored, as q_i^* is as good as q_i and possibly even better for some particular belief over the state and other agents policy combination (the pair of state and individual policy of the other agent is also called *generalized state*, a probability distribution over these states is a *generalized belief*). In other words q_i^* *very weakly dominates* q_i. Individual policies which are dominated by another individual policy can be removed from the candidate pool of individual policies. This way the set of possible candidates for individual policies can be reduced greatly. Moreover, each time an individual policy is removed from candidate set of agent i, the candidate set of the other agent (denoted by $-i$) can be also

Algorithm 2.1 Iterative Elimination of Very Weakly Dominated Strategies

Input:Q_0,Q_1
Output: C_0,C_1
$C_0 \leftarrow Q_0$, $C_1 \leftarrow Q_1$
$pruned \leftarrow true$
while $pruned$
  $pruned \leftarrow false$
  for each agent i (1)
    $C_i^* \leftarrow RemoveDominatedPolicies(C_i, C_{-i})$ (2)
    if $C_i^* \neq C_i$ then $pruned \leftarrow true$
    $C_i \leftarrow C_i^*$
  end for
end while
return C_0,C_1

possibly reduced (as dominance for its policies depends partially on candidate pool agent i). So the whole process can be iterated until no individual policy can be pruned away.

The procedure very briefly described here is called *Iterative Elimination of Very Weakly Dominated Strategies* and is described in more detail in Algorithm 2.1 (for more details on how the dominated policies are found see Appendix A). C_i represents the candidate pool of individual policies for agent i, C_{-i} represents the candidate pool of the other agent. Q_i is the initial candidate pool generated by the *exhaustive backup* (see Section 2.2). Because all possible generalized beliefs are considered in testing dominance, *Hansen DP* does not need to estimate possible beliefs in which the subpolicies will be executed.

Hansen DP is described in Algorithm 2.2. As *Hansen DP* is a bottom up approach, it starts from the last time step $h-1$, constructs C_0^{h-1}, C_1^{h-1} and works its way back in time. When generating a candidate policy for agent i at stage t, an *exhaustive backup* (see Section 2.2) is performed. Finally C_i^t is pruned. At the first timestep $t = 0$, the pruned sets C_0^0, C_1^0 are used to generate all possible candidates for q which are evaluated (Equation 2.4) and the *joint policy* with the highest expected reward of b_0 is returned as the optimal solution.

As observed in [Hansen (2004)] and confirmed by experiments described in Appendix C, *very weak dominance* turns out to be too restrictive, as it requires q_i to be dominated for *all* possible *generalized beliefs*. As a result too many policy candidates are retained at each stage and *Hansen DP* cannot solve larger problems, or even small problem for all but the

Algorithm 2.2 Hansen Dynamic Programming

Output: $q \in Q^h$,$v \in \mathbb{R}$
$Q_0 \leftarrow Q_0^1$, $Q_1 \leftarrow Q_1^1$
$C_0^{h-1} \leftarrow Prune(Q_0)$,$C_1^{h-1} \leftarrow Prune(Q_1)$
for $t = h-2$ down to 0
  for each agent i
    $C_i^t \leftarrow GenerateFrom(C_i^{t+1})$
    $C_i^t \leftarrow Prune(C_i^t)$
  end for
end for
$Q^{final} = \left\{ \langle q_0, q_1 \rangle \,|q_i \in C_i^0 \right\}$
$q = \max_{q \in Q^{final}} V(q)$
$v = V(q)$
return q,v

smallest horizons (see Appendix C). More aggressive methods to reduce the candidate pool at each stage are suggested by algorithms discussed in the upcoming sections. While these methods can solve larger problems than *Hansen DP,* they do not guarantee optimality of the found solution. For a recent ε approximate method capable of solving larger problems see [Kumar and Zilberstein (2009)].

Bounded Dynamic Programming

In *Bounded Dynamic Programming* (*BDP* [Amato et al. (2007b)]) the memory consumption is restricted for each timestep, so that $\forall t, i \,|C_i^t| \leq maxTrees$[3]. This is achieved by a more aggressive pruning which is ruled by parameter ε. Therefore *BDP* can solve problems with slightly higher horizon than *DP,* although results are no longer guaranteed to be optimal (see Appendix C). Apart from modified pruning, *BDP* is identical with *Hansen DP*.

Similarly to *Hansen DP* a series of tests is performed to decide whether an individual policy q_i is dominated (and should be pruned away). Each of these tests compares two policies q_i and q_i^* and a linear program is constructed to find a *generalized belief* for which

[3]Multiple variants of *Bounded Dynamic Programming* are considered in Amato et al. (2007b). This work is concerned only with the *maxTrees* variant.

Algorithm 2.3 Pruning for BDP

Input:Q_0,Q_1
Output: C_0,C_1
$C_0 \leftarrow Q_0$, $C_1 \leftarrow Q_1$
$pruned \leftarrow true$
$maxTreesOK \leftarrow false$
$\varepsilon \leftarrow 0$
while !$maxTreesOK$ (1)
  while $pruned$
    $pruned \leftarrow false$
    $maxTreesOK \leftarrow true$
    for each agent i
      $C_i^* \leftarrow RemoveDominatedPolicies(C_i, C_{-i}, \varepsilon)$ (2)
      if $C_i^* \neq C_i$ then $pruned \leftarrow true$
      if $\left|C_i^{h-1}\right| > maxTrees$
        $maxTreesOK \leftarrow false$
      end if
      $C_i \leftarrow C_i^*$
    end for
  end while
  if !$maxTreesOK$ (3)
    $\varepsilon \leftarrow \varepsilon + 0.1$
  end if
end while
return C_0,C_1

q_i performs strictly better than q_i^* (see Appendix A). If such a generalized belief cannot be found, it means that q_i^* does never worse than q_i and therefore very weakly dominates q_i (which can be removed). In *Bounded Dynamic Programming* this condition is made more strict, it is not enough that the expected value of q_i is larger than that of q_i^* for some *generalized belief* (to avoid pruning), but the difference in values must be larger than $\varepsilon \geq 0$. In fact ε is dynamically increased until $\forall t, i\,|C_i^t| \leq maxTrees$. The larger ε is, the larger is the margin by which q_i has to outperform q_i^* (in order to survive the pruning). Thus the stricter condition enables *BDP* to prune more individual policies, which improves scalability. The *Bounded Dynamic Programming* pruning algorithm is summed up in Algorithm 2.3. This Algorithm differs from *Hansen DP* pruning in the addition of an outer while loop (1) which assures that pruning is performed until the size of both candidate pools is at most $maxTrees$. The routine $RemoveDominatedPolicies$ is now parametrized by ε in (2), which is the parameter for *very weak dominance testing* as described above. If one of the candidate pools contains too many candidates and no more pruning is possible in standard *IEVWDS*, ε is increased by 0.1 *(3)* to facilitate more pruning. This way ε can be increased until policy pools of size below $maxTrees$ are obtained.

2.2.2 Memory-Bounded Dynamic Programming

Memory-Bounded Dynamic Programming(*MBDP* [Seuken and Zilberstein (2007b)]) is similar to *BDP* in that it restricts the size of C_i^t to $maxTrees$ in each timestep t $(\forall t, i\,|C_i^t| \leq maxTrees)$. Contrary to *BDP, MBDP* does not perform an exhaustive backup and subsequent pruning. Instead, $maxTrees$ best performing joint policies are selected from exhaustive backup at each stage t and their individual policies form the individual policy candidate pool C_i^t.

The joint policy candidates are evaluated according to equations Equations 2.2-2.4. As the policies are constructed bottom up, at stage t the policy $q^{(0,t-1)}$ is not known (has not been constructed yet), and so is the belief b inferred by following of $q^{(0,t-1)}$. Therefore a heuristic is used to generate the belief b which is used to evaluate the *joint policy* candidates in 2.2-2.4. In the belief generation process, the actions are determined by the heuristic and the observations are sampled from the current belief (updated according to T,O from the

Algorithm 2.4 Memory-Bounded Dynamic Programming

Output: $q \in Q^h, v \in \mathbb{R}$
$heurPool \leftarrow fullyRandPol$
for $r = 1$ to $recDepth$ (1)
  for $t = h-1$ down to 0
    $C_0^t \leftarrow \emptyset, C_1^t \leftarrow \emptyset$
    if $t = h-1$
      $C^t \leftarrow Q^1$ (2)
    else
      $C^t \leftarrow GenerateFrom(C_0^{t+1}, C_1^{t+1})$ (3)
    end if
    for $n = 1$ to $maxTrees$ (4)
      $h \leftarrow sampleHeur(heurPool)$
      $b \leftarrow generateBelief(h)$
      $q = \langle q_0, q_1 \rangle \leftarrow findBestPol(C^t, b)$
      $C^t \leftarrow C^t \setminus \{\langle q_0, q_1 \rangle\}$
      $C_0^t \leftarrow C_0^t \cup \{q_0\}, C_1^t \leftarrow C_1^t \cup \{q_1\}$
    end for
  end for
  $Q^{final} = \{\langle q_0, q_1 \rangle | q_i \in C_i^0\}$
  $q = \max_{q \in Q^{final}} V(q)$ (5)
  $v = V(q)$
  $update(heurPool, q)$ (6)
end for
return q,v

Dec-POMDP model). Note that the same heuristic may generate different beliefs, due to uncertainty in the transition (T) and observation function (O).

The original paper suggested a pool of multiple heuristics to generate beliefs, but based on the results in Appendix B we decided to use a random policy as sole heuristics. [Seuken and Zilberstein (2007b)] also propose to use the solution obtained by *MBDP* as a heuristic to generate new solutions. This element of recursion in *MBDP* has a positive impact on the performance. As discussed in Appendix B, as a fully random policy performs better than any Q heuristic (for some domains, while on others it performs the same as the best alternative), the heuristic pool in my experiments consist of a single fully random policy and

the best performing policy from the previous recursive steps. The whole *MBDP* algorithm is summed up in Algorithm 2.4.

The outer loop (1) governs the recursive process of using policies found in previous recursions as heuristic. Firstly, the set of all possible *joint policy trees* of depth 1, Q^1, is used to initialized C^{h-1} in (2). In any subsequent stage t, pools of individual policy trees C_i^{t+1} found in previous stage $t+1$ are used to generate C^t in (3). Loop (4) selects the *maxTrees* best *joint policy trees*. After a policy q is selected by $findBestPol(C^t, b)$ for the generated belief b, it is removed from C^t and its individual components q_i are added into C_i^t. When timestep 0 is reached, the best joint policy q for b_0 is found (5). If its expected value v is higher than the highest value v^* (for policy q^*) found in previous recursions, $q*$ is replaced by q in the heuristics pool (6). Thus the *heurPool* contains at most two heuristics, the fully random policy and the best policy found in previous recursive steps.

While *Hansen DP* can find optimal solution for small horizons, and approximate *DP* methods provide for some scaling, the solvable horizon remain small (see Appendix C). The reason for this is the use of exhaustive backup to generate policy candidates, which requires much time and memory. One way to avoid this problem is to abandon exhaustive backup altogether as suggested in [Szer and Charpillet (2006); Amato et al. (2009)]. Another possible alternative is to approximate *Dec-POMDPs* by a series of *Bayesian Games* (*Bayesian Game Approximation - BAGA*), each of which corresponds to one timestep. As actions are assigned to each timestep separately, a full joint policy does not need to be constructed. In addition to good scaling properties, the *BAGA* method discussed in next section is an online algorithm, so it can utilize information obtained during runtime to find better solutions.

2.3 Bayesian Game Approximation Methods

This section is devoted to *Bayesian Game Approximation* (*BAGA*) methods, which approximate the *Dec-POMDP* by a series of *Bayesian Games* (*BGs*), each representing a separate timestep. While the *Bayesian Game Approximation* was coined in [Emery-Montemerlo et al. (2004)] as a name for a particular solution method, some of its elements have been adopted by other algorithms (for instance the *GMAA** family). Elements of *BAGA* used in

GMAA* are discussed in Section 2.3.5. The methods using *BAGA* in one step look-ahead fashion are referred to as *Forward-Sweep Policy Computation (FSPC)* methods. All methods introduced by [Emery-Montemerlo et al. (2004, 2005)] are *FSPC,* but the converse does not hold (some *GMAA** variants are *FSPC* too). In the next section the notion of *Bayesian Games with Identical Payoff* (*BGIP*) is introduced and a how a sequence of *BGIPs* can be used to estimate *Dec-POMDPs* is explained. The process that will be described next was first introduced in [Emery-Montemerlo et al. (2004)].

2.3.1 Bayesian Games with Identical Payoff

Bayesian Game (*BG*) is normally an augmented strategic game, in which individual agents posses some private information called *individual type*. The value of an outcome does depend both on the *joint action* and on the types of all agents (the *joint type*). As we consider *BGs* to approximate *Dec-POMDPs*, we are only concerned in a purely cooperative version of *BGs - Bayesian Game with Identical Payoff* (*BGIP*s). Formally a *BGIP* is a tuple $\langle \mathscr{D}, \mathscr{A}, \Theta, Pr(\Theta), u \rangle$ (notation adopted from [Oliehoek et al. (2010)]) where:

- $\mathscr{D}$ is a finite set of n agents
- $\mathscr{A} = \times_i \mathscr{A}_i$ is a finite set of possible *joint actions*. $\mathscr{A}_i$ is the finite set of *individual actions* for agent i.
- $\Theta = \times_i \Theta_i$ is a finite set of possible *joint types* $\theta = \langle \theta_0, \theta_1 \rangle$, where $\theta_i \in \Theta_i$ are the *individual types*
- $Pr(\Theta)$ is a probability distribution over *joint types*
- u is the reward function, a mapping from *joint types* and *joint actions* to a real number $u : \Theta \times \mathscr{A} \mapsto \mathbb{R}$[4]

As mentioned before in Section 2.1, we restrict ourselves to the two agent case for the sake of clarity and assume that $n = 2$ for the rest of this thesis. The set of agents $\mathscr{D}$ and actions $\mathscr{A}$ in the *BGIP* correspond directly to their counterparts in the approximated *Dec-POMDP*.

[4] General *BGs* differ from *BGIPs* in that each agent i has its own reward function u_i.

The *joint types* of *BGIP* approximating timestep t correspond directly to *JAOHs* of length t in the *Dec-POMDP* and consequently *IAOHs* map directly to *individual types* at the same timestep. Let us assume that the *BGIP* approximates timestep t. The probability of a joint type θ, $Pr(\theta)$ equals to the probability that *JAOH* corresponding to $\vec{\theta}^t$ occurs. The probability of each *joint type* θ can be calculated by each agent individually, as $\vec{\theta}^t$ entails *joint action history* $\overrightarrow{a_i^t} = (a_i^1, ..., a_i^t)$ and *joint observation history* $\overrightarrow{o_i^t} = (o_i^1, ..., o_i^t)$. This knowledge, together with initial belief b_0, transition function T and observation function O from the original *Dec-POMDP* are sufficient to calculate the probability of any *JAOH* $\vec{\theta}^t$. Therefore all agents can maintain individually the same distribution $Pr(\Theta)$, which is an important mean of coordination. The same information could be used calculate belief b^t resulting from b_0 after θ^t has occurred, and use the *Dec-POMDP* reward function R to calculate value of each joint action a given the belief b^t. Such an utility function u would consider only the immediate reward at each timestep and would result into a greedy/myopic planning algorithm. It is desirable to factor in expectations of reward in future timesteps into u and the therefore an admissible heuristic *QMDP* is used as the utility function in this work (see [Emery-Montemerlo (2005); Oliehoek et al. (2008b); Spaan et al. (2011)] for other suitable candidates).

The solution of the *BGIP* is a *joint policy* $\beta = \langle \beta_0, \beta_1 \rangle$, where an *individual policy* $\beta_i : \Theta_i \rightarrow \mathscr{A}$ maps an *individual type* θ_i to an individual action a_i. There are many solution methods to solve *BGs* and obtain β, in this work mostly Brute Force Solver and Alternative Maximization [Yokoo et al. (2003)] were used, for a more efficient solution method tuned to solve *BGIPs* consult [Oliehoek et al. (2010)].

So the Bayesian Game Approximation, in a nutshell, is described in Algorithm 2.5.

In practice, it is important to keep the size of $BGIP^t$ manageable. This is achieved in an additional step (a), in which multiple individual types are assigned to one cluster and treated the same, thus reducing the size of constructed $BGIP^t$ (in terms of the number of *individual* and *joint types*). Introduction of clustering also makes step 3 nontrivial, as it is now necessary to match *IAOH* $\vec{\theta}_i^t$ to an *individual type* corresponding to a cluster in the clustered $BGIP^t$. In the next section the processes of constructing $BGIP^t$, its clustering and matching *IAOH* $\vec{\theta}_i^t$ to an individual type used in *FSPC* are discussed in more detail. All of them were described in [Emery-Montemerlo et al. (2004, 2005)], with exception of

Algorithm 2.5 Essence of BAGA

1. For each timestep t a *BGIP* $BGIP^t$ is constructed. Each agent independently constructs the **same** $BGIP^t$.
   (a) Perform clustering on $BGIP^t$
2. Each agent solves $BGIP^t$ independently and obtains the **same** solution $\beta^t = \langle \beta_0^t, \beta_1^t \rangle$
3. Each agent matches its **individual** history $\vec{\theta}_i^t$ to its individual β^t and performs its **individual** action a_i^t.
4. The environment changes in reaction to the *joint action* $a^t = \langle a_0^t, a_1^t \rangle$ and each agent obtains its **individual** observation o_i^t of the joint observation $o^t = \langle o_0^t, o_1^t \rangle$. Both agents receive the **same** reward r^t.
5. Back to step 1.

the *Lossless* clustering. It originates from [Oliehoek et al. (2009)] and its adaptation into *BAGA* constitutes the original contribution of this thesis.

2.3.2 Constructing $BGIP^t$

Construction of the $BGIP^t$ for timestep $t = 0$ is trivial. There is only one *JAOH* $\vec{\theta}^0$ to be considered, corresponding to the empty observation. $Pr(\theta^0) = 1.0$ and u is defined by the used heuristics (typically *QMDP*). $\vec{\theta}^0$ practically represents an empty *JAOH*, so the *belief* over the state after the occurrence of this *JAOH* b^θ is equivalent to the initial b_0. As we shall see soon, the belief associated with a *joint type* θ (corresponding to *JAOH* $\vec{\theta}$*)* and the probability of θ in $BGIP^t$ are both important to extend $BGIP^t$ into the next timestep and create the subsequent $BGIP^{t+1}$.

For timestep $t > 0$ $BGIP^t$ is constructed from the *BGIP* for previous stage $t-1$, $BGIP^{t-1}$, and the solution for this game β^{t-1}. To each *joint type* $\theta \in \Theta^{t-1}$ corresponds a belief b^θ and a probability value p^θ in $BGIP^{t-1}$. The solution of $BGIP^{t-1}$, β^{t-1} also assigns a *joint action* $a^\theta = \beta^{t-1}(\theta)$ to each $\theta \in \Theta^{t-1}$. Each possible combination of a joint observation o^t, a^θ,θ corresponds to an extended *joint type* $\theta^e \in \Theta^t$ in the extended $BGIP^t$.

Shall we define an extension operator Ext over joint types as θ as

$$Ext(\langle \theta, a, o \rangle) = \theta^{a,o} = \theta^{e} \tag{2.6}$$

the whole extension procedure can be formalized by

$$Cand = \Theta^{t-1} \times \mathscr{O}$$

$$\Theta^{t} = \left\{ Ext(\theta, \beta^{t-1}(\theta), o) \mid \langle \theta, o \rangle \in Cand \right\}$$

If there would be no clustering involved θ^{e} would represent the *JAOH* $\vec{\theta}$ extended by a^{θ} and observation o^{t}. In general, all possible joint observations o^{t} have to be considered[5], but the *joint actions* are prescribed for each θ by $a^{\theta} = \beta^{t-1}(\theta)$. Using the transition function T and the observation function O, the belief $b^{\theta^{e}}$ and probability $p^{\theta^{e}}$ can be calculated from b^{θ} and p^{θ}. Thus we have shown how Θ^{t-1} can be extended into Θ^{t} and how $Pr(\Theta^{t-1})$ can be used to derive $Pr(\Theta^{t})$. As the utility function u is defined by the chosen heuristic and $\mathscr{D}$, $\mathscr{A}$ constituents of the $BGIP^{t}$ remain the same for all time steps, the procedure of extension of $BGIP^{t-1}$ to $BGIP^{t}$ is now fully defined.

2.3.3 Clustering

Following of the extension procedure explained above would result in a steady growth in size of *BGIPs* as the set of joint types increases with the set of observations $|\Theta^{t}| = |O| \,.\, \left|\Theta^{t-1}\right|$. This growth may hurt the scalability for larger horizons. Therefore right after the $BGIP^{t}$ is constructed and before it is solved a clustering step is performed (so that a smaller $BGIP^{t}$ is solved).

Clustering operates on *individual types*. A cluster is a set of *individual types*, one of which serves as the *representative* of this cluster. This representative is used to extend $BGIP^{t}$ into next timestep (it plays the role of θ in the process described in the above section), and it may be also used in clustering and type matching for some *BAGA* variants.

[5] See point 3 in Section 3.6.1 for an exception.

After $BGIP^t$ is constructed, each cluster consist of a single *individual type*. Then the clustering procedure identifies pairs of clusters to which a similarity measure is applied. If they are similar enough (according to the applied measure), the two sets of *individual types* corresponding to the clusters are merged. A representative is assigned to the newly created cluster, depending on the clustering method. The new cluster is added to the $BGIP^t$, replacing the two original clusters which are removed, thus reducing the size of Θ_i^t by one. This procedure is repeated for each agent i until the clustering method cannot identify more pairs of *similar types* or another stopping criterion of the clustering method is met. After the clustering is finished for all individual types, the probability function $Pr(\Theta^C)$ of the clustered *BGIP*, $BGIP_C^t$, needs to be adjusted, as both the sets individual types Θ_i^C, and the set of *joint types* Θ^C have been altered by clustering. The value of reward function has to be defined for all the new types, based on the reward of the replaced types.

Note that while each agent i performs the clustering individually, it is crucial for the coordination that the resulting $BGIP_C^t$ remains the *same* for all agents. If randomization is used in the clustering process (as in *Low Probability Clustering*) this is achieved by using the same seed for the pseudo random generator for all agents. Next the used similarity measures will be discussed.

Similarity Measure

Let us first focus on the case in which each individual type θ_i corresponds to an *IAOH* $\vec{\theta}_i$ for agent i (that is, before clustering). Considering all possible *IAOHs* of the other agent $-i$ (given that *BAGA* is used [6]) a belief b_{θ_i} can be calculated from formula

$$b_{\theta_i} = \sum_{\theta \in \Theta} b_\theta P(\theta|\theta_i) \tag{2.7}$$

where b_θ is the belief associated with *joint type* (corresponding to JAOH $\vec{\theta}$) θ in the considered *BGIP*, calculated as described in (Section 2.3.2). The conditional probability in (2.7) is defined by

$$P(\theta|\theta_i) = \frac{P(\theta \wedge \theta_i)}{P(\theta_i)} = \frac{Pr(\theta)}{P(\theta_i)} \tag{2.8}$$

[6] In the extension process for *BAGA* only a single action is considered for each joint type $a^\theta = \langle a_0^\theta, a_1^\theta \rangle = \beta^{t-1}(\theta)$, which reduces the individual actions for $-i$ in any *IAOH* to be a_{-i}^θ for some joint type θ.

$$P(\theta_i) = \sum_{\langle \theta_0^*, \theta_1^* \rangle \in \Theta, \theta_i^* = \theta_i} Pr(\theta^*) \tag{2.9}$$

where (2.9) sums the probability of all joint types $\theta^* \in \Theta$, in which the individual type of agent i is θ_i, to calculate the marginal probability $P(\theta_i)$.

Now if the belief generated by two different *individual types* $\theta_i^* \neq \theta_i$ for some agent i would be exactly the same and under the assumption that the other agent would act exactly the same for both *IAOHs* θ_i^*, θ_i, the immediate and expected future reward would also be the same. Therefore agent i would act exactly the same for both types and they could be merged into one type. This idea forms the jest of *Lossless Clustering* (although the assumption about the other agents policy is weakened, see (2.18)).

In the case of lossy clusterings used in this work, the similarity measure uses the belief about the state b_{θ_i} only and possible policies and *IAOHs* of other agent $-i$ are ignored altogether. The basic similarity measure used is the *worst-case reward difference* which operates on *reward profiles*.

A *reward profile* for *type* θ_i and agent *i,* r^{θ_i}, is a vector of $|\mathscr{A}|$ elements. Each element corresponds to an *joint action* $a \in \mathscr{A}$ and is defined by

$$r_a^{\theta_i} = E(a, b_{\theta_i}) \tag{2.10}$$

$$E(a, b_{\theta_i}) = \sum_{s \in S} b_{\theta_i}(s) R(s, a) \tag{2.11}$$

so effectively each element $r_a^{\theta_i}$ yields the expected immediate reward for action a in state described by b_{θ_i}. R is the reward function defined in *Dec-POMDP.* (2.10) is actually used only to calculate a reward profile of the true *IAOHs,* when it is matched to some *individual type* in *BGIP* (type matching).

Reward profiles used during clustering use the reward function of the *BGIP* itself (u)

$$r_a^{\theta_i} = E(u(\theta, a) | \theta_i) \tag{2.12}$$

$$E(u(\theta, a) | \theta_i) = \sum_{\theta \in \Theta} u(\theta, a) P(\theta | \theta_i) \tag{2.13}$$

where $P(\theta|\theta_i)$ is defined by (2.8). The generalization of *reward profile* for clusters is straightforward, it is the average of reward profiles individual types contained in cluster c_i.

$$r_a^{c_i} = E(r_a^{\theta_i}|\theta_i \in c_i) \tag{2.14}$$

Now we are ready to introduce *worst-case reward difference* for two *individual types* θ_i^*, θ_i as

$$\max_{a \in A} \left| r_a^{\theta_i} - r_a^{\theta_i^*} \right| \tag{2.15}$$

As it represents the maximum expected loss in reward for any possible action, it is ought to be minimized. In *BAGA, worst-case reward difference* is used only to perform type matching.

To assess similarity of two clusters which are candidates for merging, *worst-case expected loss* is used. This measure takes into account the probability of the considered clusters (only available in runtime) and is defined as follows

$$\max_{a \in A} \left[P(c^1) \left| r_a^{c^1} - r_a^c \right| + P(c^2) \left| r_a^{c^2} - r_a^c \right| \right] / P(c) \tag{2.16}$$

where c^1, c^2 are the candidates to merge and c is the cluster that would result from the merger. Therefore $P(c) = P(c^1) + P(c^2)$. The subscript denoting the involved agent i is omitted from the notation of clusters to increase clarity of (2.16). The *reward profiles* used in (2.16) may be the reward profiles of the clusters themselves (*cluster reward profiles*, see (2.14)) or the *reward profiles* of the cluster representatives (*representative reward profiles*, see (2.12)). Which of the two is used depends on the selected clustering algorithm. Next we shall describe the three clustering algorithms used in this work.

Low Probability Clustering

Is a fast, randomized, one pass algorithm, described in Algorithm 2.6.

Algorithm 2.6 Low Probability Clustering

Input: $indivTypes$,$threshold$
Output: $clusteredTypes$
$shuffled \leftarrow shuffle(indivTypes)$ (1)
$clusteredTypes \leftarrow shuffled$
$size \leftarrow |shuffled|$
for $i = 1$ to $size$
  $cluster_1 \leftarrow shuffled[i]$
  if $Pr(cluster_1) < threshold$
    $found \leftarrow findNearestNeighbor(i+1, size, i)$(2)
    $cluster_2 \leftarrow shuffled[found]$
    $newCluster \leftarrow cluster(cluster_1, cluster_2)$
    $setRepresentative(new, getRepresentative(cluster_2))$(3)
    $setUtility(new, getUtility(cluster_2))$
    $setProbability(new, P(cluster_1) + P(cluster_2))$
    $remove(clusteredTypes, cluster_1)$
    $shuffled[found] \leftarrow newCluster$
    $clusteredTypes[found] \leftarrow newCluster$
  end if
end for
return $clusteredTypes$

In *findNearestNeighbor representative reward profiles* are used to calculate *worst-case expected loss.*

At the beginning, each cluster consist of a single *individual type*, so the initial set of clusters is defined by the set of *individual types*. In (1) the set of *individual types* is randomly reordered (coordination among agents is achieved by using the same seed for pseudorandom generator). Then a loop is performed over the reordered $clusteredTypes$. For each cluster $cluster_1$ whose marginal probability (calclated by (2.9) for a single type clusters) falls below $threshold$, the most similar cluster $cluster_2$ is found, which comes after $cluster_1$ in the shuffled set of clusters $clusteredTypes$. $cluster_2$ is found in (2) which considers each cluster $cluster_*$ coming after $cluster_1$ and calculates the *worst-case expected loss* for the pair $cluster_1$, $cluster_*$. The $cluster_*$ resulting in least *worst-case expected loss* is assigned to $cluster_2$. *Representative reward profiles* are used to calculate *worst-case expected loss.* $cluster_1$ and $cluster_2$ are merged into a *newCluster*, the representative for *newCluster* is that of $cluster_2$ (and so is the utility). $cluster_1$ is removed from

clusteredTypes, and *newCluster* takes the place of $cluster_2$. So effectively, *Low Probability Clustering* takes individual types from low probability clusters and moves them to the most similar cluster coming later in the ordering, thus reducing the size of *clusteredTypes* by one in each iteration.

Min Distance Clustering

In *Min Distance Clustering,* clustering is not performed in single pass as in *Low Probability Clustering*. Instead it finds the most similar pair of clusters and merges them until the found clusters are not similar enough or the minimum number of clusters is reached (see Algorithm 2.7).

As before, set of individual types is used to initialize the *clusteredTypes* in (1). Similarity of all possible cluster pairs from *clusteredTypes* is evaluated (2), to find the two nearest neighbors. The similarity of two cluster corresponds to their *worst-case expected loss*, calculated using the *cluster reward profiles.* If this value (distance, (3)) is above the *threshold*, the behaviors optimal for the two cluster differs too much to merge them into one cluster and hence the clustering is stopped. The process is also stopped if the minimum number of clusters is reached. The procedure of merging the two clusters is the same as in *Low Probability Clustering*, with the exception that the most likely individual type is assigned as the representative of the new cluster (5). The utility of the more likely merged cluster is assigned to the new cluster (6). Because the ordering of *clusteredTypes* is no longer relevant in $findNearestNeighbor$, a new cluster can be added anywhere in *clusteredTypes*, and does not need to replace an old cluster. In the end result the size of *clusteredTypes* is reduced by one in each iteration of (4).

Because each performance of (4) requires a loop over the full *clusteredTypes*, *Minimum Distance Clustering* is a more expensive algorithm than *Low Probability Clustering.* On the other hand it also results in smaller *BGIPs* and the probability distribution over the clustered types reflects the original (non-clustered) distribution more accurately [Emery-Montemerlo et al. (2005); Emery-Montemerlo (2005)].

Algorithm 2.7 Min Distance Clustering

Input: $indivTypes$,$threshold$,$minClusters$
Output: $clusteredTypes$
$clusteredTypes \leftarrow indivTypes(1)$
$size \leftarrow |clusteredTypes|$
$cluster_1, cluster_2 \leftarrow findNearestNeighbor(1, size)(2)$
$dist \leftarrow worstCaseExpLoss(cluster_1, cluster_2)$ (3)
while $dist < threshold$ and $size > minClusters$ (4)
  $newCluster \leftarrow cluster(cluster_1, cluster_2)$
  $repr_1 \leftarrow getRepresentative(cluster_1)$
  $repr_2 \leftarrow getRepresentative(cluster_2)$
  if $P(repr_1) < P(repr_2)$ (5)
    $setRepresentative(new, repr_2)$
  else
    $setRepresentative(new, repr_1)$
  end if
  if $P(cluster_1) < P(cluster_2)$ (6)
    $setUtility(new, getUtility(cluster_2))$
  else
    $setUtility(new, getUtility(cluster_1))$
  end if
  $setProbability(new, P(cluster_1) + P(cluster_2))$
  $remove(indivTypes, cluster_1, cluster_2)$
  $insert(clusteredTypes, newCluster)$
  $size \leftarrow |indivTypes|$
  $cluster_1, cluster_2 \leftarrow findNearestNeighbor(1, size)$
  $dist \leftarrow worstCaseExpLoss(cluster_1, cluster_2)$
end for
return $indivTypes$

In $findNearestNeighbor$ and *worstCaseExpLoss cluster reward profiles* are used to calculate *worst-case expected loss.*

Lossless Clustering

Lossless Clustering [Oliehoek et al. (2009)] is the only clustering algorithm discussed here not introduced in the original work on *BAGA* [Emery-Montemerlo et al. (2004, 2005)]. It differs from the previous algorithms in that it performs *lossless* clustering on *individual types*.

Algorithm 2.8 Lossless Clustering

Input: $indivTypes$
Output: $clusteredTypes$
for all pairs $type_1, type_2$
  if $checkExactEquivalence(type_1, type_2)$
    $newCluster \leftarrow cluster(type_1, type_2)$
    $repr_1 \leftarrow getRepresentative(type_1)$
    $repr_2 \leftarrow getRepresentative(type_2)$
    if $P(repr_1) < P(repr_2)$
      $setRepresentative(new, repr_2)$
    else
      $setRepresentative(new, repr_1)$
    end if
    if $P(cluster_1) < P(cluster_2)$
      $setUtility(new, getUtility(cluster_2))$
    else
      $setUtility(new, getUtility(cluster_1))$
    end if
    $setProbability(new, P(cluster_1) + P(cluster_2))$
    $remove(indivTypes, type_1, type_2)$
    $insert(indivTypes, newCluster)$
end for
return $indivTypes$

Lossless Clustering (Algorithm 2.8) is similar to *Minimum Distance Clustering* in that it finds and merges similar pairs of clusters, until no more merging is possible. The merging process in both algorithms is the same. The key differences lies in the similarity measure used instead of *worst-case expected loss,* called the *Probabilistic Equivalence Criterion* (also introduced in [Oliehoek et al. (2009)]).

Two *IAOHs* are considered to be *probabilistically equivalent*, if they induce the same belief over the state s and the *IAOHs* of the other agent $\vec{\theta}_{-i}$. Formally two *IAOHs* of agent i $\vec{\theta}_{i,a}$ and $\vec{\theta}_{i,b}$ are *probabilistically equivalent* iff

$$\forall \vec{\theta}_{-i} \forall s P(s, \vec{\theta}_{-i} | \vec{\theta}_{i,a}) = P(s, \vec{\theta}_{-i} | \vec{\theta}_{i,b}) \tag{2.17}$$

which can be rewritten as

$$\forall \vec{\theta_{-i}} P(\vec{\theta_{-i}} | \vec{\theta_{i,a}}) = P(\vec{\theta_{-i}} | \vec{\theta_{i,b}}) \tag{2.18}$$

$$\forall \vec{\theta_{-i}} \forall s P(s | \vec{\theta_{-i}}, \vec{\theta_{i,a}}) = P(s | \vec{\theta_{-i}}, \vec{\theta_{i,b}}) \tag{2.19}$$

The expression $P(\vec{\theta_{-i}} | \vec{\theta_{i,a}})$ can be calculated similarly to (2.8) as

$$P(\vec{\theta_{-i}} | \vec{\theta_{i,a}}) = \frac{P(\vec{\theta_{-i}} \wedge \vec{\theta_{i,a}})}{P(\vec{\theta_{i,a}})} = \frac{Pr(\langle \theta_{i,a}, \theta_{-i} \rangle)}{P(\theta_{i,a})} \tag{2.20}$$

where the marginal probability of $P(\theta_{i,a})$ is calculated by (2.9).

Algorithm 2.9 Probabilistic Equivalence Test

```
Input: type_1,type_2
Output: equal
for all type_-i of the other agent -i (1)
   if P(type_-i|type_1) ≠ P(type_-i|type_2)
      return false
   end if
end for
for all type_-i of the other agent -i (2)
   b_1 = getBelief(⟨type_1,type_-i⟩)
   b_2 = getBelief(⟨type_2,type_-i⟩)
   for all s ∈ S (3)
      if b_1(s) ≠ b_2(s)
         return false
      end if
   end for
end for
return true
```

Algorithm 2.9 provides some details on the implementation of the *Probabilistic Equivalence Test*. Loop (1) considers all possible types $type_{-i}$ of the other agent $-i$, and checks the condition (2.18) for the two candidate types $type_1, type_2$. Only if condition (2.18) is fulfilled, the beliefs corresponding to relevant *joint types* (generated by (2)) are tested for equivalence in (3), to verify condition (2.19).

2.3.4 Type Matching

Because of the applied clustering, matching of *IAOH* to an *individual type* in the clustered *BGIP* is no longer straight forward. The belief induced by the *IAOH* $\vec{\theta}_i$ for agent i, b_i, is used to calculate the corresponding *reward profile* (see (2.10)) which is then matched to an individual type. These beliefs may differ among the agents (and therefore the coordination can not be guaranteed). The process of calculating the belief b_i is anologous to (2.7), but the set of considered *joint histories* is different (how this set is maintained will be described soon).

The reward profiled r^{b_i} induced by b_i is compared against the reward profiles corresponding to individual clusters r^{c_i} (depending on the clustering method). The cluster c_i^* with the least *worst-case expected loss* is selected and given the solution of the current *BGIP* $\beta = \langle \beta_0, \beta_1 \rangle$ each agent performs action $\beta_i(c_i^*) = a_i$ and receives the observation o_i. For *Low Probability Clustering* the *representative reward profile* of c_i is used for r^{c_i}, *Minimum Distance Clustering* and *Lossless Clustering* employ the *cluster reward profile* for this purpose.

Each agent maintains a set of beliefs H_i^t (corresponding to all possible past *JAOHs*) in order to calculate b_i equivalently to (2.10)

$$b_i = \sum_{\theta \in H_i^t} b_\theta P(\vec{\theta} | \vec{\theta}_i) \tag{2.21}$$

where $\vec{\theta}_i$ is the *IAOH* received by agent i. At the first timestep $t = 0$ both sets contain the initial belief b_0 and are extended at each timestep. As agent i has no knowledge about the other agents action and observation, it constrains the possible future *JAOHs* only by using its individual knowledge and the *BAGA* extension process[7]. As each individual agent maintains an identical copy of the constructed *BGIP* (which is *Common Knowledge*), it can restrict the possible set of *JAOHs* which it has to consider (and expect the other agent to do the same). Thus maximum level of coordination allowed by the constructed *BGIP* is guaranteed.

[7] as discussed in Section 2.3.2, only joint actions a ascribed to some type θ by $a = \beta(\theta)$ are considered. For some domains partial state information can be used to constraint the possible observations see point 3 in Section 3.6.1.

All possible *joint actions* (and *observations)* in which agent i performs its individual action a_i (and receives its individual observations o_i) are used to extend each belief $\theta \in H_i^t$ into next timestep and form the set H_i^{t+1}. This set of joint actions B_i^t can be defined through

$$B^t = \left\{ \beta^t(\theta) \mid \theta \in H^t \right\} \tag{2.22}$$

$$B_i^t = \left\{ a \mid a = \left\langle a_i^{'}, a_{-i}^{'} \right\rangle \in B^t \wedge a_i^{'} = a_i \right\} \tag{2.23}$$

and the extension process can be formalized (for the case without partial state information) via the *Ext* operator introduced in (2.6) as

$$Cand = H_i^t \times B_i^t \times \mathscr{O} \tag{2.24}$$

$$H_i^{t+1} = \{ Ext(c) \mid c \in Cand \} \tag{2.25}$$

If partial state information is available, a modified set of observations $\mathscr{O}_i^t$ has to be used.

$$\mathscr{O}_i^t = \left\{ o \mid o = \left\langle o_i^{'}, o_{-i}^{'} \right\rangle \in \mathscr{O} \wedge o_i^{'} = o_i \right\} \tag{2.26}$$

This extension process is essentially the same as the one described in Section 2.3.2, it only differs in the joint actions considered for the extension.

2.3.5 GMAA*

Similarly to the original *MAA** [Szer et al. (2005)], *GMAA** (*Generalized Multi Agent A**) is an offline planning algorithm which performs search in the space of *Partial Joint Policy Trees*. To each partial tree q^t a value $V(q^t)$, can be assigned which is the average reward obtained by following the policy corresponding to q^t. A heuristic H^k is used to estimate the highest possible value obtained by extending q^t $k > 0$ steps into future. The heuristic only evaluates the contribution of the extension itself, the reward obtained from timestep t on (it ignores the reward earned the first t steps (following q^t)). So an estimate of the value obtained by the optimal joint policy π^h following subpolicy q^t in first t timesteps can be

obtained by

$$F(q^t) = V(q^t) + H^{h-t}(q^t) \tag{2.27}$$

*GMAA** maintains a pool of candidate *partial joint policy trees*. In each iteration, the most promising candidate q^{*t} is selected and extended into q^{*t+1}. The extension is done through constructing the corresponding *BGIP* as described in Subsection 2.3.2. Each leaf node in q^{*t} corresponds to a *JAOH* $\vec{\theta}$ and assigns a joint action $a = q^{*t}(\vec{\theta})$ to it. All possible observations o are considered to extend $\vec{\theta}$ to $\theta^{\vec{a},o}$ and these extended *JAOHs* are used as the set of joint types Θ^{t+1} for $BGIP^{t+1}$ constructed as described in Section 2.3.2. $BGIP^{t+1}$ is then solved, and its solution β^{t+1} assigns a *joint action* to each possible *JAOH* $\vec{\theta}^{t+1}$, thus effectively extending q^{*t} into next timestep. The *GMAA*Cluster* variant performs *Lossless Clustering* discussed in the previous Section after each extension, to keep the size of *BGIPs* manageable. Shall q^{*t+1} turn out to be suboptimal, the second (third, and so on) next best solution to $BGIP^{t+1}$ can be used to generate an alternative extension to q^{*t+1}.

Apart from significant optimizations (such as early removal of the suboptimal subpolicies from the pool of candidates for extension), *GMAA** performs a standard A*. In each timestep the best candidate according to (2.27) q^{*t} is selected, its extension is generated, the value estimate of the extended policy is adjusted and it is added to the candidate pool. If all possible extensions of q^{*t} have been evaluated or q^{*t} is known to be suboptimal, it is removed from the candidate pool.

As in A* Search, for an admissible heuristics for H^k (such as *QMDP*) GMAA*(Cluster) is guaranteed to find the optimal solution. Because using of *Lossless Clustering* improves scalability significantly, *GMAA*Cluster* is used as the optimal benchmark in the experimental Chapter 3.

Resumé

This Chapter introduces the formalism necessary to describe *Dec-POMDPs* and covers two families of algorithms which solve them.

For *Dynamic Programming* a way to evaluate joint policies and multiple methods how construct joint policy candidates recursively bottom up are presented. The construction

methods also reduce the set of candidate policies considered by pruning (*Hansen DP, BDP*) or selection of suitable candidates (*MBDP*).

Bayesian Game Approximation is also introduced in full detail. The process of construction of Bayesian *Games with Identical Payoff*, possible methods of their clustering and type matching of the true *IAOH* to an individual type in clustered *BGIPs* are all presented.

In addition to *Low Probability Clustering* and *Minimum Distance Clustering* a new *Lossless Clustering* is introduced. The adaptation of *Lossless Clustering* into *BAGA* constitutes the original contribution of this thesis.

The end of this section is devoted to the *GMAA** and *GMAA*Cluster* offline solution methods, and the way they use *BGIPs* to generate new joint policy candidates (analogous to the *BGIP* extension process in *BAGA*).

In the next Chapter, the aforementioned solution methods are compared against each other on a set of benchmark problems, and the results of this experiments are analyzed.

Chapter 3

Experiments

In this chapter I shall evaluate the performance of the *Forward-Sweep Policy Computation* (*FSPC*) methods, mainly the variants of *BAGA*. Whenever possible, comparison with the *Dynamic Programming* (DP) and optimal solution methods is also available. As we shall see, the prior applies only to the smallest problems. For all reimplemented algorithms we provide results on benchmarks which were not used (and often not even introduced) in the time of their original publication. To the best of my knowledge, these results will be published in this thesis for the very first time.

For larger domains, the *DP* methods reimplemented here are unable to find a solution, so the comparison is narrowed down to the *Forward-Sweep Policy Computation* methods and *GMAA*Cluster*. Whenever *GMAA*Cluster* cannot solve the problem, the best known results, obtained by [Amato et al. (2009)], are presented. This method is approximate and offline and is not discussed in further detail within this thesis.

The largest domain discussed here is the *Robotic Tag 2*. As only variants of *BAGA* can solve it, they are compared against each other. Next I shall introduce the domains in more detail, and proceed to the analysis of results on these domains.

3.1 Domains

The reimplemented algorithms were evaluated on a set of standard benchmark problems, which are part of the *MADP Toolbox*. One more larger domain has been added into the

Table 3.1: The size of the respective domains

domain	$\|S\|$	$\|\mathscr{A}\|$	$\|\mathscr{O}\|$	$\|\Pi_2\|$	$\|\Pi_5\|$	$\|\Pi_{10}\|$
DecTiger	2	9	4	7.29e02	3.82e29	1.55e976
BroadcastChannel	4	4	4	6.4e01	4.61e18	8.08e615
Recycling	4	9	4	7.29e02	3.82e29	1.55e976
GridSmall	16	25	4	1.563e04	2.17e43	1.24e1430
BoxPushing	100	16	25	1.68e07	2.62e940	5.63e2939745
Mars	256	36	64	1.02e14	1.13e7285	!(6^306783378)
Robotic Tag 2	17577	25	784	3.47e40	2.08e891076	!(5^21940501236698)
Robotic Tag A	281217	25	784	3.47e40	2.08e891076	!(5^21940501236698)
Robotic Tag Small	1729	25	196	9.31e20	1.5e57834	!(5^44500716150)

S, $\mathscr{A}$, $\mathscr{O}$ are parts of the *Dec-POMDP* and correspond to the set of *states*, set of *joint actions* and set of *joint observations*. Π_t is set of all possible *joint policies* π_t of *horizon* t. For entries marked by '!' the number of policies could not be enumerated in exponential notation.

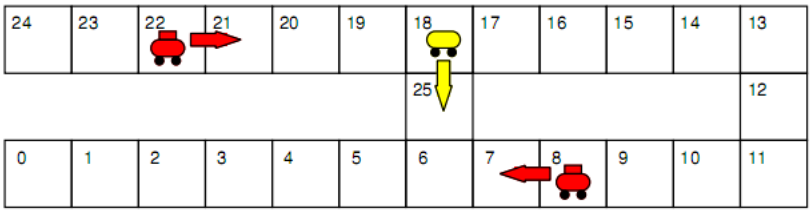
MADP Toolbox, the *Robotic Tag 2* [Emery-Montemerlo et al. (2004)]. Based on their size, the benchmarks can be divided into three groups:

- *Small domains*: *BroadcastChannel Problem* [Hansen (2004)], *DecTiger Problem* [Nair et al. (2003)], *GridSmall* [Amato et al. (2006)] and the *Recycling Robots* [Amato et al. (2007a)]. These domains can be solved (at least for small horizons) by all reimplemented algorithms (*DP, BDP, MBDP, BAGA*) and their optimal solution values are also known.

- *Medium sized domains*: *Box Pushing* [Seuken and Zilberstein (2007a)] and *Mars Stochastic Rover* [Amato and Zilberstein (2009)]. Neither *DP* (*BDP* respectively) nor *MBDP* can solve these problems. The optimal solution values are known for small horizons. The best performing approximate algorithms (such as *PBIP-IPG* [Amato et al. (2009)], *TBDP* [*Wu et al. (2010)*]) can provide solutions for larger horizons (of magnitude of 100, 1000). *BAGA* is able to solve them for the horizon 10 - 20, albeit only suboptimally.

- *Large Domain*: *Robotic Tag Variants*. *Robotic Tag 2* [Emery-Montemerlo et al. (2005)] can be solved by *BAGA* only. As we shall discuss in Section 3.6.2, partial state information is necessary to solve this problem efficiently. Unfortunately

Figure 3.1: Robotic Tag 2 Domain



The figure is taken from [Emery-Montemerlo (2005)]. Trials are run for a maximum of 15 timesteps.

it is not possible to integrate this additional information into *GMAA** based methods seamlessly, an therefore this methods cannot solve this problem for meaningful horizons. To the best of my knowledge the optimal solution values are not known.

More detail about the size of the respective domains is included in Table 3.1. Further details on all domains, with exception of *Mars* and *Robotic Tag* variants, together with a brief description of the domains can be obtained from [Oliehoek (2010)]. The description of Mars can be found in [Amato and Zilberstein (2009)], *Robotic Tag 2* is described next.

3.1.1 Robotic Tag 2

In the *Robotic Tag 2* a team of two robots attempts to tag an adversary in a corridor consisting of 26 cells (see Figure 3.1) . The action space of the robots consists of moving one cell in any cardinal direction or tagging. The adversary may also move the same way or remain in the same cell, and executes each of its actions with the same probability. All actions of the both robots are deterministic and never fail. The adversary is tagged, if it shares the same cell with at least one of the robots, which has to perform the tagging action.

Each robot can sense the adversary only if it is in the same cell. Additionally, each robot has a perfect knowledge of its own location as well as the location of its team mate. The location of the adversary is not known, unless one of the robots can sense it as described above. As we do not consider communication, both robots can know the position of the adversary only if all three of them are in the same cell.

The tagging is considered successful if the robot performing the tag action was in the same cell as the opponent. Please note that coordination on the tagging is not necessary and

it is sufficient for one robot to tag the opponent. Also the opponent can be tagged in any cell. Upon a successful tagging the robots receive a reward of 10 and the problem resets and assigns each agent (including the opponent) a new random starting location. If the tagging is not successful (because the tagging agent had a wrong belief about the location of the opponent), the robots receive reward of -10 and the herding continues. The robots are also penalized by -1 for each movement action, which gives them an incentive to (successfully) tag the opponent as soon as possible. Additionally the problem resets if they fail to capture the adversary within a given number of time steps. I opted here for 15 steps instead of 100 in [Emery-Montemerlo et al. (2004)], as later timesteps tend to be more time consuming for *BAGA*.

Due to the size of the problem, a script to generate a *.decpomdp* file has been implemented. Unfortunately the *MADP Toolbox* run out of memory when it has attempted to load the problem. Therefore a class representing this problem was created, in order to reduce the overhead caused by parsing and processing the *.decpomdp* file. However it would not compile because of the size of this class (the compiling process would be killed on time limit). Finally the problem was solved by splitting the *.decpomdp* into four parts (problem information, transition model, observation model and reward model). The same approach was applied to a larger variant of the same domain, *Robotic Tag A* [Emery-Montemerlo et al. (2005)], but because of the size of the problem information it would not compile.

3.2 GMAA*Cluster vs BAGA

To put the comparison of this these two algorithms into perspective, it is a good idea to keep in mind some of the key differences between them.

While *GMAA** can extend a *Joint Policy* φ^t multiple time steps ahead (also in multiple possible ways) and based on the new information resulting from this extension decide to abandon φ^t for the sake of an alternative policy φ'^t (a.k.a backtrack), this is not possible for *BAGA*. The *BAGA* can reason only one step ahead, and once it takes an action, it cannot revoke it in the future and try another alternative. Therefore we present here also the results

for *GMAA*K1*[1] which is a forward-sweep variant of *GMAA*Cluster*. The candidate *Joint Policy Pool* always contains only one (k=1) candidate, the most promising one, and the *Tree Search Algorithm* degenerates to *Best First Search*. This results in one step look ahead similar to *BAGA*, and makes *GMAA*K1* to a suitable baseline to which *BAGA* should be compared. Specifically *BAGA-Lossless* should yield results close to *GMAA*K1* as they also use the same clustering algorithm.

3.3 Experimental Settings

As both *DP* and *BDP* are deterministic algorithms, multiple trials are not necessary for evaluation. The same holds for *GMAA*Cluster*. Although GMAA*K1 uses a randomized *Bayesian Game with Identical Payoff Solver* (we use the *MADP Alternating Maximization Solver* [Yokoo et al. (2003)] with 30 restarts, the same solver is used by *BAGA*), in practice it always finds the optimal solution, so multiple trials were not performed.

For *MBDP* and *BAGA,* 10 respectively 1000 trials were performed. The values presented are the averages of these trials and the standard deviation is included in the parenthesis.

All experiments were run without discounting future expected reward (equivalently, the discount factor $\gamma = 1$). The results for *GridSmall* and *Recycling Robots* have been sometime published with $\gamma = 0.9$ in other works, so it is important to keep in mind that $\gamma = 1$ in this work, to avoid confusion.

In some cases the maximal horizon is constrained by some implementation details inherent in the *MADP Toolbox*. When such an implementation detail disabled finding the solution, this case is marked by an '?'. If the solutions could not be found due to the limitation of method itself '!' is used instead.

All experimens have been run on a Fujitsu Siemens Amilo Pa 1538 notebook equiped with a Turion 64 X2 TL-50 dual core processor and 2GB of memory. For more details on settings under which a particular algorithm was analyzed, consult Appendix B.

[1] GMAA*K1 stands for GMAA*kGMAACluster variant and should not be confused GMAA*kGMAA which does not perform Bayesian Game clustering

3.4 Results on Small domains

In this section I shall focus on the *Forward-Sweep Policy Computation* on *BroadcastChannel* and *Recycling Robots* compared against *GMAA*Cluster*. These domains are specific, as the optimal solutions for horizons well beyond 10 are known. As the *DP* methods do not scale to such large horizons, they performance is omitted here and is discussed in Appendix C. I shall first draw the general conclusion for both *DP* and *Forward-Sweep Policy Computation methods* for small horizons (6 and bellow). Then the performance of *FSPC* for larger horizons will be discussed.

(*Hansen*) *DP* is able to find the optimal solution, albeit only for horizon 2 and sometimes 3. *BDP* can increase the maximum solvable horizons by two, but the solution found is not optimal. While in some domains (*BroadcastChannel, Recycling Robots*) *BDP* achieves values close to optimum, for other domains the difference is more significant. Both *DP* and *BDP* have difficulties to solve *DecTiger*, presumably because of the importance of starting belief in this problem. As both use the starting belief only in the last timestep, it can not be used to guide pruning from early on [Amato et al. (2007b)].

MBDP can solve problems with small horizons optimally, but with growing horizon the solution quality slightly deteriorates. Increasing the *MaxTrees* and *recursion depth* parameter enables for solving problems with larger horizons, but it also makes the algorithm more time consuming.

The *BAGA* variants scale well and are able to find solutions close to optimum. *GridSmall* is a notable exception, as the observations provide little information about relevant state features, so *BAGA* is unable to maintain an useful belief about the state.

Next we shall focus on the two domains which can be solved even for larger horizons. The *BroadcastChannel* enables for very effective clustering [Oliehoek et al. (2009)] so that the *Bayesian Game* resulting from clustering contains only one *joint type*. As a consequence, each agent has the same belief about the state and this is the most accurate knowledge possible, given the inherent uncertainty. Interestingly the *MBDP* can solve this domain by maintaining only a single policy tree and without performing recursion. So the optimal solution is fairly easy to find. All three *BAGA* variants (*BAGA-LowProb*, *BAGA-MinDist*, *BAGA-Lossless*) perform close to optimal. As the *GMAA*K1* can solve

Table 3.2: Broadcast Channel Long Horizon Results

h	BAGA-LowProb	BAGA-MinDist	BAGA-Lossless	GMAA*K1	GMAA*C
2	2(±0)	2(±0)	2(±0)	2	2
3	2.991(±0.094)	2.988(±0.109)	3(±0)	2.99	2.99
4	3.885(±0.319)	3.879(±0.326)	3.888(±0.315)	3.89	3.89
5	4.793(±0.436)	4.843(±0.364)	4.72(±0.5)	4.79	4.79
6	5.661(±0.557)	5.75(±0.495)	5.776(±0.454)	5.69	5.69
100	?	90.375(±2.985)	90.328(±2.798)	90.29	!(90.76)
250	?	225.191(±4.669)	225.456(±4.799)	225.29	!(226.501)
500	?	450.166(±6.731)	450.424(±6.399)	450.29	!(452.738)
600	?	540.027(±7.280)	540.357(±7.046)	540.29	!(543.228)
700	?	630.373(±7.635)	630.356(±7.590)	630.29	!(633.724)
800	?	720.676(±8.604)	720.659(±8.336)	720.29	!
900	?	810.748(±8.838)	810.366(±8.969)	810.29	!(814.709)

For BAGA the value represents an average of 1000 trials. The value in parenthesis denotes the standard deviation. If the algorithm could not solve a horizon it is noted by '!' and the values in the following parenthesis are obtained by *GMAA*-ICE* [Spaan et al. (2011)]. If it fails to do so because of implementation issues it is marked by and '?'.

this domain close to optimum, multiple steps look ahead seems not to provide a significant advantage for this domain. For full results see Table 3.2.

For *Recycling Robots* the *BAGA* performs worse than both *GMAA** variants. As both *GMAA** variants perform the same, multiple steps look ahead is not necessary (the values of *GMAA**K1 are optimal for known horizons). *BAGA-Lossless* consistently outperforms the other two *BAGA* variants and is able to obtain values above 80% of the optimum (as apparent from Table 3.3). As *BAGA-Lossless* uses the same *BGIP Solver* and *BGIP Clustering* as *GMAA*K1*, the performance difference is likely to come from matching the true *IAOH* to the wrong *individual type*. Since the matching function is the *worst case reward difference*, only the value difference for the single worst action is considered. A more sophisticated similarity measure, that takes into account the two beliefs as a whole may yield results closer to *GMAA*K1*. Finding such a suitable measure may be an objective for future research.

The *Recycling Robots* domain has been chosen to compare the timings of all the methods. As apparent from Figure 3.2, methods performing *Bayesian Game Approximation*

Table 3.3: Recycling Robots Long Horizon Results

h	BAGA-LowProb	BAGA-MinDist	BAGA-Lossless	GMAA*K1	GMAA*C
2	6.794(±1.1052)	6.794(±1.1052)	6.776(±1.353)	6.8	7
3	10.333(±1.592)	10.333(±1.592)	10.823(±2.112)	10.66	10.66
4	10.59(±1.291)	10.59(±1.291)	10.846(±1.474)	13.38	13.38
5	11.118(±1.869)	11.118(±1.869)	13.638(±1.981)	16.486	16.486
6	13.79(±2.166)	13.79(±2.166)	17.236(±1.976)	19.554	19.554
10	22.328(±3.617)	23.026(±1.401)	24.052(±2.292)	31.864	31.864
20	46.386(±5.113)	44.892(±2.005)	56.834(±3.83)	62.633	!(62.633)
30	70.419(±5.935)	65.03(±1.993)	78.896(±4.367)	93.402	!(93.402)
40	95.566(±6.97)	84.99(±1.981)	110.092(±5.113)	124.172	!(124.172)
50	118.187(±7.694)	104.928(±1.981)	134.67(±5.638)	154.941	!(154.941)
60	141.173(±8.206)	124.94(±2.021)	173.65(±6.497)	185.71	!(185.71)
70	166.264(±9.094)	146.904(±2.32)	203.516(±7.008)	216.479	!(216.479)

For BAGA the value represents an average of 1000 trials. The value in parenthesis denotes the standard deviation. If the algorithm could not solve a horizon it is noted by '!' and the values in the following parenthesis are obtained by *GMAA*-ICE* [Spaan et al. (2011)].

Figure 3.2: Recycling Robots Timings

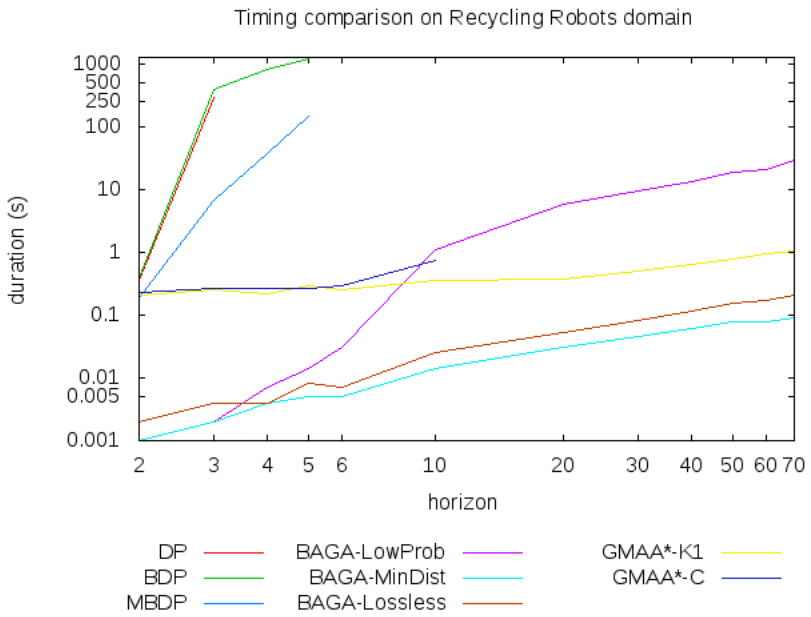
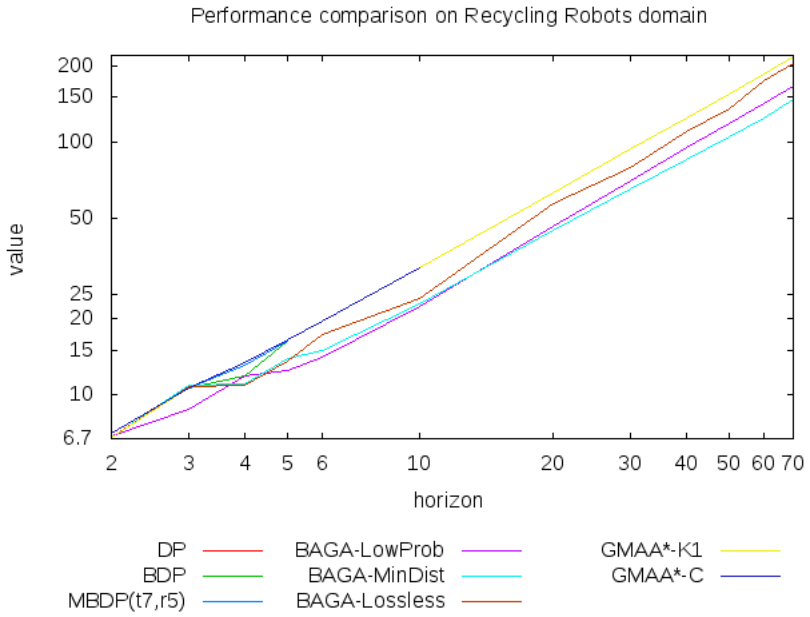


Figure 3.3: Recycling Robots Performance



(*GMAA*, BAGA*) are notably faster than those generating a *Joint Policy Pool (DP, BDP, MBDP). BDP* also performs an explicit trade off between time spent on additional pruning and the *Exhaustive Back Up (*more time spent on pruning is more time saved on the back up*).* Whether *BDP* actually speeds up *DP* (at the cost of possible loss of optimality) depends on which of the two phases is more time consuming. *GMAA** methods seem to have some overhead (compared to *BAGA),* which may be related to the evaluation of the policy candidates and maintaining the candidate pool.

The performance of algorithms on this domain in term of values is plotted in Figure 3.3.

3.5 Results on Medium domains

In the Medium Sized domains, the optimal values are known only for the lowest horizons, and even *GMAA*K1* cannot solve them for the larger horizons. This makes gaining meaningful insights about the domains more difficult. While some of the state of art algorithms

Table 3.4: Box Pushing Results

h	BAGA-LowProb	BAGA-MinDist	BAGA-Lossless	GMAA*K1	GMAA*C
2	17.76(±4.027)	17.54(±4.445)	17.38(±4.435)	17.6	17.6
3	65.85(±46.405)	65.8(±46.106)	67.96(±45.614)	66.081	66.081
4	52.368(±47.116)	64.055(±41.918)	66.306(±40.528)	98.594	98.594
5	0.286(±32.698)	25.683(±45.812)	25.283(±45.48)	104.864	!
6	-3.9554(±7.614)	-4.235(±5.687)	-4.483(±5.765)	!	!
10	-13.768(±13.521)	-19.547(±13.737)	?	!	!

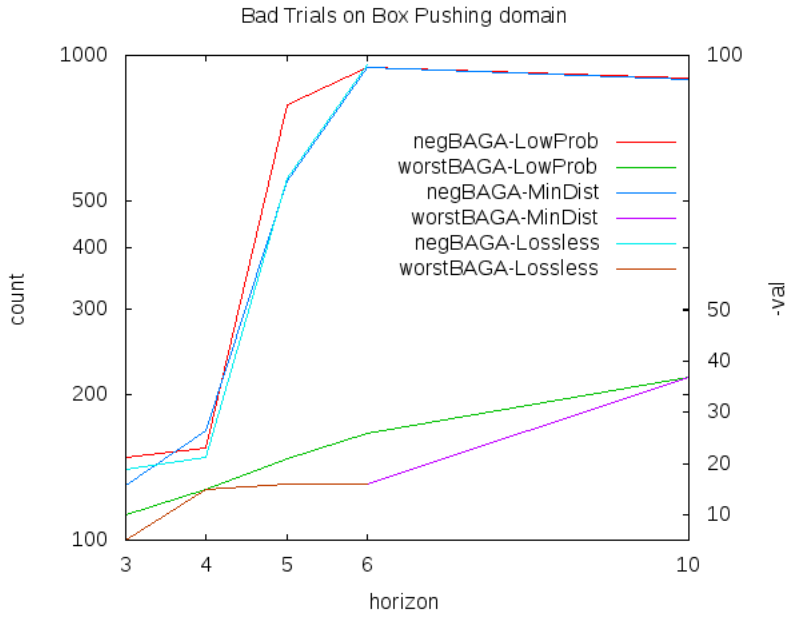
For BAGA the value represents an average of 1000 trials. The value in parenthesis denotes the standard deviation. If the algorithm could not solve a horizon it is noted by '!'.

[Amato et al. (2009); Wu et al. (2010)] are known to solve problems of this size approximately, the published results remain sparse.

We analyze the results of *FSPC* methods on the *Box Pushing* and *Stochastic Mars Rover*, which are often used as benchmarks for the state of the art algorithms. A scaled up version of *GridSmall* dubbed *Meeting on a 3x3 Grid* [Bernstein et al. (2005)] is also often used for this purpose, but as *BAGA* can not solve the smaller version, experiments on this domain have been omitted.

For the *Box Pushing* (Table 3.4), *BAGA* falls noticeably behind the optimal results even for horizon 4. Because of the abrupt decay for larger horizons, the maintained *BGIPs* and beliefs about the state do not seem to estimate the state for the larger horizons accurately. This may be due to the uncertainty which accumulates through many timesteps, and therefore it would affect lower horizon solutions to a lesser extent. The steeply growing number of trials which achieved a negative result (Figure 3.4) supports this hypothesis, as negative values may indicate miss coordination among the agents.

Figure 3.4: Number of negative value trials in Box Pushing



Beware! The worst values are plotted as their opposites. So value -9 is plotted as a 9.

For the *Stochastic Mars Rover* the *BAGA* variants performs suboptimally, although comparably to optimum up to horizon 4. The degradation is more graceful than in the *Box Pushing* domain. The performance of *BAGA-LowProb* does not terribly worse than the best known results (which are approximate, not optimal) achieved by *PBIP-IPG*. These results were published in [Amato et al. (2009)] and are presented in a separate column in Table 3.5, to highlight that no *PBIP-IPG* experiments have been carried out within the scope of this thesis.

Figure 3.5: Performance on Stochastic Mars Rover

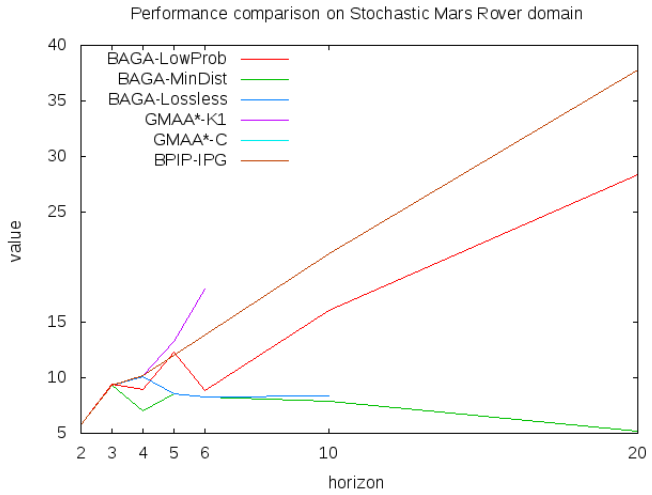


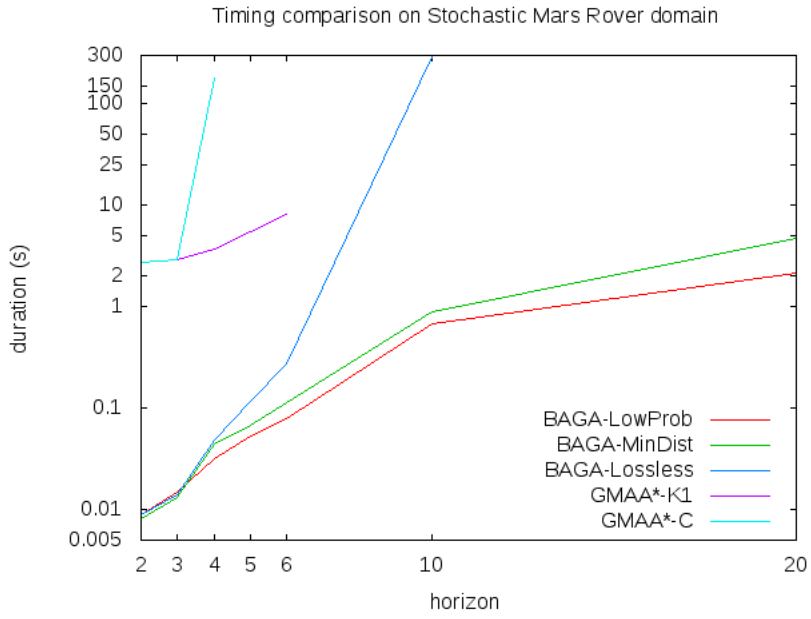
Table 3.5: Mars Results

h	BAGA-LowProb	BAGA-MinDist	BAGA-Lossless	GMAA*K1	GMAA*C	PBIP-IPG
2	5.792(±0.183)	5.78(±0.289)	5.788(±0.224)	5.8	5.8	5.8
3	9.403(±0.879)	9.38(±0.934)	9.386(±0.916)	9.38	9.38	9.38
4	8.973(±2.731)	6.98(±1.884)	10.062(±2.583)	10.181	10.181	10.18
5	12.298(±4.773)	8.538(±0.418)	8.546(±0.399)	13.267	13.267	?
6	8.812(±1.38)	8.304(±0.174)	8.29(±0.273)	17.991	!	?
10	16.044(±7.968)	7.833(±0.967)	8.351(±2.204)	!	!	21.18
20	28.298(±16.712)	5.184(±2.202)	?	!	!	37.81

For BAGA the value represents an average of 1000 trials. The value in parenthesis denotes the standard deviation. If the algorithm could not solve a horizon it is noted by '!'. The *PBIP-IPG* results were published in [Amato et al. (2009); Wu et al. (2010)] and experiments with *PBIP-IPG* have not been reproduced in this work.

Therefore *BAGA* seems to maintain a *BGIPs* and beliefs over state decently estimating the true state, and its performance could be improved with a suitable *IAOH* to *individual type* matching function in this domain. As this is the largest medium domain, it was chosen for further analysis of both timing and performance. The performance (timing) on this domain is visualized in Figure 3.5 (Figure 3.6). Although *BAGA* methods are noticeably faster and can solve problems of larger horizon than the *GMAA** variants, their solutions are inferior to *GMAA** beyond small horizons.

Figure 3.6: Timing on Stochastic Mars Rover



3.6 BAGA on Robotic Tag 2 Domain

3.6.1 Restricting the Belief size

Restricting the size of the individual *belief* b_i (in terms of number of histories considered) about the possible *joint action-observation histories* (*JAOH* in turn determines the state of the environment) is crucial. It enables us to scale to higher horizons, but it also provides a more accurate belief. The use of *Bayesian Game Approximation* enables us to manage the size of the *Belief* in three ways:

1. Explicitly, using *Clustering* after the *Bayesian Game* is extended into the next stage.

2. Implicitly by restricting the set of the possible *joint actions*. The solution of the *Bayesian Game with Identical Payoff* assigns a *joint action* to each *joint type*. If there are some actions which are not assigned to a type, they will be never performed, and can be ignored in extending the *Bayesian Game* to the next stage. This is always done in *BAGA* (and also *GMAA**).

3. Implicitly by restricting the set of possible *joint observations*. This is done only in the *Robotic Tag 2* (*Robotic Tag Small* respectively), as in this domain agents have partial information about the state (the precise location of both robots in the team). As the location of each robot is known with certainty, only a few *joint observations* are possible:

   (a) Robot 1 senses the adversary

   (b) Robot 2 senses the adversary

   (c) None of the robots senses the adversary

   Therefore we need to consider only 2-3 possible *joint observations* instead of the full set of 729. It is also applied in the process of generating the pool H_i^t to evaluate the belief induced by true *IAOHs* (see Section 2.3.4). This results in great speed up and the ability to solve problems with a larger horizon.

3.6.2 Full Robotic Tag 2

While generating, clustering and solving of a *BGIP* in *BAGA-Lossless* for timestep $t = 1$ takes typically less than a second when partial state information used, without this information it can take more than 1700 seconds. Thus the *RT 2* domain can be efficiently solved only with the additional state information. Unfortunately it is not possible to integrate this additional information into *GMAA** based methods seamlessly, an therefore this methods cannot solve this problem for meaningful horizons. It is important to realize that it not only restricts the growth of the *Bayesian Games*, but more importantly it reduces the size of probability distribution H_i^t above possible *JAOHs* which each agent i has to maintain. This distribution is used (see Section 2.3.4) to calculate the belief about the state induced by *IAOH,* which is then used to match the *IAOH* to an *individual type* (which in turn determines the *individual action* performed). As this probability distribution is not reduced in any way (contrary to the size of *BGIPs* reduced by clustering), it grows steadily from stage to stage. Thus the usage of partial state information enables for scaling to a larger horizons. Even when using the partial state information, stage 14 can take on average 2700 seconds to complete (ranging from less than 450 and more than 4700 second). Therefore I

have opted to stop herding after 15 timesteps instead of 100 used by [Emery-Montemerlo (2005)].

Table 3.6: Robotic Tag 2 results

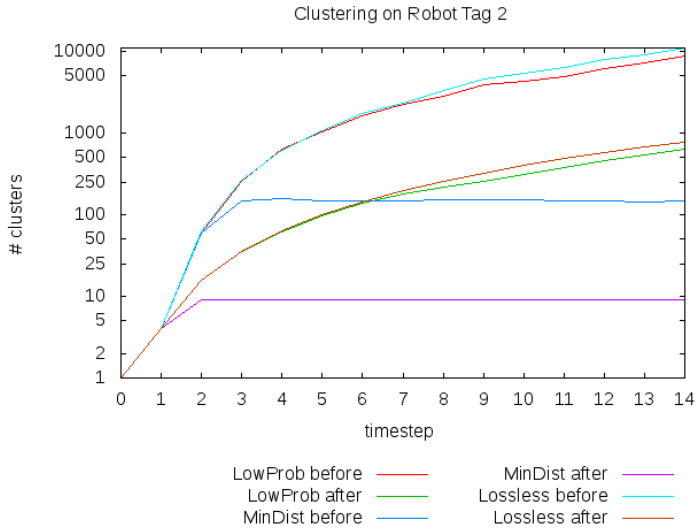
h	BAGA-LowProb		
	value	length	count
15	-1.95(±8.042)	9.65(±4.727)	100(23)
h	BAGA-MinDist		
	value	length	count
15	-1.71(±8.841)	9.01(±5.110)	100(27)
h	BAGA-Lossless		
	value	length	count
15	-2.83(±8.934)	9.63(±4.868)	100(32)

Length represents the number of timesteps needed to capture the adversary, count is the total number of trials. For value and length the values in parenthesis represent the standard deviation. For count, the number in parenthesis stands for number of trials finishing in the last (14th) timestep *without* capturing the adversary.

All *BAGA* variants can capture the adversary on average in 9-9.65 timesteps. Around 70% of trials ends in a successful capture of the adversary. Because the environment contains 26 cells, it is unclear whether the agents could capture it more frequently, if they were given more timesteps to capture. As the *Robotic Tag 2* cannot be run for much longer horizons, a more in depth analysis has been performed on a scaled down version of *Robotic Tag 2* which will be discussed in next section.

Figure 3.7 provides some insight on the impact of using different clustering methods. *BAGA-MinDist* maintains the least number of clusters. From timestep 3 on a typical *BGIP* constructed by *BAGA-MinDist* contains from 142 to 155 *joint types*, the *Clustered BGIP* contains only 9 *joint types* (3 for each agent). *BAGA-MinDist* needs the fewest number of steps to capture the adversary and does obtain the highest reward. The number of clusters for *BAGA-LowProb* and *BAGA-Lossless* grows steadily with the timestep on a similar rate.

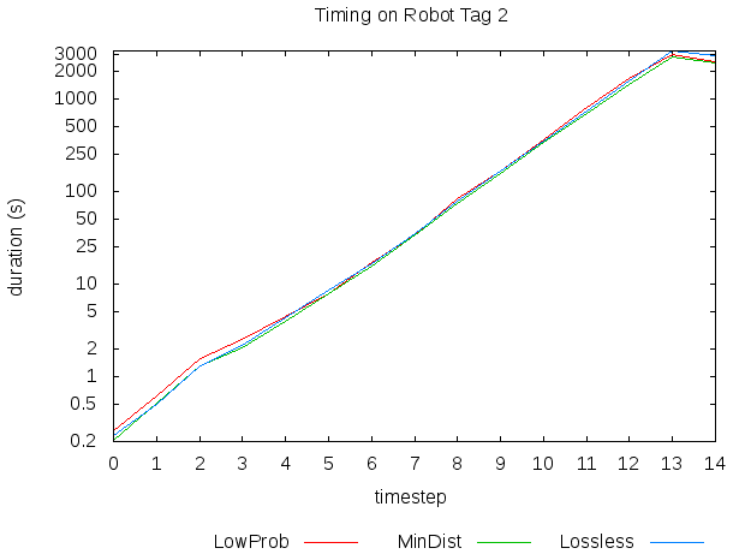
Figure 3.7: Clustering on Robotic Tag 2



The number of clusters is averaged from 30 trials of maximal length.

While the performance of *BAGA-LowProb* and *BAGA-MinDist* is similar, *BAGA-Lossless* does noticeably worse. It turns out that it has the highest number of trials terminated on timeout (32 compared to 23 and 27 respectively), and no run of length 15 resulted in capture (there were 7 such runs for both alternatives). As these runs obtain the worst possible reward (-16) this results in the observed performance difference. Other than that the distributions of lengths of the runs observed are fairly similar for all variants.

Figure 3.8: Timing on Robotic Tag 2

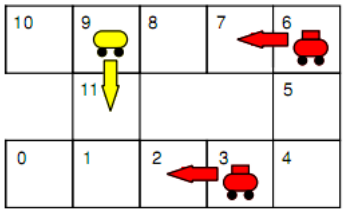


The times are averaged from 30 trials of maximal length.

Figure 3.8 reveals that time spend on each timestep is about the same for all three *BAGA* variants. While *BAGA-MinDist* performs slightly better for some timesteps, the difference is not too significant, despite of maintaining noticeable smaller *BGIPs.* This is explained by the fact that it is the maintaining the probability distribution H_i^t to calculate the *individual belief* (as explained in (2.21), see also Figure 3.11) which takes the majority of time. This procedure is the same in all *BAGA* methods, so the efficient clustering in *BAGA-MinDist* does not provide a significant speedup. In the last timestep the effect of the movement action is no longer relevant (as the agents do not 'arrive' within the considered horizon), so there are effectively only two actions to consider *move* and *tag*. This reduction of number of actions together, with efficient maintenance of H_i^t[2] can explain why the last timestep take less time than the timestep before.

[2]We don not keep multiple copies of identical beliefs in H_i^t. If multiple *JAOHs* produce the same belief, the sum of their probabilities is used as the likelihood of single belief in H_i^t.

Figure 3.9: Robotic Tag Small layout



Trials are run for a maximum of 18 timesteps.

3.6.3 Robotic Tag Small

Because the size of the *Robotic Tag 2,* it is not possible to run the problem for sufficiently large horizon. In order to circumvent this problem, experiments has been carried out on a scaled down version of *Robotic Tag 2*, dubbed *Robotic Tag Small.* It differs from the *Robotic Tag 2* in two key aspects:

1. The corridor contains only 12 cells, instead of original 26. The layout of the down sized corridor is pictured in Figure 3.9.

2. Trials are run for a maximum of 18 timesteps. Because of the smaller number of cells, agents are given enough time to visit the complete domain before the time out.

The increased number of timesteps enables an increase in the number of trials in which the adversary was captured to more than 94% for all *BAGA* variants (Table 3.7). *BAGA-MinDist* performs better than *BAGA-LowProb* and *BAGA-Lossless* outperforms them both.

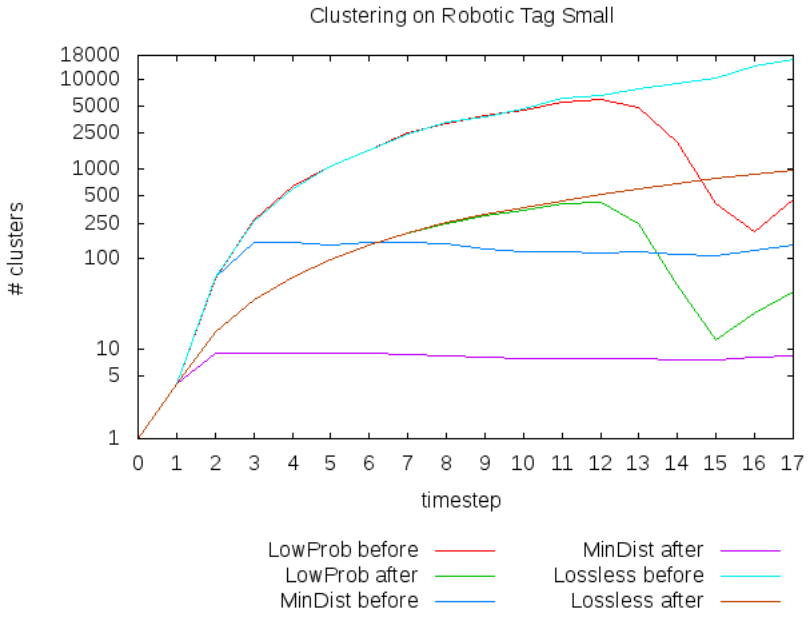
Table 3.7: Robotic Tag Small Results

h	BAGA-LowProb		
	value	length	count
18	4.213(±6.251)	5.257(±4.534)	1000(53)
h	BAGA-MinDist		
	value	length	count
18	4.495(±5.461)	5.165(±4.220)	1000(34)
h	BAGA-Lossless		
	value	length	count
18	4.756(±4.714)	5.064(±3.970)	1000(18)

Length represents the number of timesteps needed to capture the adversary, count is the total number of trials. For value and length the values in parenthesis represent the standard deviation. For count, the number in parenthesis stands for number of trials finishing in the 18th timestep *without* capturing the adversary.

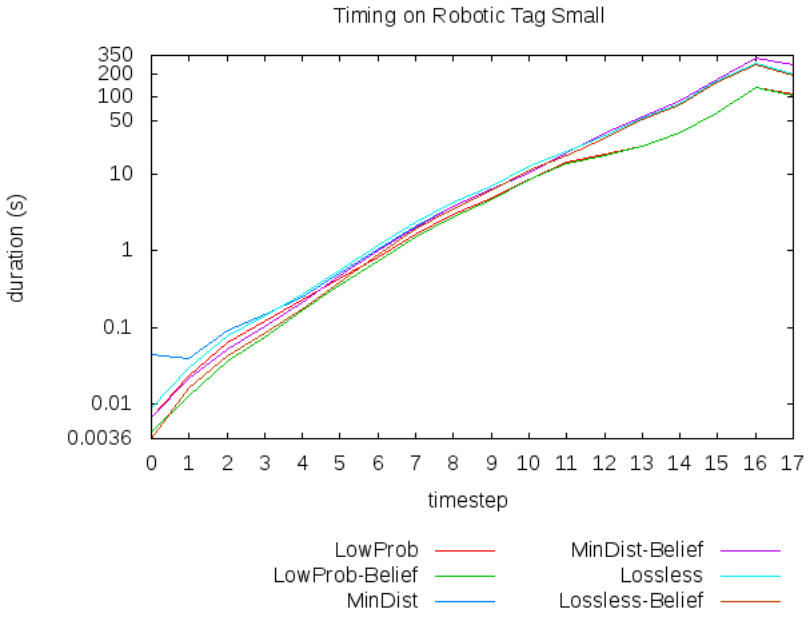
Figure 3.10 confirms the same trends as observed in Figure 3.7. Although the the size of the resulting *Clustered BGIP* does not remain constant for *BAGA-MinDist*, it oscillates in between 7.56 and 9.0 from timestep 2 on. An interesting phenomenon can be observed for *BAGA-LowProb*. As the size of *BGIP* steadily grows, the probability of *joint types* decreases. At some stage, a probability of many *joint types* falls below a threshold and is clustered away. The probability of *joint types* to which the low probability *joint types* are attached is increased, but still many *joint types* are unaffected by clustering. In next timestep some of these unaffected *joint types* (together with some joint types whose probability has not been increased significantly) are used to generate new joint types, whose probability falls below the threshold. Thus the size of the *extended BGIP* (and as consequence also the clustered one) can decrease in multiple subsequent timesteps, starting from timestep 12 to 16 in the described *Robotic Tag Small* experiment. Before this point, the size of *BGIPs* grows roughly at the same pace for *BAGA-LowProb* and *BAGA-Lossless* as observed in *Robotic Tag 2*.

Figure 3.10: Clustering on Robotic Tag Small



The number of clusters is averaged from 25 trials of maximal length.

Figure 3.11: Timing on Robotic Tag Small

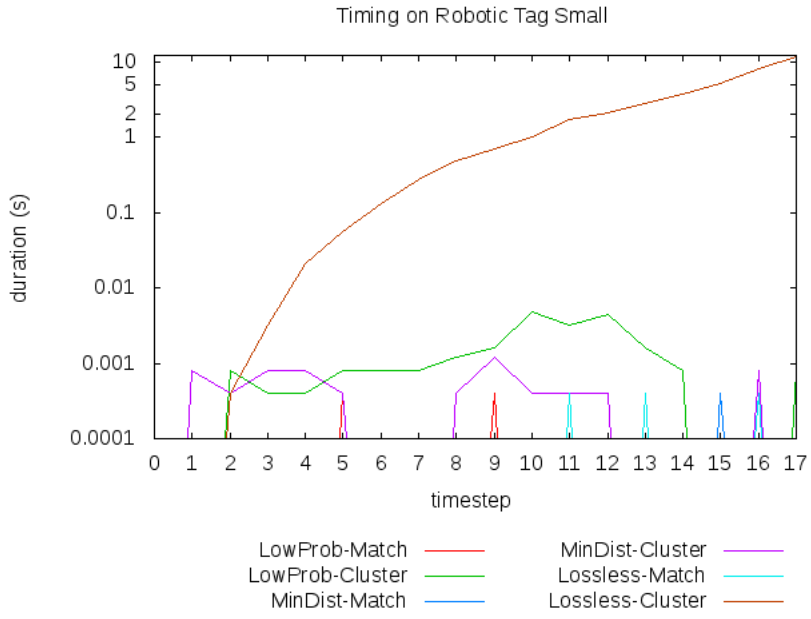


The times averaged from 25 trials of maximal length.

According to Figure 3.11 the *BAGA-Lossless* is the most time consuming followed closely by *BAGA-MinDist* until timestep 12, after which the order is reversed. *BAGA-LowProb* is faster than the two alternatives. From timestep 12, when the decrease of the size of *BGIPs* occurs, the difference becomes more significant. The last timestep take less than the timestep before for the same reason as in *Robotic Tag 2*.

The *BAGA-Variant Belief* data pictured represents the time used to calculate the individual belief given the *IAOH* in each timestep. Apparently from Figure 3.11 this operations consumes the wast majority of time. In Figure 3.12 the time consumption of the remaining *BAGA* operations is discussed, namely the clustering and matching of *IAOH* to an individual *type*. The time consumption of type matching is negligible, it is noticeable only for a few timesteps and takes always at most 0.0004 second. Regarding to clustering, *BAGA-MinDist* is faster than *BAGA-LowProb* for all horizons except 1,3 and 4. From timestep 12 to 16 the time spend for clustering by *BAGA-LowProb* drops, in accordance with the decrease of *BGIP* size as discussed above. *BAGA-Lossless* is the only clustering method whose time consumption is growing steadily, due to the complexity of the clustering algorithm.

Figure 3.12: Timing of Matching and Clustering on Robotic Tag Small



The times averaged from 25 trials of maximal length.

Resumé

In this Chapter multiple algorithms for solving finite horizon *Dec-POMDPs* have been evaluated on different benchmarks. Both offline Dynamic Programming *(DP*, *BDP, MBDP*) and online *Forward-Sweep Policy Computation* (*BAGA* variants) have been reimplemented and compared against each other and state of the art methods (mainly *GMAA*Cluster*).

While *DP* can find optimal solution for small domains and horizons, it does not scale well. The approximate variants (*BDP, MBDP*) scale slightly better, but they cannot guarantee an optimal outcome. *MBDP* outperforms *BDP* in both scaling and performance.

On the other hand, *FSPC* methods can solve larger problems with larger horizon and their results are often close to optimum. Among the evaluated *FSPC* methods the newly introduced *BAGA-Lossless* performs the best on many domains.

An additional contribution of this work is the inclusion of the *Robotic Tag 2* domain [Emery-Montemerlo et al. (2004)] into the *MADP Toolbox.* This domain is much larger than the domains present in the toolbox so far, and can be solved only by *FSPC* methods. In depth analysis on the use of clustering in solving the *Robotic Tag 2* and it scaled down version *Robotic Tag Small* has been presented in this chapter.

In conclusion, it can be said that *FSPC* scale well with the growing horizon and are able to solve larger problems than *GMAA*Cluster.* Although their performance is suboptimal, the found solutions come close to optimal results.

Therefore methods able to improve the performance of *FSPC (*such as finding a new matching measure for *BAGA-Lossless)* remain an interesting research avenue.

Chapter 4

Conclusion

In this thesis multiple offline *Dynamic Programming* (*DP*, *BDP*, *MBDP*) and online *Forward-Sweep Policy Computation* algorithms (*BAGA* variants) for solving finite horizon *Dec-POMDPs* have been reimplemented into *MADP Toolbox*. In addition, a new *BAGA* variant has been proposed, the *BAGA-Lossless,* which is the best performing variant in many investigated domains. These methods has been compared against each other and some of the state of art methods (*GMAA*Cluster*, *PBIP-IPG*) in terms of time and performance.

Many methods have been evaluated in new domains, and the results achieved in these domains are presented here for the first time. Performance of both *DP* and *BAGA* in these domains is analyzed in this thesis. A substantially larger *Robotic Tag 2* domain has been added into *MADP Toolbox.*

The *FSPC* methods have shown to be faster than the considered alternatives and capable of scaling to larger horizons than *DP* and *GMAA** variants. Moreover they perform close to optimum in most of the domains. *BAGA-Lossless* turns out to be the most promising online algorithm. Improving the type matching for this algorithms is identified as a possible research avenue to improve its performance.

While the *online FSPC* algorithms perform suboptimally to the best known offline planning methods in general, their speed and scalability make them a suitable candidates for future research.

Appendix A

Hansen Dynamic Programming

In *Hansen DP* [Hansen (2004)] finding an optimal solution of the *DecPOMDP* amounts to finding an optimal *individual policy tree* q_i for each agent i. For the sake of simplicity we shall again focus on the two agents case. The *resulting joint policy* tree $q = \langle q_0, q_1 \rangle$ is evaluated according to equations:

$$V^h(s, q, \vec{\theta}) = R(s, a) \tag{A.1}$$

$$V^t(s, q, \vec{\theta}) = R(s, a) + \sum_{s' \in S} \left[\mathscr{P}(s'|s, a) \left(\sum_{o \in \mathscr{O}} \mathscr{P}(o|a, s') V^{t+1}(s', q, \theta^{\vec{a},o}) \right) \right] \tag{A.2}$$

$$V(q) = \sum_{s \in S} b_0(s) V^0(s, q, \vec{\theta}) \tag{A.3}$$

Equation 2.2 forms the base of the recursion and describes how the final *joint action a* of *joint policy tree q* ($a = q(\vec{\theta})$) taken after a *JAOH* $\vec{\theta}$ has occurred and the environment is in state s should be evaluated. Equation 2.3 represents the recursive step and aggregates the immediate reward $R(s,a)$ with the expected future reward. In the future reward term, the value for each possible future state s' and future *JAOH* $\theta^{\vec{a},o}$ ($\theta^{\vec{a},o}$ stands for *JAOH* $\vec{\theta}$ extended by a and o) is weighted by the probability of occurrence of s' and $\theta^{\vec{a},o}$ given the action a. Action a is uniquely determined by q and past *JAOH* $\vec{\theta}$ as $a = q(\vec{\theta})$. The

probability functions in (A.2) are precisely the same as in (2.3) and are retrieved from the *DecPOMDP* model. Finally (A.3) explains how the value of the whole *joint policy tree q* can be calculated recursively. As $\vec{\theta}$ consist of an empty observation it can be omitted and $V^0(s,q,\vec{\theta})$ can be written as

$$V^0(s,q,\vec{\theta}) = V^0(s,\langle q_0,q_1\rangle) \tag{A.4}$$

We shall adopt a standard *Game Theory* notation where q_{-i} represents individual policies of all agents other than i, $q_{-i} = \langle q_0,...,q_{i-1},q_{i+1},...,q_{n-1}\rangle$. Thus the notation $q = \langle q_i,q_{-i}\rangle$ can be used to present any joint policy tree q from the perspective of agent i. As the value of an individual policy q_i can be calculated only from (A.4) only for a given s and q_{-i}, [Hansen (2004)] introduces the notion of *generalized state* defined by the pair $< s,q_{-i} >\in S\times Q_{-i}$. *Generalized belief* is defined as a probability distribution over *generalized states.*

To generate candidates for q it is not necessary to consider all possible combinations of *individual policy trees*[1]. If there exist two individual policy tree candidates q_i, q_i^* for agent i, for which the following condition

$$\forall q_{-i}\in Q_{-i},\forall s\in S : V^0(s,\langle q_i,q_{-i}\rangle) < V^0(s,\langle q_i^*,q_{-i}\rangle) \tag{A.5}$$

holds, q_i needs not to be considered as an optimal policy for agent i. (A.5) states that q_i^* always achieves higher outcome than q, no matter what is the state of the environment s or which policy q_{-i} is followed by the other agent. Therefore q_i is suboptimal to q_i^*, and should be removed (pruned away) from the candidate pool for optimal policies for agent i. In Game Theoretic jargon (A.5) expresses the fact that q_i^* *strictly dominates* q_i . Instead of (A.5) a weaker condition

$$\forall q_{-i}\in Q_{-i},\forall s\in S : V^0(s,\langle q_i,q_{-i}\rangle) \leq V^0(s,\langle q_i^*,q_{-i}\rangle) \tag{A.6}$$

is used. Condition (A.6) expresses the fact that q_i^* does at least as good as q_i (and possibly even better) in every situation (*generalized state*). In game theoretic terms, q_i^*

[1] as a policy tree is just a representation of a policy, we shall use the two term interchangeably from now on

very weakly dominates q_i. To determine whether q_i^* *very weakly dominates* q_i, the following linear program (see Algorithm (A.1)) is solved by [Hansen (2004)].

Algorithm A.1 Very weak dominance test

For variables: $b(s, q_{-i})$ and ε
Maximize ε
Such that

$$\sum_{<s,q_{-i}>\in S\times Q_{-i}} b(s,q_{-i}) \left[V^0\left(s,\langle q_i,q_{-i}\rangle\right) - V^0\left(s,\langle q_i^*,q_{-i}\rangle\right)\right] \geq \varepsilon \tag{A.7}$$

Fulfilling the probability constraints

$$\forall < s,q_{-i} > \in S\times Q_{-i} : 0 \leq b(s,q_{-i}) \leq 1$$

$$\sum_{<s,q_{-i}>\in S\times Q_{-i}} b(s,q_{-i}) = 1$$

(A.7) essentially attempts to find a *generalized belief* for which q_i does strictly better than q_i^* (thus attempts to find an counter example). When the $\varepsilon \leq 0$, such a case does not exist, and q_i^* *very weakly dominates* q_i fulfilling (A.6) and conversely q_i is *very weakly dominated* by q_i^*.

Hansen DP uses the game theoretic *Iterative Elimination of Very Weakly Dominated Strategies (IEVWDS)* to reduce the set of candidates for optimal individual policies C_i for each agent i. *IEVWDS* is described in Algorithm A.2 for the two agent case in more detail.

Algorithm A.2 Iterative Elimination of Very Weakly Dominated Strategies

Input:Q_0,Q_1
Output: C_0,C_1
$C_0 \leftarrow Q_0$, $C_1 \leftarrow Q_1$
$pruned \leftarrow true$
while $pruned$
  $pruned \leftarrow false$
  for each agent i (1)
    $C_i^* \leftarrow RemoveDominatedPolicies(C_i, C_{-i})$ (2)
    if $C_i^* \neq C_i$ then $pruned \leftarrow true$
    $C_i \leftarrow C_i^*$
  end for
end while
return C_0,C_1

Note that *RemoveDominatedPolicies* takes C_{-i} as parameter, as it specifies the *generalized state space* $S \times C_{-i}$ over which the *generalized belief* used in (A.7) is defined. So each time an agent manages to reduce its candidate set C_i, it may enable further pruning for C_{-i} in (2), and therefore loop (1) should be repeated as long as one of the agent can prune its candidate set.

Hansen DP is described in Algorithm A.3. As *Hansen DP* is an bottom up approach, it starts from the last time step $h-1$, constructs C_0^{h-1},C_1^{h-1}, and works its way back in time. When generating a candidate policy for agent i at stage t, each possible individual action a_i^0 in the root node is considered and to each possible individual observation o_i^0 a subpolicy $q_i \in C_i^{t+1}$ is assigned, which is to be followed shall o_i^0 occur. This way the pruning of C_i^{t+1} reduces the size of $C_i^{t'}$ for each $t' < t+1$. Finally C_i^t is pruned by *IEVWDS* (Algorithm (A.2)).

This recursive construction of policy candidates is the reason why the concept of *very weak dominance* (A.6) is applied instead of strict dominance (A.5). It is known that a policy which is *NOT strictly dominated* can have a *strictly dominated* subtree. As a consequence the optimal solution found by the described *Hansen DP* algorithm (using the strict dominance criterion) would not necessarily be the optimal solution to the original problem. However using of *very weak dominance* guarantees this, and therefore *Hansen DP* always finds the optimal solution (see [Hansen (2004)]).

Algorithm A.3 Hansen Dynamic Programming

Output: $q \in Q^h$,$v \in \mathbb{R}$
$Q_0 \leftarrow Q_0^1$, $Q_1 \leftarrow Q_1^1$
$C_0^{h-1} \leftarrow Prune(Q_0)$,$C_1^{h-1} \leftarrow Prune(Q_1)$
for $t = h-2$down to 0
  for each agent i
    $C_i^t \leftarrow GenerateFrom(C_i^{t+1})$
  end for
  $C_0^t, C_1^t \leftarrow Prune(C_0^t, C_1^t)$ //performs IEVWDS
end for
$Q^{final} = \left\{ \langle q_0, q_1 \rangle \,|q_i \in C_i^0 \right\}$
$q = \max_{q \in Q^{final}} V(q)$
$v = V(q)$
return q,v

At the first timestep $t = 0$, the pruned sets C_0^0, C_1^0 are used to generate all possible candidates for q which are evaluated according to (A.4), and $\max_{q \in Q^{final}} V(q)$ is returned as the optimal solution. [Amato et al. (2007b)] provides more details about the implementation of *Hansen DP.*

Appendix B

Experimental Settings

All experimens have been run on a Fujitsu Siemens Amilo Pa 1538 notebook equiped with a Turion 64 X2 TL-50 dual core processor and 2GB of memory. Most of the evaluated methods offer the possibility to influence their performance by adjusting some of their parameters. For the experiments whose outcomes are analyzed in Chapter 3 , the following parameter settings were used (unless explicitly stated otherwise in Chapter 3):

Dynamic Programming(DP[Hansen (2004)]) This method is deterministic in its nature and does not have any parameters. It is worthy of notice, that the outcome is significantly influenced by some arbitrary choices, such as the ordering in which the *joint policy* candidates are processed or which of the multiple policies of the same value should be selected for further processing . The memory management also influences the *Linear Programming Solver* we have used (lpsolve[1]), so that early memory freeing results in a more effective pruning.

Approximate Dynamic Programming(ADP[Amato et al. (2007b)]) The *MaxTrees* variant of ε pruning as described in [Amato et al. (2007b)] and Chapter 2 is employed. In this approach the ε bound starts with 0 value (equivalent to the standard *DP*) and if the size of the policy pool is larger than the *MaxTrees* threshold (*MaxTrees*=10) after the pruning phase, ε is increased by 0.1 and the pruning process is repeated until less than *MaxTrees* policies remain in *both individual* candidate

[1] http://lpsolve.sourceforge.net/5.5/

*policy pool*s. For each stage ε starts from 0 and is increased as described above, if necessary.

Memory Bound Dynamic Dynamic Programming(MBDP[Amato et al. (2007b)]) For all domains we keep 7 trees (*MaxTrees* = 7) and employ *recursion depth* of 5. Although the original paper has suggested to use a pool of multiple heuristics to generate beliefs, we have found out that using a Random policy[2] as the sole heuristics yields the best results as detailed in Table B.1.

For DecTiger and GridSmall domain it outperforms the *QMDP, QPOMDP, QBG* (for details on these heuristics see [Oliehoek et al. (2008b)]) and in Recycling Robots there is no performance difference between Random policy and the alternatives. ε denotes the exploration factor for the randomized variants of Q heuristics (which choose a random action with ε probability and follow the Q heuristics otherwise). The All column represents a pool consisting of all (ε randomized) Q heuristics (QMDP, QPOMDP, QBG) and a Random policy. The other columns represent pool containing only a single heuristic. QBG could not be computed for horizon 4 Recycling Robots. According to Table B.1 Random policy outperforms also the pool of All policies for DecTiger and GridSmall domains.

The poor performance of non random heuristics is rather surprising. Shall we use a pool containing multiple different heuristics, different heuristics may be used to evaluate the performance of policy candidates in subsequent timesteps. This 'inconsistency' in evaluation my hurt the performance in the 'All' case. For a single heuristic, one could speculate that the Q heuristics overestimate the future reward and select the wrong current action, resulting in a poor performance in the long run. But than one would expect that the tighter heuristics perform better, which is not the case (of course one could argue that they are not tight enough).

When recursion is performed, we keep only a single policy, which is the best performing policy from all the past recursions in that trial (not necessarily the best performing policy from the previous recursion).

The results for the *MBDP* are an average value among 10 trials (with the standard

[2]Random here means fully random, so every action is taken with the same probability under all circumstances

Table B.1: MBDP heuristics

h	ε	Random	QMDP	QPOMDP	QBG	All
DecTiger						
3	0	**5.191(±0)**	4.644(±1.094)	4.644(±1.094)	4.1(±1.805)	4.097(±1.34)
3	0.1	↑	4.37(±1.254)	4.097(±1.34)	4.097(±1.34)	4.9172(±0.821)
3	0.2	↑	4.097(±1.34)	3.823(±1.368)	4.37(±1.254)	4.097(±1.34)
4	0	**4.521(±0.313)**	2.816(±0.628)	2.567(±0.968)	3.194(±0.443)	3.194(±0.443)
4	0.1	↑	2.97(±0.926)	2.022(±1.42)	3.153(±0.652)	3.605(±1.067)
4	0.2	↑	2.621(±0.785)	2.774(±0.894)	3.014(±0.976)	3.201(±1.084)
Recycling Robots						
3	all	10.66(±0)	10.66(±0)	10.66(±0)	10.66(±0)	10.66(±0)
4	0	13.134(±0)	13.134(±0)	13.134(±0)	-	-
4	0.1	↑	13.134(±0)	13.11(±0)	-	-
4	0.2	↑	13.11(±0)	13.134(±0)	-	-
GridSmall						
3	0	**1.5504(±0)**	1.546(±0)	1.546(±0)	1.546(±0)	1.55(±0)
4	0	**2.189(±0.02)**	2.142(±0.001)	2.12(±0.111)	2.163(±0.039)	2.188 (±0.023)

The bold entries denote the single best heuristic. For horizon 4 Recycling Robots QBG could not be computed.

deviation noted in parentheses). The version of the *MADP Toolbox* we have been working with implicitly constrains the highest horizon for which a *Joint Policy Pool* could be constructed to 5 or 6 (depending on the domain size). In case of any similar limitation of the implementation and not the method the missing value is marked by a '?'. In contrast, '!' represents the instances when the method as such could not solve the problem.

Bayesian Game Approximation(BAGA[Emery-Montemerlo et al. (2004, 2005)]) We use the *MADP Alternating Maximization* [Yokoo et al. (2003)] with 30 restarts to solve the *Bayesian Games*. The *Lossless Clustering* is precisely the same as used by *GMAA*Cluster*. For *Low Probability Clustering* and *Minimum Distance Clustering* we use the threshold of 0.01. For *Minimum Distance Clustering* the value of 0.01 is used as the *maximum worst-case expected loss* stopping criterion and 2 is the minimal number of retained clusters stopping criterion. The results of BAGA are an

average value of 1000 trials by default. If such an evaluation would have taken too long, we performed fewer trails which is made apparent in the text.

GMAA*(GMAA*Cluster[Oliehoek et al. (2009)], GMAA*K1[Oliehoek et al. (2008b)])
Both versions use *Lossless Clustering*. *GMAA*K1* employs the the same *Bayesian Game Solver* as *BAGA*, *GMAA*Cluster* (abbreviated as GMAA*C in the tables and plots) uses the default *Brute Force Search Solver* (the default setting in *MADP Toolbox*).

Appendix C

Small Domain Results

BroadcastChannel

Despite of the small size of the *BroadcastChannel* domain, *DP* can solve the problem only for the smallest horizons. *BDP* can solve it for larger horizons, but its results are suboptimal. *MBDP* can solve the domain optimally up to horizon 5 and possibly more (see Table C.1).

The *BroadcastChannel* enables for very effective clustering [Oliehoek et al. (2009)] so that the resulting *Bayesian Game* contains only one *joint type*. As a consequence, each agent has the same belief about the state and this is the most accurate knowledge possible, given the inherent uncertainty. Interestingly the *MBDP* can solve this domain by maintaining only a single policy tree and without performing recursion. So the optimal solution is fairly easy to find. All three *BAGA* variants (*BAGA-LowProb*, *BAGA-MinDist*, *BAGA-Lossless*) perform close to optimal (even for significantly larger horizons as apparent from Table 3.2). For horizons 3 and 6, *BAGA-Lossless* achieves higher reward than the optimal offline algorithm *GMAA*Cluster* (*BAGA-MinDist* does better than *GMAA*Cluster* for horizons 3,5 and 6). See the concluding section of this Appendix for an explanation. *As* the *GMAA*K1* can solve this domain optimally, multiple steps look ahead seems not to be necessary for this domain.

Table C.1: Broadcast Channel Results

h	DP	BDP	MBDP	BAGA-LowProb	BAGA-Lossless	GMAA*C
2	2	2	2(±0)	2(±0)	2(±0)	2
3	2.99	2.9	2.99(±0)	2.991(±0.094)	3(±0)	2.99
4	!	3.8	3.89(±0)	3.885(±0.319)	3.888(±0.315)	3.89
5	!	4.7	4.79(±0)	4.793(±0.436)	4.72(±0.5)	4.79
6	!	!	?	5.661(±0.557)	5.776(±0.454)	5.69

If the algorithm could not solve a horizon it is noted by '!'. If it fails to do so because of implementation issues it is marked by an '?'.

DecTiger

For the *DecTiger* domain *DP* cannot scale beyond horizon 2. *BDP* performs poorly on this domain in accordance with [Amato et al. (2007b)]. The authors explain it by the importance of starting condition in this problem, which is not used in the *(B)DP* pruning phase until the last time step.

MBDP can solve the problem for horizons 2,3 using the default setting. To obtain optimal solution for higher horizons (4,5) is possible, but in order to get in on each trial, the number of kept policies (*MaxTree*) needs to be increased significantly (to 13, more than 20 respectively). For that many trees the algorithm becomes quite time consuming. For illustration, one trial for horizon 4, 13 trees and recursion level 5 (and always finds the optimal solution) takes more than 750 seconds on average, which is more than 5 times longer than the default. I have also experimented with pre- and post-pruning (and both together), but did not find any performance gain as published in [Seuken and Zilberstein (2007b)]. Contrary to the mentioned work, a noticeable slow down was experienced, perhaps due to the usage of a different *Linear Programming Solver*.

As apparent from Table C.2 *BAGA-Lossles* seemingly outperforms the optimal offline solution method (*GMAA*Cluster*) for all horizons but 5 and 2 (for horizon 2 both perform the same). *BAGA-LowProb* does also better than *GMAA*Cluster* for horizon 3 and 6. See the concluding section of this Appendix for an explanation. Horizons which are multiples of three have special importance in this domain, as the optimal strategy requires the agents to gather data, until they obtain the same observation in two consecutive timesteps and than act upon this information. If there is no noise, this boils down to perform information

gathering twice and than act, which gives us the period of 3. For horizons of 5 (and to a lesser extent 4), there are multiple possible scenarios in which a noisy observation spoils the coordination between agents. This results in smaller reward for this horizon for all algorithms, in particular *BAGA-Lossless.*

It should be also noted, that there is noticeable difference between *GMAA*Cluster* and *GMAA*K1* in horizons which are not multiples of 3. For these cases multiple steps look ahead seems to provide an advantage to *GMAA*Cluster.*

Table C.2: DecTiger Results

h	DP	BDP	MBDP	BAGA-LowProb	BAGA-Lossless	GMAA*K1	GMAA*C
2	-4	-4	-4(±0)	4(±0)	-4(±0)	-4	-4
3	!	-8.75	5.191(±0)	5.714(±23.689)	5.947(±23.989)	5.191	5.191
4	!	-10.75	4.52(±0.313)	4.774(±21.25)	5.412(±18.399)	3.191	4.803
5	!	-8.192	3.509(±0.517)	1.867(±22.739)	0.667(±26.384)	1.191	7.026
6	!	!	?	10.63(±33.168)	12.321(±30.892)	10.382	10.382

If the algorithm could not solve a horizon it is noted by '!'. If it fails to do so because of implementation issues it is marked by an '?'.

The rather large standard deviation can be explained by the nature of the domain. Both agents are required to coordinate their actions in opening one of the two doors. If the wrong door is opened both agents receive a penalty of -50,-100,-101 (depending on the precise *joint action*), but if only the proper door is opened, the reward is only 9 or 20 (again depending on the precise *joint action*). Even though agents can select the proper action most of the time, the occasional cases when the miss coordination occurs (due to noise) have an order of magnitude larger detrimental impact. This increases the range of the obtained values and therefore also the *standard deviation*. As *MBDP* is an offline algorithms, it finds a *joint policy* performing well on average, and thus the difference among individual trials is less significant.

While in [Emery-Montemerlo et al. (2005)] the authors could reach optimal results, they have used a special heuristics tuned for this domain.

Recycling Robots

For *Recycling Robots* the *BAGA* performs worse than both *GMAA** variants (see Table C.3). As both *GMAA** variants perform the same, multiple steps look ahead is not necessary. *BAGA-Lossless* consistently outperforms *BAGA-LowProb* (*BAGA-Lossless* and BAGA-*MinDist* perform about the same) and is able to obtain values above 80% of the optimum. This is true even for substantially larger horizons (see Table 3.3). The situation in horizon 3 for which *BAGA-Lossless* outperforms *GMAA*Cluster* is addressed in the last section of this Appendix.

Table C.3: Recycling Robots Results

h	DP	BDP	MBDP	BAGA-LowProb	BAGA-Lossless	GMAA*C
2	7	7	7(±0)	6.794(±1.105)	6.776(±1.353)	7
3	10.66	10.607	10.66(±0)	10.333(±1.592)	10.823(±2.112)	10.66
4	!	11.873	13.11(±0.073)	10.59(±1.291)	10.846(±1.474)	13.38
5	!	16.23	16.363(±0.042)	11.118(±1.869)	13.638(±1.981)	16.486
6	!	!	?	13.79(±2.166)	17.236(±1.976)	19.554

If the algorithm could not solve a horizon it is noted by '!'. If it fails to do so because of implementation issues it is marked by an '?'.

GridSmall

In *GridSmall BAGA* is again inferior to *GMAA** (Table C.4). Although multiple steps look ahead seems to have a positive influence (because of the difference between *GMAA*K1* and *GMAA*Cluster*), it is not the only reason of the worse performance of *BAGA*. It should be mentioned that in this domain the agents are required to exhibit tight coordination, while they cannot observe each other. As *BAGA* relies also on individual observations to build up the individual beliefs of the agents, it is more affected than the offline planning methods. Again, the case of horizon 2 for which *BAGA-Lossless* outperforms *GMAA*Cluster* is discussed in the next section.

Table C.4: GridSmall Results

h	DP	BDP	MBDP	BAGA-LowProb	BAGA-Lossless	GMAA*K1	GMAA*C
2	0.91	0.584	0.91(±0)	0.888(±0.260)	0.915(±0.242)	0.91	0.91
3	!	0.637	1.55(±0)	1.386(±0.533)	1.411(±0.544)	1.55	1.55
4	!	0.339	2.189(±0.02)	1.901(±0.615)	1.912(±0.584)	2.24	2.24
5	!	!	?	2.414(±0.701)	2.408(±0.685)	2.96	2.97*
6	!	!	?	2.767(±0.834)	2.741(±0.841)	3.7	3.72*

If the algorithm could not solve a horizon it is noted by '!'. If it fails to do so because of implementation issues it is marked by an '?'. Results marked by an '*' are optimal result obtained by another offline algorithm (*GMAA*-ICE* [Spaan et al. (2011)]).

BAGA-Lossless outperforming Offline optimal solution.

For all domains *BAGA-Lossless* seemingly outperforms the optimal offline solution obtained by *GMAA*Cluster* for some horizons. To investigate whether this is the case, or the performance difference is caused by the sampling of the trials alone, many more trials have been performed for the relevant domains and horizons. Results of this trials are reported in Table C.5.

Table C.5: Long Trials for BAGA-Lossless outperforming optimum

h	25K	50K	70K	90K	100K	GMAA*C
DecTiger						
3	5.168(±24.667)	5.144(±24.648)	5.195(±24.51)	5.359(±24.237)	5.372(±24.205)	5.190812
4	3.05(±24.524)	2.549(±25.599)	2.634(±25.567)	2.716(±25.427)	2.768(±25.325)	4.802755
Recycling Robots						
3	10.598(±2.102)	10.616(±2.097)	10.625(±2.094)	10.637(±2.096)	10.638(±2.096)	10.66
GridSmall						
2	0.909(±0.257)	0.912(±0.256)	0.912(±0.256)	0.912(±0.255)	0.912(±0.258)	0. 91

For some domains the better performance was clearly just an issue of sampling (*DecTiger* horizon 4). For *GridSmall* horizon 2, *BAGA-Lossless* attains average reward of 0.910157 even for 1 million trials. We have also performed a trial based evaluation of the optimal policy returned by *GMAA*Cluster*. It has revealed that this optimal policies have a standard deviation of the same order of magnitude as *BAGA*. There has been a case in which the average reward obtained by *GMAA*Cluster* in 100K trials was higher than the

expected optimum. Therefore the sampling of the trials is the the most likely explanation of the observed behavior. In particular, GMAA*K1 forms an theoretical performance upper bound for *BAGA-Lossless* (as they are the same in everything but type matching). The performance of *BAGA-Lossless* is nonetheless very good, as it can quickly find a very good approximation of the optimal solution.

Bibliography

C. Amato and S. Zilberstein. Achieving goals in decentralized POMDPs. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, pages 593–600, Budapest, Hungary, 2009.

C. Amato, D. S. Bernstein, and S. Zilberstein. Optimal fixed-size controllers for decentralized POMDPs. In *Proceedings of the AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, pages 61–71, Hakodate, Japan, 2006.

C. Amato, D. S. Bernstein, and S. Zilberstein. Optimizing memory-bounded controllers for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, Vancouver, British Columbia, 2007a.

C. Amato, A. Carlin, and S. Zilberstein. Bounded dynamic programming for decentralized POMDPs. In *Proceedings of the AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, Honolulu, Hawaii, 2007b.

C. Amato, J. S. Dibangoye, and S. Zilberstein. Incremental policy generation for finite-horizon DEC-POMDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 2–9, Thessaloniki, Greece, 2009.

R. Arunachalam and N. M. Sadeh. The supply chain trading agent competition. *Electronic Commerce Research and Applications*, 4(1):63–81, 2005.

D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4): 819–840, 2002.

D. S. Bernstein, E. A. Hansen, and S. Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1287–1292, Edinburgh, Scotland, 2005.

C. Block, J. Collins, W. Ketter, and C. Weinhardt. A multi-agent energy trading competition. Technical Report ERS-2009-054-LIS, RSM Erasmus University, Rotterdam, The Netherlands, 2009.

A. Carlin and S. Zilberstein. Value-based observation compression for DEC-POMDPs. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems*, pages 501–508, Estoril, Portugal, 2008.

R. Emery-Montemerlo. *Game Theoretic Control for Robot Teams*. Dissertation, Carnegie Mellon University, 2005.

R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *In Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 136–143, 2004.

R. Emery-Montemerlo, G. Gordon, and J. Schneider. Game theoretic control for robot teams. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1175–1181, 2005.

E. Giesen, W. Ketter, and R. Zuidwijk. An agent-based approach to improving resource allocation in the dutch youth health care sector. In *Proceedings of the European Conference on Information Systems (ECIS)*, pages 2403–2415, Verona, Italy, July 2009.

E. A. Hansen. Dynamic programming for partially observable stochastic games. In *In Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, 2004.

H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. RoboCup Rescue: search and rescue in large-scale disasters as a domain for autonomous agents research. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International*

Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028), volume 6, pages 739–743. IEEE, 1999.

A. Kumar and S. Zilberstein. Dynamic programming approximations for partially observable stochastic games. In *Proceedings of the Twenty-Second International FLAIRS Conference*, pages 547–552, Sanibel Island, Florida, 2009.

R. Nair, M. Tambe, M. Yokoo, D. Pynadath, S. Marsella, R. Nair, and M. Tambe. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *In IJCAI*, pages 705–711, 2003.

F. A. Oliehoek. *Value-Based Planning for Teams of Agents in Stochastic Partially Observable Environments*. PhD thesis, Informatics Institute, University of Amsterdam, Feb. 2010.

F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008a.

F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008b.

F. A. Oliehoek, S. Whiteson, and M. T. J. Spaan. Lossless clustering of histories in decentralized POMDPs. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, pages 577–584, May 2009.

F. A. Oliehoek, M. T. J. Spaan, J. S. Dibangoye, and C. Amato. Heuristic search for identical payoff Bayesian games. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, pages 1115–1122, 2010.

Z. Rabinovich, C. V. Goldman, and J. S. Rosenschein. The complexity of multiagent systems: The price of silence. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1102–1103, July 2003.

S. Seuken and S. Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, Vancouver, British Columbia, 2007a.

S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 2009–2015, Hyderabad, India, 2007b.

S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

M. T. J. Spaan and F. A. Oliehoek. The MultiAgent Decision Process toolbox: software for decision-theoretic planning in multiagent systems. In *Multi-agent Sequential Decision Making in Uncertain Domains*, 2008. Workshop at AAMAS08.

M. T. J. Spaan, F. A. Oliehoek, and C. Amato. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *Proc. of International Joint Conference on Artificial Intelligence*, pages 2027–2032, 2011.

D. Szer and F. Charpillet. Point-based dynamic programming for dec-pomdps. In *21st National Conference on Artificial Intelligence - AAAI'2006*, Boston/USA, June 2006.

D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 576–583, Edinburgh, Scotland, 2005.

S. van der Putten, V. Robu, H. La Poutré, A. Jorritsma, and M. Gal. Automating supply chain negotiations using autonomous agents: a case study in transportation logistics. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, pages 1506–1513, New York, NY, USA, 2006.

F. Wu, S. Zilberstein, and X. Chen. Trial-based dynamic programming for multi-agent planning. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence*, pages 908–914, Atlanta, Georgia, 2010.

M. Yokoo, R. Nair, S. Marsella, M. Tambe, and D. V. Pynadath. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 705–711, 2003.

S. Zilberstein, R. Washington, D. S. Bernstein, and A.-I. Mouaddib. Decision-theoretic control of planetary rovers. In *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents*, pages 270–289, London, UK, 2002. Springer-Verlag.