Heuristic Search for Identical Payoff Bayesian Games

Frans A. Oliehoek
Informatics Institute,
University of Amsterdam
Amsterdam, The Netherlands
F.A.Oliehoek@uva.nl

Matthijs T.J. Spaan
Inst. for Systems and Robotics
Instituto Superior Técnico
Lisbon, Portugal
mtjspaan@isr.ist.utl.pt

Jilles S. Dibangoye
Laval University, Canada
University of Caen
Basse-Normandie, France
gdibango@info.unicaen.fr

Christopher Amato
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
camato@cs.umass.edu

ABSTRACT

Bayesian games can be used to model single-shot decision problems in which agents only possess incomplete information about other agents, and hence are important for multiagent coordination under uncertainty. Moreover they can be used to represent different stages of sequential multiagent decision problems, such as POSGs and DEC-POMDPs, and appear as an operation in many methods for multiagent planning under uncertainty. In this paper we are interested in coordinating teams of cooperative agents. While many such problems can be formulated as Bayesian games with identical payoffs, little work has been done to improve solution methods. To help address this situation, we provide a branch and bound algorithm that optimally solves identical payoff Bayesian games. Our results show a marked improvement over previous methods, obtaining speedups of up to 3 orders of magnitude for synthetic random games, and reaching 10 orders of magnitude speedups for games in a DEC-POMDP context. This not only allows Bayesian games to be solved more efficiently, but can also improve multiagent planning techniques such as top-down and bottom-up algorithms for decentralized POMDPs.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Theory, Performance, Experimentation

Keywords

Decision-Making under Uncertainty, Cooperative Multiagent Systems, Bayesian Games, Cooperative Game Theory

1. INTRODUCTION

Bayesian games (BGs) offer a rich model for analyzing decision-making with incomplete (partial) information [7]. In a Bayesian game, each player possesses a type which is not revealed to the other players. Decisions must be made by considering the possible types of the other players as well as the possible actions that may be taken. This results in a model that is able to represent many areas of multiagent decision-making under uncertainty.

In addition to the many cases that can modeled as competitive games with incomplete information like coalition formation [2], Bayesian games are also common in cooperative scenarios. For instance, identical payoff Bayesian games are a central component of the decision making in decentralized POMDPs (DEC-POMDPs) [1, 9, 6]. In DEC-POMDPs each agent has local information about the environment in the form of observation histories that are not shared with the other agents. Identical payoffs are used because agents are cooperative and thus seek to maximize a joint objective. Because DEC-POMDPs are sequential problems, Bayesian games can be used at each step to determine the actions that maximize the value for the different observation histories that may occur [6]. BGs have also been employed in general-payoff sequential settings—modeled by partially observable stochastic games (POSGs) [5]—for the control of a team of robots [4].

While Bayesian games can model many common multiagent scenarios, to the best of our knowledge only a single optimal algorithm is available, namely brute-force evaluation. In this paper, we present BaGaBaB, a branch and bound algorithm for solving identical payoff BGs. This algorithm is able to exploit the structure of the game to solve it more efficiently. This is accomplished by approaches such as using heuristics based on centralized values, avoiding expansion of invalid nodes, ordering the search-tree nodes based on action contribution and memory efficient representations.

To demonstrate the performance of our approach, we test the branch and bound algorithm on a set of randomly generated games as well as those encountered in the use of a DEC-POMDP solver PBIP [3]. We show marked improvement over brute-force evaluation. This is promising for solving large BGs as well as scaling up DEC-POMDP algorithms.

2. BAYESIAN GAMES

A strategic game of imperfect information or *Bayesian game* [7] is an augmented strategic game in which the players hold some private information. This private information defines the *type* of the agent. That is, a particular type

Cite as: Heuristic Search for Identical Payoff Bayesian Games, Oliehoek, Spaan, Dibangoye and Amato, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 1115-1122

$\theta_i \in \Theta_i$ of an agent i corresponds to that agent knowing some particular information. The payoff that the agents receive depends not only on their actions, but also on their private information. Formally, a BG with identical payoffs (IP) is defined as follows:

Definition 1. A *Bayesian game with identical payoffs* is a tuple $\langle \mathcal{D},\mathcal{A},\boldsymbol{\Theta}, \Pr(\boldsymbol{\Theta}),u\rangle$, where

- $\mathcal{D}$ is the set of n agents,
- $\mathcal{A} = \{\mathbf{a}^1, \ldots ,\mathbf{a}^{|\mathcal{A}|}\}$ is the set of joint actions $\mathbf{a} = \langle a_1, \ldots ,a_n\rangle$, with $a_i \in \mathcal{A}_i$ an individual action of agent i,
- $\boldsymbol{\Theta} = \times_i \Theta_i$ is the set of joint types $\boldsymbol{\theta} = \langle \theta_1, \ldots ,\theta_n\rangle$, with $\theta_i \in \Theta_i$ an individual type of agent i. $\boldsymbol{\theta}^k$ denotes the k-th joint type and θ_i^k denotes the k-th individual type of agent i,
- $\Pr(\boldsymbol{\Theta})$ is the probability function specified over the set of joint types, and
- $u : \boldsymbol{\Theta} \times \mathcal{A} \to \mathbb{R}$ is the payoff function for the team.

In a BG, the agents can condition their action on their type. This means that the agents use policies that map types to actions. We denote a joint policy $\boldsymbol{\beta} = \langle \beta_1,...,\beta_n\rangle$, where β_i is the individual policy of agent i. We consider deterministic (pure) individual policies that are mappings from types to actions $\beta_i : \Theta_i \to \mathcal{A}_i$.

The *value* of a joint policy is its expected payoff:

$$V(\boldsymbol{\beta}) = \sum_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \Pr(\boldsymbol{\theta}) u(\boldsymbol{\theta},\boldsymbol{\beta}(\boldsymbol{\theta})), \tag{1}$$

where $\boldsymbol{\beta}(\boldsymbol{\theta}) = \langle \beta_1(\theta_1),...,\beta_n(\theta_n)\rangle$ is the joint action specified by $\boldsymbol{\beta}$ for joint type $\boldsymbol{\theta}$. For a BG with identical payoffs, a solution is guaranteed to exist in deterministic BG-policies and this solution is given by $\boldsymbol{\beta}^* = \arg\max_{\boldsymbol{\beta}} V(\boldsymbol{\beta})$. This solution constitutes a Pareto optimal Bayes-Nash equilibrium. The standard approach of solving general (non-IP) BGs is to convert them to a normal-form game and then use standard solution methods to solve it. In the IP case, this approach reduces to brute force search (BFS): every deterministic $\boldsymbol{\beta}$ is evaluated using (1) and the optimal one is maintained. There also are approximate solution methods for BGs, such as alternating maximization [4], but these only guarantee to find an (arbitrarily worse) local optimum solution.

Note that the value of a joint BG-policy is defined as the sum of payoffs generated by joint types. We refer to this as the *contribution* for this joint type, defined as

$$C_{\boldsymbol{\theta}}(\mathbf{a}) \equiv \Pr(\boldsymbol{\theta}) u(\boldsymbol{\theta},\mathbf{a}).$$

Complete Information Assumption

In a BG, each agent only knows its own individual type. In this paper, however, we also consider a heuristic that assumes complete information and relaxes this requirement. If we assume that both agents can observe the joint type, then they could employ a different kind of policy: one that maps from *joint* types to actions. That is, we define complete information (CI) policies as follows. A joint CI policy is a tuple $\boldsymbol{\Gamma} = \langle \Gamma_1, \ldots ,\Gamma_n\rangle$ where an individual CI policy Γ_i maps *joint types* to *individual actions* $\Gamma_i : \boldsymbol{\Theta} \to \mathcal{A}_i$. The joint action specified for $\boldsymbol{\theta}$ is

$$\boldsymbol{\Gamma}(\boldsymbol{\theta}) = \langle \Gamma_1(\boldsymbol{\theta}), \ldots ,\Gamma_n(\boldsymbol{\theta})\rangle = \mathbf{a}_{\boldsymbol{\theta}}^{\boldsymbol{\Gamma}}.$$

		θ_2^1		θ_2^2	
		a_2	$\bar{a}_2$	a_2	$\bar{a}_2$
θ_1^1	a_1	-0.3	$+0.6$	-0.6	$+4.0$
	$\bar{a}_1$	-0.6	**$+2.0$**	-1.3	**$+3.6$**
θ_1^2	a_1	$+3.1$	**$+4.4$**	-1.9	**$+1.0$**
	$\bar{a}_1$	$+1.1$	-2.9	$+2.0$	-0.4

(a) Illustration of the optimal BG policy $\boldsymbol{\beta}^*$. $V(\boldsymbol{\beta}^*) = (2.0 + 3.6 + 4.4 + 1.0)/4 = \frac{11.0}{4} = 2.75.$

		θ_2^1		θ_2^2	
		a_2	$\bar{a}_2$	a_2	$\bar{a}_2$
θ_1^1	a_1	-0.3	$+0.6$	-0.6	**$+4.0$**
	$\bar{a}_1$	-0.6	**$+2.0$**	-1.3	$+3.6$
θ_1^2	a_1	$+3.1$	**$+4.4$**	-1.9	$+1.0$
	$\bar{a}_1$	$+1.1$	-2.9	**$+2.0$**	-0.4

(b) The optimal complete information policy $\boldsymbol{\Gamma}^*$. $V(\boldsymbol{\Gamma}^*) = (2.0 + 4.0 + 4.4 + 2.0)/4 = \frac{12.4}{4} = 3.1.$

Figure 1: Illustration of the difference between the BG policy $\boldsymbol{\beta}$ and the CI policy $\boldsymbol{\Gamma}$. This example assumes a uniform distribution over joint types.

The value of a joint CI policy $\boldsymbol{\Gamma}$ can also be written as a summation of values for each joint type

$$V(\boldsymbol{\Gamma}) = \sum_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \Pr(\boldsymbol{\theta}) u(\boldsymbol{\theta},\boldsymbol{\Gamma}(\boldsymbol{\theta})) = \sum_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} C_{\boldsymbol{\theta}}(\mathbf{a}_{\boldsymbol{\theta}}^{\boldsymbol{\Gamma}}). \tag{2}$$

The optimal joint CI policy $\boldsymbol{\Gamma}^*$ is much easier to find than $\boldsymbol{\beta}^*$, since it simply specifies to take the joint action that maximizes the contribution for each joint type.

$$\forall_{\boldsymbol{\theta}} \quad \boldsymbol{\Gamma}^*(\boldsymbol{\theta}) = \arg\max_{\mathbf{a}} C_{\boldsymbol{\theta}}(\mathbf{a}). \tag{3}$$

Figure 1 illustrates the difference between the optimal (regular) joint BG policy $\boldsymbol{\beta}^*$ and the CI policy $\boldsymbol{\Gamma}^*$.

3. BRANCH AND BOUND SEARCH

Here we introduce Bayesian game branch and bound policy search, dubbed BaGaBaB.

3.1 Joint Policies as Joint Action Vectors

A CI joint policy $\boldsymbol{\Gamma}$ is equivalent to a vector of joint actions, one for each joint type. For instance, in the example shown in Figure 1 there are four joint types

$$\boldsymbol{\Theta} = \{\langle \theta_1^1,\theta_2^1\rangle , \langle \theta_1^1,\theta_2^2\rangle , \langle \theta_1^2,\theta_2^1\rangle , \langle \theta_1^2,\theta_2^2\rangle\}.$$

If we interpret this set as an ordered list, that we can represent $\boldsymbol{\Gamma}^*$ simply as

$$\boldsymbol{\Gamma}^* = \langle \langle \bar{a}_1,\bar{a}_2\rangle , \langle a_1,\bar{a}_2\rangle , \langle a_1,\bar{a}_2\rangle , \langle \bar{a}_1,a_2\rangle\rangle . \tag{4}$$

Similarly, it is also possible to specify any regular joint BG-policy $\boldsymbol{\beta}$ as such a vector. For instance, $\boldsymbol{\beta}^*$ from Figure 1a can be represented as

$$\boldsymbol{\beta}^* = \langle \langle \bar{a}_1,\bar{a}_2\rangle , \langle \bar{a}_1,\bar{a}_2\rangle , \langle a_1,\bar{a}_2\rangle , \langle a_1,\bar{a}_2\rangle\rangle . \tag{5}$$

We will refer to such vectors representing policies as joint-action vectors (JAVs).

However, there is one big difference between regular and CI joint policies: every JAV of size $|\boldsymbol{\Theta}|$ corresponds to some joint CI policy $\boldsymbol{\Gamma}$. However not every such JAV corresponds to a joint BG policy $\boldsymbol{\beta}$. The reason for this difference is as follows: because the domains of both $\boldsymbol{\Gamma}$ and Γ_i are the same (namely the set of joint types $\boldsymbol{\Theta}$), it is always possible to decompose any $\boldsymbol{\Gamma}$ as specified by a JAV into valid individual policies Γ_i. In contrast, when specifying a $\boldsymbol{\beta}$ by a vector, it may not be possible to decompose it into individual β_i, which means that $\boldsymbol{\beta}$ is not valid.

3.2 Partial Vectors and Heuristic Value

Given the representation of policies using vectors, we define Bayesian game branch and bound (BaGaBaB), a heuristic search algorithm. The basic idea is to create a search tree in which the nodes are partially specified vectors (i.e., joint policies). We can compute an upper bound on the value achievable for any such partially specified vector by computing the maximum value of the CI joint policy that is consistent with it. Since this value is a guaranteed upper bound to the maximum value achievable by a consistent joint BG policy, it is an admissible heuristic.

Each node N in the search tree represents a partially specified JAV and thus a partially specified joint BG-policy. For instance a completely unspecified vector $\langle \cdot, \ldots, \cdot \rangle$ will correspond to the root node. While an internal node N at depth k (root being at depth 0) specifies joint actions for the first k joint types $N = \langle \mathbf{a}_{\boldsymbol{\theta}^1}, \ldots, \mathbf{a}_{\boldsymbol{\theta}^k}, \cdot, \ldots, \cdot \rangle$. The value of a node $V(N)$ is the value of the best joint BG-policy that is consistent with it. Unfortunately, this value is not known in advance, so we need to resort to heuristic values to guide our search.

In order to compute a heuristic value for a node N, we compute the true contribution of the (joint actions for the) k specified joint types and add a heuristic estimate of the maximum contribution for the other joint types. For this heuristic estimate, we use the contribution that the non-specified joint types would bring under complete information. That is, for a some node at depth k we construct a joint CI policy

$$\boldsymbol{\Gamma}_N = \left\langle \mathbf{a}_{\boldsymbol{\theta}^1}, \ldots, \mathbf{a}_{\boldsymbol{\theta}^k}, \mathbf{a}^{\boldsymbol{\Gamma}^*}_{\boldsymbol{\theta}^{k+1}}, \ldots, \mathbf{a}^{\boldsymbol{\Gamma}^*}_{\boldsymbol{\theta}^{|\boldsymbol{\Theta}|}} \right\rangle, \tag{6}$$

that selects the maximizing joint action for the unspecified joint types using (3). The value of $\boldsymbol{\Gamma}_N$, given by (2), then serves as a heuristic for N

$$f(N) \equiv V(\boldsymbol{\Gamma}_N) = g(N) + h(N) \tag{7}$$

with

$$g(N) = C_{\boldsymbol{\theta}^1}(\mathbf{a}_{\boldsymbol{\theta}^1}) + \cdots + C_{\boldsymbol{\theta}^k}(\mathbf{a}_{\boldsymbol{\theta}^k}) \tag{8}$$

the value actually achieved by the actions specified by N, and the heuristic for the remainder:

$$h(N) = C_{\boldsymbol{\theta}^{k+1}}(\mathbf{a}^{\boldsymbol{\Gamma}^*}_{\boldsymbol{\theta}^{k+1}}) + \cdots + C_{\boldsymbol{\theta}^{|\boldsymbol{\Theta}|}}(\mathbf{a}^{\boldsymbol{\Gamma}^*}_{\boldsymbol{\theta}^{|\boldsymbol{\Theta}|}}). \tag{9}$$

Let us continue with the example of Figure 1. We assumed an ordering of joint types:

$$\boldsymbol{\theta}^1 = \langle \theta_1^1, \theta_2^1 \rangle, \boldsymbol{\theta}^2 = \langle \theta_1^1, \theta_2^2 \rangle, \boldsymbol{\theta}^3 = \langle \theta_1^2, \theta_2^1 \rangle, \boldsymbol{\theta}^4 = \langle \theta_1^2, \theta_2^2 \rangle \tag{10}$$

which allowed us to write a joint BG-policy as a vector of joint actions. Let us define the joint actions in this example as follows: $\mathbf{a}^1 = \langle a_1, a_2 \rangle, \mathbf{a}^2 = \langle a_1, \bar{a}_2 \rangle, \mathbf{a}^3 = \langle \bar{a}_1, a_2 \rangle, \mathbf{a}^4 = \langle \bar{a}_1, \bar{a}_2 \rangle$. Now we can consider a node N in the search tree that represents a partially specified joint BG policy $N = \langle \mathbf{a}^3, \cdot, \cdot, \cdot \rangle$ that maps $\boldsymbol{\theta}^1 \to \mathbf{a}^3$. From this node we construct $\boldsymbol{\Gamma}_N = \langle \mathbf{a}^3, \mathbf{a}^2, \mathbf{a}^2, \mathbf{a}^3 \rangle$ because (3) specifies that $\mathbf{a}^2 = \boldsymbol{\Gamma}^*(\boldsymbol{\theta}^2) = \arg\max_{\mathbf{a}} C_{\boldsymbol{\theta}^2}(\mathbf{a})$ is maximizing for $\boldsymbol{\theta}^2$ under complete information. Similarly, $\mathbf{a}^2, \mathbf{a}^3$ are maximizing for $\boldsymbol{\theta}^3, \boldsymbol{\theta}^4$. The heuristic value of N is given by

$$f(N) = (-0.6 + 4.0 + 4.4 + 2.0)/4 = \frac{9.8}{4} = 2.45.$$

Because $h(N)$ is a guaranteed over-estimation as we will show next, search using (7) as its heuristic value is guaranteed to find an optimal solution.

Lemma 1. *The complete information (CI) heuristic yields a guaranteed over-estimation. That is,* (7) *yields an upper-bound on the achievable value:* $V(N) \leq V(\boldsymbol{\Gamma}_N)$.

Proof. In order to show that the heuristic value $f(N)$ is an upper bound, we introduce a mapping also called CI that specifies what joint actions are consistent with a particular node $N = \langle \mathbf{a}_{\boldsymbol{\theta}^1}, \ldots, \mathbf{a}_{\boldsymbol{\theta}^k}, \cdot, \cdot \rangle$. Namely,

$$CI(N, \boldsymbol{\theta}^m) = \begin{cases} \{\mathbf{a}_{\boldsymbol{\theta}^m}\} & \text{if } 1 \leq m \leq k \\ \mathcal{A} & \text{otherwise.} \end{cases} \tag{11}$$

Therefore, if N specifies a joint action for $\boldsymbol{\theta}^m$, then $CI(N, \boldsymbol{\theta}^m)$ is the singleton set containing this action, otherwise it specifies the entire set of joint actions. Thereafter, $f(N)$ can be rewritten as follows:

$$f(N) = V(\boldsymbol{\Gamma}_N) = \sum_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \max_{\mathbf{a} \in CI(N, \boldsymbol{\theta})} C_{\boldsymbol{\theta}}(\mathbf{a}) \tag{12}$$

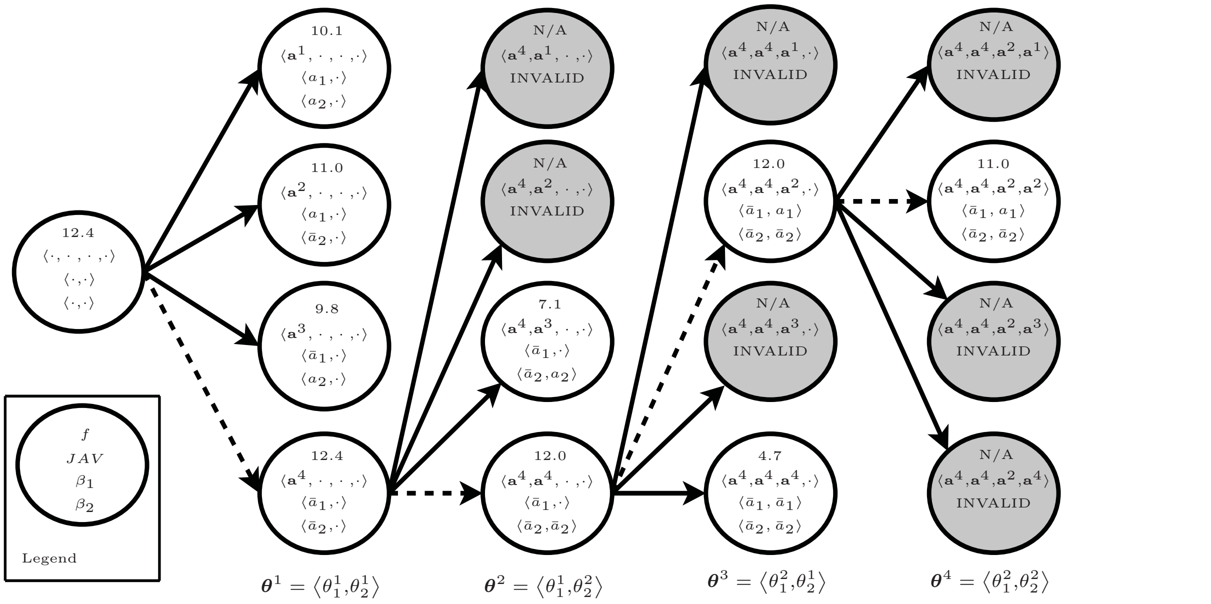
Then, for any vector N' that is obtained from N by adding constraints on $CI(N, \boldsymbol{\theta}^m)$, we are able to prove that $f(N) \geq f(N')$. In other words, value $f(N)$ is an upper bound for all N' that might result from N by further constraining it. This is true since constraining the set $CI(N, \boldsymbol{\theta}^m)$ will simply result in reducing the set of possible actions under max operator $\max_{\mathbf{a} \in CI(N, \boldsymbol{\theta})}$ in Equation (12), and the value can therefore never increase. If N is completely specified, then each set $CI(N, \boldsymbol{\theta}^m)$ contains only a single action, and the upper bound $f(N)$ coincides with the true value of N. □

3.3 Search Tree

The BaGaBaB search we propose is a form of best-first search. This can be seen in Figure 2, our illustration of the search tree for the example in Figure 1. It shows the partial JAVs specified by each node, their heuristic value f and the induced joint policy. To make it easier to relate the heuristic values to the payoffs shown in Figure 1, they have not been weighted by the (uniform) probability of their joint types. That is, the true heuristic value of the root node is given by $\frac{12.4}{4} = 3.1$ (see also the caption of Figure 1b).

Search starts with an open list that has as its sole element the root node that corresponds to the empty JAV $N_{root} = \langle \cdot, \cdot, \cdot, \cdot \rangle$. Its heuristic value is given by the value of the best CI joint BG-policy $V(\boldsymbol{\Gamma}^*)$, since $\boldsymbol{\Gamma}_{N_{root}} = \boldsymbol{\Gamma}^*$. This node is expanded by creating all child nodes $\{N = \langle \mathbf{a}, \cdot, \cdot, \cdot \rangle | \mathbf{a} \in \mathcal{A}\}$, putting them in the open list and computing their heuristic value $f(N)$ through (6) and (7). The root node is now discarded from the open list.

At this point the node with the highest heuristic value is selected to be expanded next. In Figure 2 this is the node $\langle \mathbf{a}^4, \cdot, \cdot, \cdot \rangle$ which has heuristic value 12.4. Expansion of this node leads to just two valid child nodes. E.g., $\langle \mathbf{a}^4, \mathbf{a}^1, \cdot, \cdot \rangle$ is invalid because that would specify both actions $\bar{a}_1$ and a_1 for the same individual type θ_1^1. The invalid nodes are discarded. Again, a next node is chosen to be expanded which is $\langle \mathbf{a}^4, \mathbf{a}^4, \cdot, \cdot \rangle$. This node has a heuristic value of 12.0. Further expansions lead to $\langle \mathbf{a}^4, \mathbf{a}^4, \mathbf{a}^2, \cdot \rangle$ (with $f = 12.0$) and finally $\langle \mathbf{a}^4, \mathbf{a}^4, \mathbf{a}^2, \mathbf{a}^2 \rangle$ with *exact* value $V = 11.0$. This value is exact because at this point the JAV is fully specified. This also means that any nodes N with heuristic value $f \leq 11.0$ can be pruned. In this example, this removes all nodes from



12.4
$\langle \cdot, \cdot, \cdot, \cdot \rangle$
$\langle \cdot,\cdot \rangle$
$\langle \cdot,\cdot \rangle$
10.1
$\langle \mathbf{a}^1, \cdot, \cdot, \cdot \rangle$
$\langle a_1,\cdot \rangle$
$\langle a_2,\cdot \rangle$
11.0
$\langle \mathbf{a}^2, \cdot, \cdot, \cdot \rangle$
$\langle a_1,\cdot \rangle$
$\langle \bar{a}_2,\cdot \rangle$
9.8
$\langle \mathbf{a}^3, \cdot, \cdot, \cdot \rangle$
$\langle \bar{a}_1,\cdot \rangle$
$\langle a_2,\cdot \rangle$
12.4
$\langle \mathbf{a}^4, \cdot, \cdot, \cdot \rangle$
$\langle \bar{a}_1,\cdot \rangle$
$\langle \bar{a}_2,\cdot \rangle$
N/A
$\langle \mathbf{a}^4,\mathbf{a}^1, \cdot, \cdot \rangle$
INVALID
N/A
$\langle \mathbf{a}^4,\mathbf{a}^2, \cdot, \cdot \rangle$
INVALID
7.1
$\langle \mathbf{a}^4,\mathbf{a}^3, \cdot, \cdot \rangle$
$\langle \bar{a}_1,\cdot \rangle$
$\langle \bar{a}_2,a_2 \rangle$
12.0
$\langle \mathbf{a}^4,\mathbf{a}^4, \cdot, \cdot \rangle$
$\langle \bar{a}_1,\cdot \rangle$
$\langle \bar{a}_2,\bar{a}_2 \rangle$
N/A
$\langle \mathbf{a}^4,\mathbf{a}^4,\mathbf{a}^1, \cdot \rangle$
INVALID
12.0
$\langle \mathbf{a}^4,\mathbf{a}^4,\mathbf{a}^2, \cdot \rangle$
$\langle \bar{a}_1,a_1 \rangle$
$\langle \bar{a}_2,\bar{a}_2 \rangle$
N/A
$\langle \mathbf{a}^4,\mathbf{a}^4,\mathbf{a}^3, \cdot \rangle$
INVALID
4.7
$\langle \mathbf{a}^4,\mathbf{a}^4,\mathbf{a}^4, \cdot \rangle$
$\langle \bar{a}_1,\bar{a}_1 \rangle$
$\langle \bar{a}_2,\bar{a}_2 \rangle$
N/A
$\langle \mathbf{a}^4,\mathbf{a}^4,\mathbf{a}^2,\mathbf{a}^1 \rangle$
INVALID
11.0
$\langle \mathbf{a}^4,\mathbf{a}^4,\mathbf{a}^2,\mathbf{a}^2 \rangle$
$\langle \bar{a}_1,a_1 \rangle$
$\langle \bar{a}_2,\bar{a}_2 \rangle$
N/A
$\langle \mathbf{a}^4,\mathbf{a}^4,\mathbf{a}^2,\mathbf{a}^3 \rangle$
INVALID
N/A
$\langle \mathbf{a}^4,\mathbf{a}^4,\mathbf{a}^2,\mathbf{a}^4 \rangle$
INVALID
f
JAV
β_1
β_2
Legend
$\boldsymbol{\theta}^1 = \langle \theta_1^1,\theta_2^1 \rangle$
$\boldsymbol{\theta}^2 = \langle \theta_1^1,\theta_2^2 \rangle$
$\boldsymbol{\theta}^3 = \langle \theta_1^2,\theta_2^1 \rangle$
$\boldsymbol{\theta}^4 = \langle \theta_1^2,\theta_2^2 \rangle$

Figure 2: BaGaBaB search tree, where dashed arrows indicate the search path taken. Each node shows its heuristic value f, the partially specified joint action vector JAV and the induced BG-policies β_1, β_2. Heuristic values should be divided by 4 to account for the (uniform) probabilities of joint types. The shaded nodes are invalid.

the open list, which means that we have found the optimal joint policy.

It is not necessary to expand the last node, since the joint BG policy $\boldsymbol{\beta}$ is already fully specified at $\langle \mathbf{a}^4,\mathbf{a}^4,\mathbf{a}^2,\cdot \rangle$. Expansion of the last node merely serves to evaluate the exact value. In general, it is undesirable to try and expand multiple nodes only for this purpose; when $\boldsymbol{\beta}$ is fully specified an evaluation of its value should be performed immediately.

4. IMPROVING EFFICIENCY

Here we present some ways by which the efficiency of BaGaBaB may be improved.

4.1 Avoiding Expansion of Invalid Nodes

Expansion of invalid nodes can be easily prevented. For instance, assume we want to expand $N = \langle \mathbf{a}^4, \cdot, \cdot, \cdot \rangle$ which has induced joint policy $\boldsymbol{\beta} = \langle \langle \bar{a}_1,\cdot \rangle, \langle \bar{a}_2,\cdot \rangle \rangle$. Since we are going to perform an assignment to $\boldsymbol{\theta}^2 = \langle \theta_1^1,\theta_2^2 \rangle$, we can simply look up any actions that are already specified. That is, rather than expanding nodes for all $\mathcal{A}$, we can immediately construct a smaller set $Cons$ of joint actions that are consistent. This set is easily constructed by using the same notation as for partially specified policies:

$$\begin{aligned} Cons(N,\boldsymbol{\theta}^2) &= \{\boldsymbol{\beta}(\boldsymbol{\theta}^2)\} = \{\langle \beta_1(\theta_1^1),\beta_2(\theta_2^2) \rangle\} \\ &= \{\langle \bar{a}_1,\cdot \rangle\} = \{\langle \bar{a}_1,a_2 \rangle, \langle \bar{a}_1,\bar{a}_2 \rangle\} = \{\mathbf{a}^3,\mathbf{a}^4\} \quad (13) \end{aligned}$$

That is, the '$\cdot$' specified by the induced policy $\beta_2(\theta_2^2)$ works as a wildcard, while $\beta_1(\theta_1^1)$ is specified by the induced policy and thus specifies one particular action ($\bar{a}_1$).

4.2 Improved Heuristic

The CI heuristic is an upper bound to the true value, but is not very tight. Here we discuss a second heuristic that takes into account the joint actions that have been specified already, making sure only to select consistent joint actions. To achieve this, we propose to employ a *consistent complete information (CCI)* joint policy $\boldsymbol{\Delta}$, rather than using the CI joint policy for the unspecified joint actions.

In particular, we define $\boldsymbol{\Delta}_N$ for a node N using $Cons(N,\boldsymbol{\theta})$ as defined by (13):

$$\forall_N \quad \boldsymbol{\Delta}_N(\boldsymbol{\theta}) \equiv \operatorname*{arg\,max}_{\mathbf{a} \in Cons(N,\boldsymbol{\theta})} C_{\boldsymbol{\theta}}(\mathbf{a}). \tag{14}$$

Definition 2. The *consistent complete information (CCI)* heuristic for a node N is defined as

$$f(N) = V(\boldsymbol{\Delta}_N) = \sum_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \max_{\mathbf{a} \in Cons(N,\boldsymbol{\theta})} C_{\boldsymbol{\theta}}(\mathbf{a}). \tag{15}$$

As an example, again consider node $N = \langle \mathbf{a}^3, \cdot, \cdot, \cdot \rangle$. When taking into account the specified action $\mathbf{a}^3$ for $\boldsymbol{\theta}^1$, we see that the only consistent choices for $\boldsymbol{\theta}^2$ are $\mathbf{a}^3$,$\mathbf{a}^4$. As such the CCI joint policy specifies

$$\mathbf{a}^3 = \boldsymbol{\Delta}_N(\boldsymbol{\theta}^2) = \operatorname*{arg\,max}_{\mathbf{a} \in \{\mathbf{a}^3,\mathbf{a}^4\}} C_{\boldsymbol{\theta}^2}(\mathbf{a}),$$

and the full CCI joint policy for this node is given by $\boldsymbol{\Delta}_N = \langle \mathbf{a}^3,\mathbf{a}^3,\mathbf{a}^2,\mathbf{a}^3 \rangle$.

Lemma 2. *The CCI heuristic* (15) *yields an upper-bound on the achievable value:* $\forall_N \quad V(N) \leq V(\boldsymbol{\Delta}_N)$.

Proof. We make the same argument as before, but now using the $Cons$ mapping. □

Corollary 1. *The CCI heuristic is tighter than the CI heuristic:* $\forall_N \quad V(N) \leq V(\boldsymbol{\Delta}_N) \leq V(\boldsymbol{\Gamma}_N)$.

Proof. We only need to discuss $V(\boldsymbol{\Delta}_N) \leq V(\boldsymbol{\Gamma}_N)$, as the first inequality follows from the lemmas. For each $\forall_{N,\boldsymbol{\theta}}$ $Cons(N,\boldsymbol{\theta}) \subseteq CI(N,\boldsymbol{\theta})$. Therefore we can make the same argument with respect to the constraints as before. □

The CCI heuristic is tighter and thus may allow for more pruning. However, it involves a higher computation overhead: where the CI heuristic of a node can be computed in constant time, computation of the CCI heuristic has cost $O(|\mathbf{\Theta}|\,|\mathcal{A}|)$, because we need to loop over all remaining joint types and select the maximizing consistent joint action. So clearly there is a trade-off between the CI and CCI heuristics. The worst-case cost κ of expanding a node using the CCI heuristic into its $O(|\mathcal{A}|)$ children is given by

$$\kappa = O(|\mathbf{\Theta}|\,|\mathcal{A}|^2) = O\left(|\Theta_*|^n\,|\mathcal{A}_*|^{2n}\right). \qquad (16)$$

where $\mathcal{A}_*$ and Θ_* denote the largest individual set of actions and observations.

4.3 Ordering of Joint Types

Note that the reason that Figure 2 first expands the policy of agent 2 into a full policy (at depth 2) lies in the ordering of joint types. We assumed the ordering is given by (10) which means that the first 2 joint actions both will specify an individual action for agent 1's first type θ_1^1. The ordering

$$(\langle\theta_1^1,\theta_2^1\rangle, \langle\theta_1^2,\theta_2^1\rangle, \langle\theta_1^1,\theta_2^2\rangle, \langle\theta_1^2,\theta_2^2\rangle),$$

would lead to first expanding agent 1's policy to a full one, while

$$(\langle\theta_1^1,\theta_2^1\rangle, \langle\theta_1^2,\theta_2^2\rangle, \langle\theta_1^1,\theta_2^2\rangle, \langle\theta_1^2,\theta_2^1\rangle),$$

will lead to simultaneous expansion into full policies (by assignment of just 2 joint actions). For any ordering of joint types, it is easy to determine at what point a full $\boldsymbol{\beta}$ will be specified and therefore the list of joint types can be truncated at that point. Here we discuss some different ways of selecting an ordering for the joint types.

Numeric Ordering

Any tuple of types can be interpreted as a number. For instance when both agents have k individual types $\langle\theta_1^5,\theta_2^3\rangle$ can be interpreted as '53' in base k. (Generalization to different numbers of types per agent is trivial.) Now we can simply order the joint types using the numerical value to which they correspond. This is the ordering expressed by (10) that is used in Figure 2.

Basis Joint Types

The shortest possible list can be constructed by selecting an ordering of joint types such that the first joint types specify different components for each agent. For instance, we could specify $\{\langle\theta_1^1,\theta_2^1\rangle, \langle\theta_1^2,\theta_2^2\rangle\}$ as the basis joint types. Assignment of a joint action to both these basis joint types can be done without considering any conflicts and results in a fully specified joint policy.

This highlights one feature of this ordering: it creates the shallowest possible search tree. However, it also creates the search tree with the highest possible branching factor (exactly because all joint actions are valid). Exactly how these two features trade off is hard to predict.

Simple Heuristic Orderings

Another option is to find some other heuristic ordering of the joint types. That is, we compute some value $H(\boldsymbol{\theta})$ for each joint type $\boldsymbol{\theta}$ and then order them using this value. We propose a few such heuristics:

$$\begin{aligned}
\text{probability:} \quad & H(\boldsymbol{\theta}) = \Pr(\boldsymbol{\theta}) \\
\text{maximum contribution:} \quad & H(\boldsymbol{\theta}) = \max_{\mathbf{a}} C_{\boldsymbol{\theta}}(\mathbf{a}) \\
\text{minimum contribution:} \quad & H(\boldsymbol{\theta}) = \min_{\mathbf{a}} C_{\boldsymbol{\theta}}(\mathbf{a}) \\
\text{max. contr. difference:} \quad & H(\boldsymbol{\theta}) = \max_{\mathbf{a}} C_{\boldsymbol{\theta}}(\mathbf{a}) - \min_{\mathbf{a}} C_{\boldsymbol{\theta}}(\mathbf{a})
\end{aligned}$$

The motivation for these heuristics is quite straightforward. For instance, by sorting joint types in descending order according to their probability will put joint types with large probability (that have a high contribution) early in the search tree. The 'contribution' heuristics explicitly take into account the payoffs of particular joint actions. That is, sorting by maximum contribution puts decisions about potential high utility joint actions early in the tree, while (ascending) sorting by minimum contribution tries to avoid high penalties early in the search. The maximum contribution difference heuristic tries to put decisions that potentially have the most impact early in the tree. The heuristic orderings (as well as numerical orderings) may specify certain joint types that do not result in the specification of any new action, which would result in expanding a useless node. Therefore, after we find a heuristic ordering, we check and remove any useless joint types.

4.4 Reducing Space Complexity

Although the tree in Figure 2 shows the implied joint policy at each node, there is no reason to actually store this information. Rather, the implied policy can be efficiently reconstructed from the JAV when a node is selected. Moreover the JAVs in the tree can be efficiently stored using a pointer mechanism. Therefore, there is no need to store $\langle\mathbf{a}^4,\mathbf{a}^4,\mathbf{a}^2,\cdot\rangle$, and $\langle ptrToParent,\mathbf{a}^2\rangle$ can be stored instead.

If space becomes a problem despite this compact representation of nodes, other approaches could be used. These include other memory-bounded seach strategies (for instance, recursive depth-first search), or applying weights to discount the heuristic [11]. We leave consideration of these alterations to future work.

5. BGS FOR SEQUENTIAL DECISIONS

As mentioned above, Bayesian Games and DEC-POMDPs are related. First, we give a summary of the DEC-POMDP model. Due to lack of space this is kept very concise, for further reading we refer to [9, 6]. Next, we show how BGs appear in the two main approaches to solving DEC-POMDPs, and thus how improvements in BG solution methods may transfer to DEC-POMDPs.

5.1 DEC-POMDPs

A DEC-POMDP is a model for sequential decision making for a team of n cooperative agents in a stochastic, partially observable environment. At any time this environment is in some state $s \in \mathcal{S}$ out of a set of possible states. At each time step, or stage t, the environment is at some state s^t and the agents take a joint action $\mathbf{a}$. As a result, the agents accumulate reward $R(s^t,\mathbf{a})$, the state changes (stochastically) to some next state s^{t+1} and the agents receive a joint observation $\mathbf{o}$, from which each agent i observes only its own component o_i. The goal in a DEC-POMDP is to find an optimal joint policy $\boldsymbol{\pi} = \langle\pi_1,\ldots,\pi_n\rangle$, where $\pi_i = (\delta_i^1,\ldots,\delta_i^{h-1})$

specifies a decision rule δ_i^t for all stages t, that map possible histories of observations $\vec{o}_i^{\,t} = (o_i^1, \ldots, o_i^t)$ to actions: $\pi_i(\vec{o}_i^{\,t}) = \delta_i^t(\vec{o}_i^{\,t}) = a_i$.

5.2 Forward Perspective

A DEC-POMDP can be modeled by a sequence of identical payoff BGs, one for each stage t. In this BG B^t, the set of agents and their actions are the same as in the DEC-POMDP [6]. At a particular stage t, the private information an agent i has is its observation history (OH) $\vec{o}_i^{\,t}$. As such, the types of each agent i are defined by the possible OHs it can have: $\theta_i \equiv \vec{o}_i^{\,t}$. Since a BG-policy maps types to actions, a BG-policy in B^t maps OHs to actions and therefore corresponds to a decision rule of the DEC-POMDP: $\beta_i(\theta_i) \equiv \delta_i^t(\vec{o}_i^{\,t})$. Similarly, joint BG-policies $\boldsymbol{\beta}$ correspond to joint decision rules $\boldsymbol{\delta}^t$.

The probabilities of joint types $\Pr(\boldsymbol{\theta})$ in B^t correspond to the probabilities of joint OHs. These probabilities are available if we assume that $(\boldsymbol{\delta}^0, \ldots, \boldsymbol{\delta}^{t-1})$, the joint policy up to stage t, is available. This is the case if we pass 'forward' through time: we solve BGs for stages $0, 1, \ldots, h-1$ in subsequent order. Finally, to wrap up the description of B^t, the payoff function u should be defined. In principle it is possible to compute an optimal Q-value function for the DEC-POMDP and use this as the payoff function, but more often heuristic payoffs function are used [6]. If this heuristic is admissible (i.e., a guaranteed over-estimate) an A*-like search over partially specified policies can be employed to find optimal solutions.

5.3 Backward Perspective

The backward perspective of DEC-POMDPs is given by the dynamic programming (DP) algorithm [5] and its extensions. In this perspective, a policy π_i is seen as a tree with nodes that specify actions a_i and edges labeled with observations o_i, such that each node corresponds to an observation history (the path from root to the node).

Rather than constructing such policies at once, DP seeks to construct them incrementally. DP starts with a set $\mathcal{Q}_i^{\tau=1}$ of $\tau = 1$ 'time-steps-to-go' *sub-tree policies* for each agent i (such a sub-tree policy $q_i^{\tau=1} \in \mathcal{Q}_i^{\tau=1}$ corresponds to an action that may be selected for the last stage). Now DP proceeds to, for all agents i, construct sets $\mathcal{Q}_i^{\tau=k+1}$ from $\mathcal{Q}_i^{\tau=k}$, pruning any $q_i^{\tau=k+1}$ that are dominated over $\Delta(\mathcal{S} \times \mathcal{Q}_{\neq i}^{\tau=k+1})$, the simplex over states and policies of other agents. Since the number of sub-tree policies that are non-dominated tends to be very large, point-based DP (PBDP) methods [10, 8, 3] have been introduced. In these methods, BGs appear when backing up policy trees.

PBDP methods work by sampling a set of belief points $\boldsymbol{b}$ (which are distributions over states). For each sampled $\boldsymbol{b}^{\tau=k}$ the maximizing k-steps-to-go joint subtree policy $\boldsymbol{q}^{\tau=k} = \langle q_1^{\tau=k}, \ldots, q_n^{\tau=k} \rangle$ is computed and the components are put in the sets $\mathcal{Q}_i^{\tau=k}$ of useful sub-tree policies. Computation of $\boldsymbol{q}^{\tau=k}$ is done by finding the best 'completion' $C_{\boldsymbol{b}\mathbf{a}}$ for each joint action $\mathbf{a}$. And such a completion $C_{\boldsymbol{b}\mathbf{a}} = \langle C_{\boldsymbol{b}\mathbf{a},1}, \ldots, C_{\boldsymbol{b}\mathbf{a},n} \rangle$ specifies a mapping for all agents from individual observations to individual $(k-1)$-steps-to-go subtrees.

$$C_{\boldsymbol{b}\mathbf{a},i} : \mathcal{O}_i \rightarrow \mathcal{Q}_i^{\tau=k-1}$$

denotes the completion function for agent i for belief point $\boldsymbol{b}^{\tau=k}$ and joint action $\mathbf{a}$.

Finding the best completion $C^*_{\boldsymbol{b}\mathbf{a}}$ corresponds to solving a BG. In particular we have the following correspondences. A type corresponds to an observation in the DEC-POMDP: $\theta_i \equiv o_i$ and a BG-action corresponds to selecting a subtree $a_i \equiv q_i^{\tau=k-1}$. Consequently completions correspond to BG-policies:

$$a_i = \beta_i(\theta_i) \equiv C_{\boldsymbol{b}\mathbf{a},i}(o_i) = q_i^{\tau=k-1}.$$

In fact, point-based incremental pruning (PBIP) [3] uses branch and bound to find the best $C_{\boldsymbol{b}\mathbf{a}}$. It proceeds by expanding non-specified basis joint types following a depth-first fashion using a CI heuristic only for pruning, and specifies a joint action for each joint type in the best-first fashion. BaGaBaB can be seen as a generalization to BGs, which also makes explicit the ordering of joint types and considers different orderings. Moreover, BaGaBaB is based on best-first (A^*) search and it provides tighter upper bound though the CCI heuristic resulting in a more aggressive pruning strategy.

6. EXPERIMENTS

We performed an empirical evaluation of BaGaBaB to test its performance in a number of scenarios. We compared Brute Force Search (BFS) with BaGaBaB using the Numeric Ordering, the Basis Joint Types, as well as the heuristic orderings. We noticed that the heuristic orderings give similar results, hence we only report on one of them, the Maximum Contribution Difference. Unless noted otherwise, we employ the proposed Consistent Complete Information heuristic.

Experiments were run on a dual-core processor running at 2.13GHz, with 2Gb of RAM and a Linux operating system. Processes were limited to 1Gb of memory usage, and BFS was given a deadline. If the solution was not feasible within the deadline, we report approximate timing results. Due to the straightforward nature of the BFS algorithm (looping over all possible joint policies), timing results can be easily and reliably be extrapolated based on the number of already evaluated joint policies. As all methods used are optimal, we only report on computation time.

6.1 Random Bayesian Games

First, we tested our proposed method on randomly generated BGs of different sizes. These BGs were generated by drawing the utilities $u(\boldsymbol{\theta}, \mathbf{a})$ from a uniform distribution over $[-10, +10]$. The probabilities $\Pr(\boldsymbol{\theta})$ are drawn from $[0,1]$ and then normalized. This results in a relatively uniform distribution over joint types and the expected values of different joint policies typically lie closely together. Because of this limited amount of structure in the random BGs, we expect them to be hard to solve for heuristic search methods. Table 1 shows some statistics for several of the randomly generated BGs considered.

The reported results of the methods are averaged over the same 100 randomly generated BGs, and BFS had a deadline of 36s per BG. Figure 3 shows the results we obtained for these random BGs. Figure 3a shows results for a varying number of actions with $n = 2$, $|\Theta_i| = 3$ (like those in Table 1a). It shows that BaGaBaB significantly outperforms BFS for all problems with $|\mathcal{A}_i| > 5$, the difference being up to three orders of magnitude. The figure also shows that BaGaBaB scales quite well with the number of actions.

Figure 3b shows the results for varying number of types (with fixed parameters as in Table 1b). Again, BaGaBaB

$\lvert\mathcal{A}_i\rvert$	2	4	6	8
$\lvert\boldsymbol{\beta}\rvert$	6.4e01	4.1e03	4.7e04	2.6e05
$\lvert\boldsymbol{\Theta}\rvert$	9	9	9	9
κ	1.4e02	2.3e03	1.2e04	3.7e04

(a) Varying $|\mathcal{A}_i|$. $n = 2$, $|\Theta_i| = 3$.

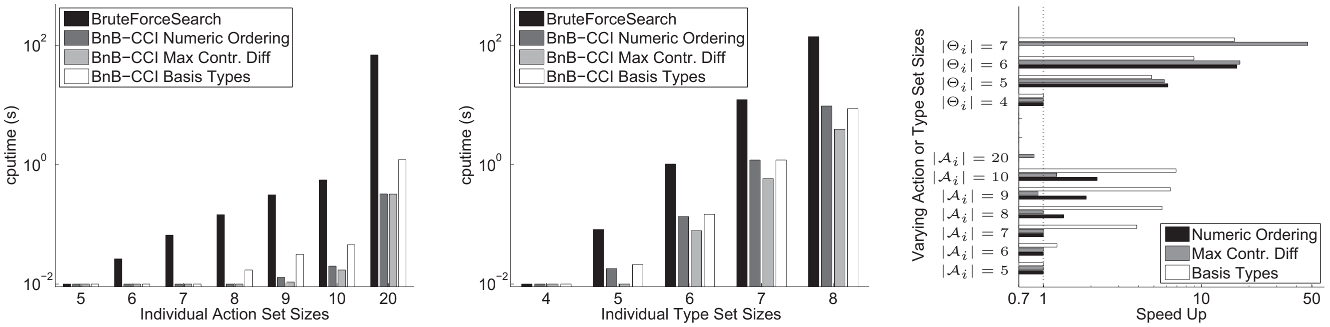
$\lvert\Theta_i\rvert$	2	4	6	8
$\lvert\boldsymbol{\beta}\rvert$	8.1e01	6.6e03	5.3e05	4.3e07
$\lvert\boldsymbol{\Theta}\rvert$	4	16	36	64
κ	3.2e02	1.3e03	2.9e03	5.2e03

(b) Varying $|\Theta_i|$. $n = 2$, $|\mathcal{A}_i| = 3$.

n	2	4	6	8
$\lvert\boldsymbol{\beta}\rvert$	7.3e02	5.3e05	3.9e08	2.8e11
$\lvert\boldsymbol{\Theta}\rvert$	9	81	730	6600
κ	7.3e02	5.3e05	3.9e08	2.8e11

(c) Varying n. $|\mathcal{A}_i| = 3$, $|\Theta_i| = 3$.

Table 1: Some statistics for different BG sizes. Shown are the number of joint policies $|\boldsymbol{\beta}|$, the number of joint types $|\boldsymbol{\Theta}|$ and the worst case complexity of expanding a node κ.



(a) Varying number of actions.

(b) Varying number of types.

(c) Speedup of CCI over CI.

Figure 3: Results for random BGs.

outperforms BFS, but by a smaller margin. Also it scales relatively poorly with respect to the number of types. This may seem surprising, since the worst-case time complexity of expanding a node grows less quickly with the number of types. However, analysis revealed that for $|\mathcal{A}_i| = 3$ and $|\Theta_i| = 8$ many more nodes are expanded then for $|\mathcal{A}_i| = 8$ and $|\Theta_i| = 3$. This has two reasons. First, as illustrated by Table 1b, the number of joint BG policies grows more quickly with respect to the number of types. This however, should hamper the performance of BFS equally. Second, the number of *joint* types negatively affects the tightness of the heuristic: For each of the unspecified joint types we make an over-estimation, so one could expect the total over-estimation to be proportional to the number of joint types.

We also compared the CCI heuristic vs. the vanilla CI version. As the latter is less tight, BaGaBaB with CI can prune fewer nodes, resulting in higher computation times (in general) and higher memory requirements. To see whether the additional overhead of re-computing the CCI heuristic is worth the effort, we ran BaGaBaB with CI for all the random BGs mentioned above. Figure 3c (note the log-scale on the x-axis) shows the speedup of using CCI vs. CI, defined as the quotient of both computation times. We see that in general the speedup is higher than 1, indicating the slight computational overhead of CCI results in faster search (less nodes to be expanded). Furthermore, for large type spaces, the speedup grows, and allows us to solve larger problems within the defined memory limits.

6.2 BGs from DEC-POMDPs

We also tested the performance on Bayesian games resulting from sequential decision problems, DEC-POMDPs in particular. Note that although the forward and backward perspective both generate BGs, the shape of these BGs is substantially different. The forward approach generates BGs with many types (because the number of histories grows exponentially with t), while the backward approach generates BGs with many actions (because the number of sub-tree policies can grow double exponentially with τ).

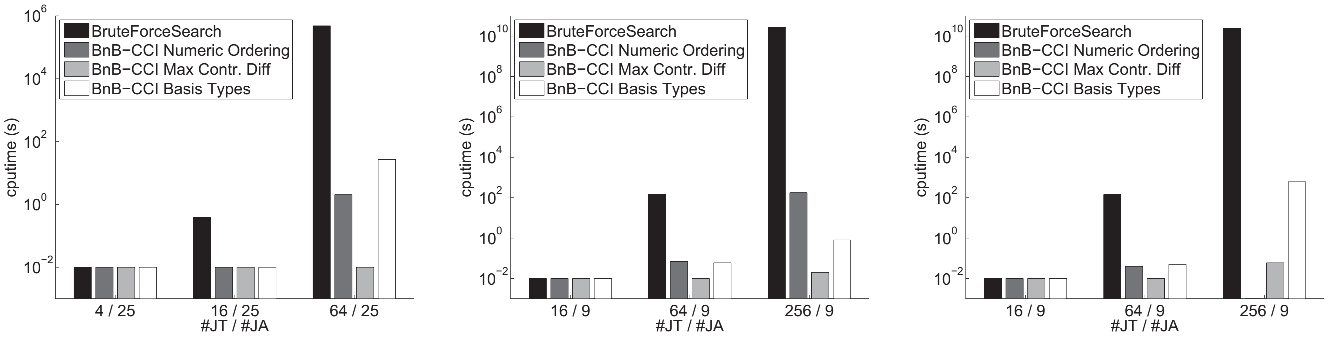
We collected a set of BGs encountered when running Forward Sweep Policy Search as implemented by GMAA* [6], for several standard benchmark problems, whose descriptions can be found in [6] as well. Figure 4 shows the results, demonstrating dramatic speedups of up to 12 orders of magnitude for the larger BGs. In particular the Max Contribution Difference performs very well. This confirms our hypothesis that randomly generated BGs are hard to solve, but that heuristic search methods can exploit the structure inherent to non-random problems. Here joint type distributions have peaks instead of being flat, and the reward structure is often very skewed. If we quickly find a high-payoff BG-policy and many others have a lower upper bound, we can prune many candidates.

We also performed a preliminary investigation into BGs resulting from Backward Perspective DEC-POMDP methods, PBIP in particular [3]. Figure 5 shows results for the Cooperative Box Pushing problem, gathering BGs solved by PBIP using parameter $maxTrees = 3$. We see that we can obtain speedups of about an order of magnitude, which is promising when considering that the number of joint actions is still low (due to the low value for *maxTrees).*

7. CONCLUSIONS AND DISCUSSION

We considered solving Bayesian games (BGs) with identical payoffs, which important models for single-shot interactions such as coalition formation as well as sequential models for cooperative teams of agents, such as DEC-POMDPs. We showed how BGs can not only be used to model DEC-POMDP solutions from the forward perspective, but also from the backward perspective. This new viewpoint could lead to new insight on overcoming the bottlenecks for current DEC-POMDP dynamic programming algorithms.

Bayesian games occur in many diverse scenarios, but meth-



(a) GridSmall problem.

(b) FireFighting problem.

(c) Recycling Robots problem.

Figure 4: Results for Forward Perspective BGs. A missing result is due to violation of the imposed memory limit.

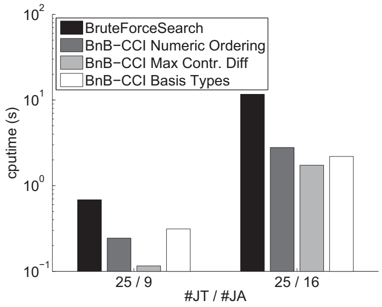


Figure 5: Backward perspective: Box Pushing problem.

ods to solve them efficiently have received surprisingly little attention. For this reason, we proposed BAGABAB, a branch and bound solution method for identical payoff Bayesian games. This algorithm is able to make use of the structure present in identical payoff BGs to solve them more efficiently. We also provide several extensions to improve the performance of the algorithm.

To test its performance, we empirically tested the proposed algorithm on randomly generated BGs as well as on BGs encountered while solving DEC-POMDPs. When compared with random BGs, BAGABAB shows a marked increase in performance over brute force search. In some cases, it ran up to 3 orders of magnitude faster. The approach scales especially well with respect to the number of actions. For BGs encountered in real problems, more structure is often present. This is supported by an evaluation on BGs encountered in the solution of DEC-POMDPs. For these problems, we encountered speedups of over 10 orders of magnitude. This shows the effectiveness of using a specialized approach to solving Bayesian games, especially in more realistic scenarios.

In the future, we plan to improve the algorithm as well as using it to build a full DEC-POMDP solver. Improvements we expect to incorporate include heuristics such as those that remove actions that are dominated in a given situation as well as exploring the anytime performance of the search. For solving DEC-POMDPs, we believe improved performance could be realized in both top-down (forward perspective) and bottom-up (backwork perspective) solvers by utilizing BAGABAB at each step of the approach.

Acknowledgements

This research is part of the Interactive Collaborative Information Systems (ICIS) project, supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024. This work was funded by Fundação para a Ciência e a Tecnologia (ISR/IST pluriannual funding) through the PIDDAC Program funds and was supported by project PTDC/EEA-ACR/73266/2006. This work was supported in part by the Air Force Office of Scientific Research under Grant No. FA9550-08-1-0181 and by the National Science Foundation under Grant No. IIS-0812149.

8. REFERENCES

[1] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *UAI*, 2000.

[2] G. Chalkiadakis and C. Boutilier. Sequential decision making in repeated coalition formation under uncertainty. In *AAMAS*, 2008.

[3] J. S. Dibangoye, A.-I. Mouaddib, and B. Chai-draa. Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs. In *AAMAS*, 2009.

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

[5] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, 2004.

[6] 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, 2008.

[7] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

[8] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *IJCAI*, 2007.

[9] S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2), 2008.

[10] D. Szer and F. Charpillet. Point-based dynamic programming for DEC-POMDPs. In *AAAI*, 2006.

[11] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *UAI*, 2005.