Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-009-9103-z

Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs

Christopher Amato · Daniel S. Bernstein · Shlomo Zilberstein

Abstract POMDPs and their decentralized multiagent counterparts, DEC-POMDPs, offer a rich framework for sequential decision making under uncertainty. Their high computational complexity, however, presents an important research challenge. One way to address the intractable memory requirements of current algorithms is based on representing agent policies as finite-state controllers. Using this representation, we propose a new approach that formulates the problem as a nonlinear program, which defines an optimal policy of a desired size for each agent. This new formulation allows a wide range of powerful nonlinear programming algorithms to be used to solve POMDPs and DEC-POMDPs. Although solving the NLP optimally is often intractable, the results we obtain using an off-the-shelf optimization method are competitive with state-of-the-art POMDP algorithms and outperform state-of-the-art DEC-POMDP algorithms. Our approach is easy to implement and it opens up promising research directions for solving POMDPs and DEC-POMDPs using nonlinear programming methods.

Keywords Decision theory · Multiagent systems · Planning under uncertainty · POMDPs · DEC-POMDPs

1 Introduction

Developing effective frameworks for reasoning under uncertainty is a thriving research area in artificial intelligence. Rapid progress in recent years has contributed to the development of several new solution techniques. Nevertheless, improving the scalability of existing algorithms is still an important challenge. When a single decision maker is considered, the partially

C. Amato (✉) · D. S. Bernstein · S. Zilberstein
Department of Computer Science, University of Massachusetts, Amherst, MA 01003, USA
e-mail: camato@cs.umass.edu

D. S. Bernstein
e-mail: bern@cs.umass.edu

S. Zilberstein
e-mail: shlomo@cs.umass.edu

Published online: 25 August 2009

observable Markov decision process (POMDP) offers a powerful and widely used framework. The POMDP model offers a rich language to describe single agent planning in domains that involve stochastic actions, noisy observations, and a variety of possible objective functions. POMDP applications include robot control [26], medical diagnosis [13] and machine maintenance [8]. In the area of robot control, partial observability is used to model sensors that provide uncertain and incomplete information about the state of the environment. In a medical setting, the internal state of the patient is often not known with certainty. The machine maintenance problem—one of the earliest application areas of POMDPs—seeks to find a cost-effective strategy for inspection and replacement of parts in a domain where partial information about the internal state is obtained by inspecting the manufactured products. Numerous other POMDP applications are surveyed in [6].

Developing effective algorithms for POMDPs is an active research area that has seen significant progress [7,11,15,16,18,21–23,28,29,31]. Despite this progress, it is generally accepted that exact solution techniques are limited to toy problems due to high memory requirements. Consequently, approximation algorithms are often necessary in practice. Approximation methods perform well in many problem domains, but have some known disadvantages. For instance, point-based methods [15,21,28,29,31] perform better with a small reachable belief space. Other algorithms rely on local search methods based on gradient ascent (GA) [18] and linear programming [23] or require fine tuning of heuristic parameters to produce high-valued solutions [22].

Solving problems in which a group of agents must operate collaboratively in some environment based solely on local information is also an important challenge. When the agents must rely on uncertain partial knowledge about the environment, this is particularly difficult. This problem can be modeled as a decentralized partially observable Markov decision process (DEC-POMDP), which is a multiagent extension of the POMDP framework. In a DEC-POMDP each agent must cope with uncertainty about the other agents as well as imperfect information of the system state. The agents seek to optimize a shared objective function using solely local information in order to act.

Many real world applications in cooperative multiagent systems can be modeled as DEC-POMDPs. Some examples include distributed robot control [2,9] and networking problems [3,12]. In multiagent domains, robot sensors not only provide uncertain and incomplete information about their own state, but also about the location of the other robots. This lack of information leads to different perspectives on the state of the environment and complicates the planning process. Similarly, in a decentralized network, each node must make decisions about when and where to send packets without full awareness of the knowledge and actions of nodes in the rest of the network. Other applications of DEC-POMDPs include e-commerce and coordination of space exploration systems.

Although there has been some recent work on exact and approximate algorithms for DEC-POMDPs [3,12,17,19,20,24,33,34], only two algorithms [3,33] are able to find solutions for the general infinite-horizon discounted case. Another algorithm can solve a restricted subclass of infinite-horizon DEC-POMDPs, using an average reward formulation rather than the standard discounted objective [20]. The infinite-horizon problem is particularly suitable for such domains as networking and some robot control problems, where the agents operate continuously. The two general algorithms for infinite-horizon problems seek to optimize fixed-size finite-state controllers. These approaches perform well in some problems, but Szer and Charpillet's algorithm [33] is limited to very small solutions because of high memory and running time requirements. Bernstein et al.'s method [3] is more scalable, but it often produces low-quality results. It is worth noting that many POMDP algorithms cannot be easily extended to apply to DEC-POMDPs. One reason for this is that the decentralized nature of the DEC-POMDP framework results in a lack of a shared belief state, making it impossible to properly estimate the state of the system. Thus, a DEC-POMDP cannot be formulated as a continuous state MDP and algorithms that use this formulation do not generalize.

In this paper, we explore a new solution technique for POMDPs and DEC-POMDPs that optimizes fixed-size controllers. Our approach formulates the optimal memory-bounded solution as a nonlinear program (NLP), and exploits existing nonlinear optimization techniques to solve the problem. Nonlinear programming is an active field of research that has produced a wide range of techniques that can efficiently solve a variety of large problems [5]. In the POMDP case, parameters are optimized for a fixed-size controller which produces the policy for that problem. In the DEC-POMDP case, a set of fixed-size independent controllers is optimized, which when combined, produce the policy for the DEC-POMDP. The formulation provides a new framework for which algorithms can be developed. We present an overview of how to solve these problems optimally, but as this would often be intractable in practice, we also evaluate an effective approximation technique using standard NLP solvers. Our approach facilitates scalability as it offers a tradeoff between solution quality and the available resources. That is, for a given amount of memory, we can search for a controller that is optimal for that size. Large controllers may be needed to provide a near optimal solution in some problems, but our experiments suggest that smaller controllers produce high quality solutions in a wide array of problems.

In order to increase the solution quality of our memory-bounded technique for DEC-POMDPs, we examine the benefits of introducing a *correlation device*, which is a shared source of randomness. This allows a set of independent controllers to be correlated in order to produce higher values, without sharing any additional local information. Correlation adds another mechanism in an effort to gain the most possible value with a fixed amount of space. This has been shown to be useful in order to increase value of fixed-size controllers [3] and we show that is also useful when combined with our NLP approach.

The rest of the paper is organized as follows. We first give an overview of the POMDP and DEC-POMDP models and explain how infinite-horizon solutions can be represented as stochastic controllers. We discuss some previous work on each model and then show how to define optimal controllers using a nonlinear program. We then provide experimental results using a generic off-the-shelf NLP solver in various POMDP and DEC-POMDP benchmark domains. For POMDP problems, our approach is competitive with other state-of-the-art solution methods in terms of solution quality and running time, but it generally produces higher quality when given less memory. When applied to DEC-POMDPs, our approach proves to be more valuable. In general, optimizing our NLP formulation produces controllers that outperform other DEC-POMDP approaches in a range of domains. In both cases, very concise representations are found. Our results show that by using the NLP formulation, small, high-valued controllers can be efficiently found for a large assortment of POMDPs and DEC-POMDPs. Furthermore, performance is likely to improve as more specialized and scalable NLP solvers are developed that can take advantage of the structure of the controller optimization problem.

2 Background

As the DEC-POMDP model is an extension of the POMDP model, we begin with a overview of POMDPs and then show how this framework can be extended to cooperative multiagent environments modeled as DEC-POMDPs.

2.1 The POMDP model

A POMDP can be defined with the following tuple: $M = \langle S, A, P, R, \Omega, O \rangle$, with

- S, a finite set of states with designated initial state distribution b_0
- A, a finite set of actions
- P, the state transition model: $P(s'|s, a)$ is the probability of transitioning to state s' if action a is taken in state s
- R, the reward model: $R(s, a)$ is the immediate reward for taking action a in state s
- Ω, a finite set of observations
- O, the observation model: $O(o|s', a)$ is the probability of observing o if action a is taken and the resulting state is s'

We consider the case in which the decision making process unfolds over an infinite sequence of stages. At each stage the agent selects an action, which yields an immediate reward and an observation. The agent must choose an action based on the history of observations seen. Note that because the state is not directly observed, it may be beneficial for the agent to remember the observation history. This will help the agent to identify the set of possible states at any step. The objective of the agent is to maximize the expected discounted sum of rewards received. Because we consider the infinite sequence problem, we use a discount factor, $0 \leq \gamma < 1$, to maintain finite sums.

Finite-state controllers can be used as an elegant way of representing POMDP policies using a finite amount of memory [11]. The state of the controller is based on the observation sequence seen, and in turn the agent's actions are based on the state of its controller. To help distinguish states of the finite-state controller from states of the POMDP, we will refer to controller states as nodes. These controllers address one of the main causes of intractability in POMDP exact algorithms by not storing whole observation histories. Thus, controllers can encapsulate key pieces of information about the observation history with a fixed number of nodes. We also allow for stochastic transitions and action selection, as this can help to make up for limited memory [27]. We provide an example showing this phenomenon in a previous paper [1]. The finite-state controller can formally be defined by the tuple $\langle Q, \psi, \eta, q_0 \rangle$, where Q is the finite set of controller nodes, $\psi : Q \rightarrow \Delta A$ is the action selection model for each node, mapping nodes to distributions over actions, $\eta : Q \times A \times O \rightarrow \Delta Q$ represents the node transition model, mapping nodes, actions and observations to distributions over the resulting nodes, and q_0 represents a designated initial node. Thus, starting at a given controller node, an action is taken based on the node's action selection distribution and after an observation is seen, the controller transitions to another node based on the node transition model. This process of stochastically selecting actions and transitioning continues for the infinite steps of the problem. The value of a node q at state s, given action selection, ψ, and node transition probabilities, η, given as $P(a|q)$ and $P(q'|q, a, o)$, is specified by the following Bellman equation:

$$V(q, s) = \sum_a P(a|q) \times \left[R(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_o O(o|s', a) \sum_{q'} P(q'|q, a, o) V(q', s') \right]$$

2.2 The DEC-POMDP model

A DEC-POMDP involves multiple agents that operate under uncertainty based on different streams of observations. Like a POMDP, the infinite-horizon DEC-POMDP unfolds over an infinite sequence of stages. At each stage, every agent chooses an action based purely on its local observations, resulting in an immediate reward for the set of agents and an observation for each individual agent. Again, because the state is not directly observed, it may be beneficial for each agent to remember its observation history. A *local policy* for an agent is a mapping from local observation histories to actions while a *joint policy* is a set of policies, one for each agent in the problem. Like the POMDP case, the goal is to maximize the infinite-horizon total cumulative reward, beginning at some initial distribution over states b_0. Again, a discount factor, γ, is used to maintain finite sums.

More formally, a DEC-POMDP is defined by the tuple $\langle I, S, \{A_i\}, P, R, \{\Omega_i\}, O\rangle$ where I is the finite set of agents, S is again the finite set of states, A_i and Ω_i are the finite sets of actions and observations for each agent and P, R and O are now defined over the set of agents. That is, P is the set of state transition probabilities: $P(s'|s, \mathbf{a})$, the probability of transitioning from state s to s' when actions $\mathbf{a}$ are taken by the set of agents, R is the reward function: $R(s, \mathbf{a})$, the immediate reward for being in state s and agents taking actions $\mathbf{a}$, and O is the set of observation probabilities: $O(\mathbf{o}|s', \mathbf{a})$, the probability of agents seeing the set of observations $\mathbf{o}$ given the set of actions $\mathbf{a}$ has been taken which results in state s'.

Finite state controllers also provide an appealing way to model DEC-POMDP policies with finite memory. Each agent's local policy can be represented as a local controller and the resulting set of controllers supply the joint policy, called the joint controller. These local controllers are defined in the same way as the single agent controllers discussed above. The joint controller has a defined initial node for each local controller and transitions occur in the local controllers based on the action taken and the observation seen by the corresponding agent. The value for starting in nodes $\mathbf{q}$ and at state s with known action selection and node transition probabilities for each agent, i, is given by the following Bellman equation:

$$V(\mathbf{q}, s) = \sum_{\mathbf{a}} \prod_i P(a_i|q_i) \times \left[R(s, \mathbf{a}) + \gamma \sum_{s'} P(s'|\mathbf{a}, s) \sum_{\mathbf{o}} O(\mathbf{o}|s', \mathbf{a}) \sum_{\mathbf{q'}} \prod_i P\left(q_i'|q_i, a_i, o_i\right) V(\mathbf{q'}, s') \right]$$

Note that these values can be calculated offline in order to determine a controller for each agent that can then be executed online in a decentralized manner.

3 Previous work

A range of solution techniques for infinite-horizon POMDPs and DEC-POMDPs already exist. We first review POMDP controller-based approaches, as that is the focus of our work. These techniques seek to determine the best action selection and node transition parameters for a fixed-size controller. We also mention some other POMDP approximation methods that are based on improving the value function for a set of belief points (distributions over the state space). We then review the current infinite-horizon DEC-POMDP approximate algorithms and discuss the shortcomings of the current single and multiagent approximate methods.

Table 1 The linear program used by BPI

For a given node q and variables x_a and $x_{q',a,o}$

Maximize ϵ, subject to

Improvement constraints:

$$\forall s \; V(q,s) + \epsilon \leq \sum_a \left[x_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_o O(o|s',a) \sum_{q'} x_{q',a,o} V(q',s') \right]$$

Probability constraints:

$$\sum_a x_a = 1, \quad \forall a \; \sum_{q'} x_{q',a,o} = x_a, \quad \forall a \; x_a \geq 0, \quad \forall q', a, o \; x_{q',a,o} \geq 0$$

Variables x_a and $x_{q',a,o}$ represent $P(a|q)$ and $P(q',a|q,o)$ for a given node, q

3.1 POMDP approaches

Policy Iteration (PI) [11] is a technique to find epsilon-optimal POMDP controllers that alternates between policy improvement and evaluation. Although PI was originally developed for deterministic controllers, it can be extended to stochastic ones as well. In the improvement phase, dynamic programming is used to enlarge the controller. Also referred to as full backup, this entails creating a node for each possible combination of action and transition for each observation to any of the nodes in the current controller. Thus a controller with $|Q|$ nodes before the backup has $|A||Q|^{|\Omega|}$ nodes afterwards. The controller is then evaluated and nodes that have lesser or equal value for all states are replaced by nodes that (pointwise) dominate them. The incoming edges of these nodes are redirected to the dominating nodes, guaranteeing at least equal value. The new controller is then evaluated to determine the updated value for each node and state. This process continues until the controller is no longer changed by the improvement phase. While this method guarantees that a controller arbitrarily close to optimal will be found an intractable amount of memory is often required. Even when a high quality controller is found, it may be very large with many unnecessary nodes generated along the way. This is exacerbated by the fact that the algorithm cannot take advantage of an initial state distribution and must attempt to improve the controller for any initial state.

Poupart and Boutilier [23] developed a method called bounded policy iteration (BPI) that uses a one step dynamic programming lookahead to attempt to improve a POMDP controller without increasing its size. Like PI, this approach also alternates between policy improvement and evaluation. It iterates through the nodes in the controller and uses a linear program, shown in Table 1, to examine the value of probabilistically taking an action and then transitioning into the current controller. If an improvement can be found for all states, the action selection and node transition probabilities are updated accordingly. The controller is then evaluated and the cycle continues until no further improvement can be found. BPI guarantees to at least maintain the value of a provided controller, but it also does not use start state information resulting in larger than necessary controllers that are not likely to be optimal.

To increase the performance of BPI, two improvements were added by Poupart and Boutilier. First, they allowed the controller to grow by adding nodes when improvements can no longer be made. This may permit a higher value to be achieved than what BPI can produce with a fixed-size controller. Second, they used a heuristic for incorporating start state knowledge and increasing the performance of BPI in practice [22]. In this extension, termed biased BPI, improvement is concentrated in certain node and state pairs by weighing each pair by the (unnormalized) occupancy distribution, which can be found by solving the following set of linear equations:

$$o(q', s') = bp_0(q', s') + \gamma \sum_{q,s,a,o} o(q, s) P(s'|s, a) P(a|q) O(o|s', a) P(q'|q, a, o)$$

for all states and nodes. The value bp_0 is the probability of beginning in a node state pair. A factor, δ, can also be included, which allows the value to decrease by that amount in each node and state pair. This makes changes to the parameters more likely, as a small amount of value can now be lost. As a result, value may be higher for the start state and node, but as value can be lost for any pair, it could become lower instead.

Meuleau et al. [18] have proposed another approach to improve a fixed-size controller. They used GA to change the action selection and node transition probabilities and increase value. A cross-product MDP is created from the controller and the POMDP by considering the states of the MDP to be all combinations of the states of the POMDP and the nodes of the controller while actions of the MDP are based on actions of the POMDP and deterministic transitions in the controller after an observation is seen. The value of the resulting MDP can be determined and matrix operations allow the gradient to be calculated. The gradient can then be followed in an attempt to improve the controller.

Other approximation approaches that have recently gained a high level of popularity are point-based methods such as point-based value iteration (PBVI) [21], PERSEUS [31], heuristic search value iteration (HSVI) [28] and point-based policy iteration (PBPI) [15]. These techniques are approximations of value iteration for finite-horizon POMDPs, but they can find solutions for sufficiently large horizons that they can effectively produce infinite-horizon policies.

We will first discuss the basis of these algorithms, value iteration, and then briefly summarize these point-based methods. Value iteration is an optimal dynamic programming algorithm for finding solutions to POMDPs [30]. This is done by transforming the POMDP into a continuous state $|S|$-dimensional "belief state" MDP and building up the value function from the last step until the first. Policies, starting with just the action set on the last step, are backed up exhaustively at each step. A policy can then be pruned if for any belief state, there is some other policy with higher value. This procedure of backing up policies and pruning them continues until a desired finite-horizon is reached or for the infinite-horizon case, value does not change more than a given epsilon. The value of a policy can be represented as a vector in $|S|$-dimensional space. The optimal value function for any given horizon is then the max over these vectors resulting in a piecewise linear and convex value function. The infinite-horizon value function can be approximated within any ϵ using a finite set of vectors.

PBVI, PERSEUS, HSVI and PBPI fix the number of belief points considered at each step of value iteration. That is, at each step, the value vector of each belief point under consideration is backed up and the vector that has the highest value at that point is retained. This permits the number of vectors to remain constant and the value function is still defined over the entire belief space. If the right points are chosen, it may help concentrate the value iteration on certain regions in the belief space. PBVI selects the set of belief points by sampling from the action and observation sets of the problem while PERSEUS backs up only a randomly selected subset of these points. Thus, PERSEUS is often faster and produces a more concise solution than PBVI. PBVI has been shown to be optimal for sufficiently large and well constructed belief point sets, but this is intractable for reasonably sized problems. HSVI chooses points based on an additional upper bound that is incorporated into the algorithm. This algorithm will also converge to the optimal solution, but again this often requires an intractable amount of resources. PBPI uses Hansen's PI, but with PBVI rather than exact improvement over the whole belief space in the improvement phase. This allows faster performance and sometimes higher values than the original PBVI.

3.2 DEC-POMDP approaches

As mentioned above, the two algorithms that can solve infinite-horizon discounted DEC-POMDPs are those of Bernstein et al. [3] and Szer and Charpillet [33]. Bernstein et al.'s approach, called BPI for decentralized POMDPs (DEC-BPI), is a multiagent extension of Poupart and Boutilier's BPI algorithm. Like BPI, it is an approximate algorithm that seeks to improve stochastic finite-state controllers. Szer and Charpillet describe an approach that searches for optimal fixed-size deterministic controllers.

DEC-BPI, like BPI, improves a set of fixed-size controllers by using linear programming. This is done by iterating through the nodes of each agent's controller and attempting to find an improvement. The algorithm and linear program are very similar to those of BPI except the optimization is done over not just the states of the problem, but also the nodes of the other agents' controllers. That is, the linear program searches for a probability distribution over actions and transitions into the agent's current controller that increases the value of the controller for any initial state and any initial node of the other agents' controllers. If an improvement is discovered, the node is updated based on the probability distributions found. Each node for each agent is examined in turn and the algorithm terminates when no agent can make any further improvements.

Szer and Charpillet have developed a best-first search algorithm that finds deterministic finite-state controllers of a fixed size. This is done by calculating an approximate value for the controller given the current known deterministic parameters and filling in the remaining parameters one at a time in a best-first fashion. The authors prove that this technique will find the optimal deterministic finite-state controller of a given size.

3.3 Disadvantages of previous approximate algorithms

Because the optimal algorithms are not very practical, we focus on approximate methods. We discuss the limitations of these methods, first considering POMDP methods and then DEC-POMDP approaches.

The linear program used by BPI may allow for controller improvement, but low quality local maxima are likely to be found since it performs only a one step lookahead while keeping the controller values fixed. Also, because the improvement must be over all states it can be difficult to find parameters that meet this requirement, requiring large controllers to represent high-valued solutions. Adding the heuristics used in biased BPI reduce this necessity by optimizing over weighted states as well as allowing value to be lost. The disadvantage of biased BPI is that the heuristics are not useful in all problems and for cases when they are, domain knowledge or a large amount of experimentation may be necessary to properly set them.

Gradient ascent has similar problems with low quality local maxima as well other concerns. Meuleau et al. must construct a cross-product MDP from the controller and the underlying POMDP in a complex procedure to calculate the gradient. Also, the authors' representation does not take into account the probability constraints and thus does not calculate the true gradient of the problem. Due to this complex and incomplete gradient calculation, GA can be time consuming and error prone. Techniques more advanced than GA may be used to traverse the gradient, but these shortcomings remain.

Point-based methods perform very well on many problems, but rely on choosing points that are highly representative of the belief space. If there is a large amount of reachable belief space and the values of different areas of that space are diverse, many points would be needed to reasonably approximate the value. As more points are used, the algorithms become more similar to value iteration and thus become less tractable.

DEC-POMDP algorithms also have significant drawbacks. As DEC-BPI is an extension of BPI, its disadvantages are similar to those stated above. In general, it allows memory to remain fixed, but provides only a low quality locally optimal solution. This is due to the linear program considering the old controller values from the second step on and the fact that improvement must be over all possible states and initial nodes for the controllers of the other agents. As the number of agents or size of controllers grows, this later drawback is likely to severely hinder improvement. Also, start state knowledge is not used in DEC-BPI. While a heuristic like that used in biased BPI could be added, its applicability often depends on the domain and a more principled solution is desireable.

Szer and Charpillet's approach is limited due to the fact that it searches for optimal deterministic controllers. Since it focuses on deterministic controllers, larger controller sizes may often be required in order to produce values comparable to those generated by a stochastic counterpart. Since the approach relies on search to find an optimal set of controllers of a fixed size, it has a very high space and time requirements. These drawbacks indicate that a new approach is needed to improve the scalability and solution quality of POMDP and DEC-POMDP approximate methods.

4 Optimizing fixed-size controllers

Unlike other controller based approaches for POMDPs and DEC-POMDPs, our formulation defines an optimal controller for a given size. This is done by creating a set of new variables that represent the values of each node (or set of nodes in the DEC-POMDP case) and state pair. Intuitively, this allows changes in the controller probabilities to be reflected in the values of the nodes of the controller. This is in contrast to backups used by other controller-based methods which iteratively improve the probabilities and get stuck in low-quality local optima. Our approach allows both the values and probabilities in the controller to be optimized in one step, thus representing the optimal fixed-size controller. To ensure that these values are correct given the action selection and node transition probabilities, nonlinear constraints (based on the Bellman equation) must be added. The resulting NLP is generally harder to solve, but many robust and efficient algorithms can be applied.

One premise of our work is that an optimal formulation of the problem facilitates the design of solution techniques that can overcome the limitations of previous controller-based algorithms and produce better stochastic controllers. The general nature of our formulation allows a wide range of solution methods to be used. This also results in a search that is more sophisticated than previously used in controller-based methods. While no existing technique guarantees global optimality in a finite amount of time, experimental results show that our new formulation is advantageous. For instance, our results suggest that many POMDPs and DEC-POMDPs have small optimal controllers or can be approximated concisely with finite state-controllers. Thus, it is often unnecessary to use a large amount of memory in order to represent a high quality approximation. Our NLP is also able to take advantage of the start distribution of the problem, thus making better use of limited controller size. Lastly, because our method searches for stochastic controllers, it is able to find higher-valued, more concise controllers than search in the space of deterministic controllers.

Compared to point-based approaches, our formulation does not need to choose a set of points and may be able to cover the belief space better in some problems. That is, while point-based methods work well when there is a small reachable belief space or when the

Table 2 The NLP defining an optimal fixed-size POMDP controller

For variables: $x(q', a, q, o)$ and $z(q, s)$

Maximize $\sum_s b_0(s)z(q_0, s)$, subject to

The Bellman constraints:

$\forall q, s\ z(q, s) =$

$$\sum_a \left[\left(\sum_{q'} x(q', a, q, o_k)\right) R(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_o O(o|s', a) \sum_{q'} x(q', a, q, o) z(q', s')\right]$$

Probability constraints:

$\forall q, o\ \sum_{q',a} x(q', a, q, o) = 1, \quad \forall q', a, q, o\ x(q', a, q, o) \geq 0,$

$\forall q, o, a\ \sum_{q'} x(q', a, q, o) = \sum_{q'} x(q', a, q, o_k)$

Variable $x(q', a, q, o)$ represents $P(q', a|q, o)$, variable $z(q, s)$ represents $V(q, s)$, q_0 is the initial controller node and o_k is an arbitrary fixed observation

chosen points are very representative of the whole space, the NLP approach seeks to optimize the value of a controller for a specific initial point. For domains in which point-based methods cannot find representative belief points, our approach may still be able to construct high quality controllers. Also, since point-based methods rely on finite-horizon dynamic programming, it may be difficult for these methods to complete the number of backups necessary to approximate the value function well. As our approach uses finite-state controllers, it is more suitable for finding infinite-horizon policies.

In the rest of this section, we give a formal description of the NLP and prove that its optimal solution defines an optimal controller of a fixed size. We begin with the POMDP formulation and then show the extension to the DEC-POMDP case.

4.1 NLP formulation for POMDPs

Unlike BPI, which alternates between policy improvement and evaluation, our nonlinear program improves and evaluates the controller in one phase. The value of an initial node is maximized at an initial state distribution using parameters for the action selection probabilities at each node $P(a|q)$, the node transition probabilities $P(q'|q, a, o)$, and the values of each node in each state $V(q, s)$. To ensure that the value variables are correct given the action and node transition probabilities, nonlinear constraints must be added to the optimization. These constraints are the Bellman equations given the policy determined by the action selection and node transition probabilities. Linear constraints are used to maintain proper probabilities.

To reduce the representation complexity, the action selection and node transition probabilities are merged into one, with

$$P(q', a|q, o) = P(a|q)P(q'|q, a, o) \quad \text{and} \quad \sum_{q'} P(q', a|q, o) = P(a|q)$$

This results in a quadratically constrained linear program. QCLPs may contain quadratic terms in the constraints, but have a linear objective function. They are a subclass of general nonlinear programs that has structure which algorithms can exploit. This produces a problem that is often more difficult than a linear program, but simpler than a general nonlinear program. The QCLP formulation permits a large number of algorithms to be applied. Because the QCLP is a subclass of the general NLP and to use the same terminology as the DEC-POMDP case, we will refer to the QCLP formulation as an NLP.

Table 2 describes the NLP which defines an optimal fixed-size controller for the POMDP case. The value of a designated initial node is maximized given the initial state distribution and the necessary constraints. The first constraint represents the Bellman equation for each node and state which maintains correct values as probability parameters change. The second and third constraints ensure that the x variables represent proper probabilities which sum to one and have value greater than 0. The last constraint guarantees that action selection does not depend on the resulting observation which has not yet been seen. That is, the action probabilities are equal for all observations and thus for any observation, o_{k}, the value $\sum_{q'} P(q', a|q, o_{\mathrm{k}}) = P(a|q)$.

Theorem 1 *An optimal solution of the NLP in Table 2 results in an optimal stochastic controller for the given size and initial state distribution.*

Proof To prove this, we must show that an optimal solution of the NLP provides a valid finite-state controller and that the value of this controller is optimal for the given controller size. The NLP defines the controller parameters as $P(a|q) = \sum_{q'} x(q', a, q, o_{\mathrm{k}})$ and $P(q'|q, a, o) = \frac{x(q',a,q,o_{\mathrm{k}})}{\sum_{q'} x(q',a,q,o_{\mathrm{k}})}$ with $q, q', q_0 \in Q$. The probability constraints guarantee that these values are nonnegative and sum to one. Because the parameters are valid, this ensures that a valid stochastic finite-state controller is defined by the NLP. These probabilities are utilized in the Bellman equation for each node q and state s, ensuring that the correct value is determined for each node and state pair. Lastly, because the value of the controller is maximized at the initial state distribution of the problem, an optimal solution of NLP must provide an optimal stochastic controller for the the initial distribution and given size. □

4.2 NLP formulation for DEC-POMDPs

The nonlinear program for DEC-POMDPs is an extension of the one defined above which incorporates multiple controllers. That is, the NLP seeks to optimize the value of a set of fixed-size controllers given an initial state distribution and the DEC-POMDP model. The variables for this problem are the action selection and node transition probabilities for each node of each agent's controller as well as the joint value of a set of controller nodes. Hence, these variables are for each agent i, $P(a_i|q_i)$ and $P(q_i'|q_i, a_i, o_i)$ and for the set of agents and any state, $V(\mathbf{q}, s)$. Similar to the way our POMDP approach differs from BPI, this approach differs from DEC-BPI in that it explicitly represents the node values as variables, thus allowing improvement and evaluation to take place simultaneously. To ensure that the values are correct given the action and node transition probabilities, nonlinear constraints must be added to the optimization. These constraints are the DEC-POMDP Bellman equations given the policy determined by the action and transition probabilities. We must also ensure that the necessary variables are valid probabilities.

The NLP representation for DEC-POMDPs is shown formally for an arbitrary number of agents in Table 3. The value of the initial set of nodes at the initial state distribution is maximized subject to Bellman and probability constraints. The Bellman constraints ensure that correct values are maintained for each set of nodes and state of the domain. The probability constraints ensure that the action and node transition values are proper probabilities.

Theorem 2 *An optimal solution of the NLP in Table 3 results in optimal stochastic controllers for the given size and initial state distribution.*

Table 3 The NLP defining a set of optimal fixed-size DEC-POMDP controllers

For variables of each agent i: $x(q_i, a_i)$, $y(q_i, a_i, o_i, q_i')$ and $z(\mathbf{q}, s)$

Maximize $\sum_s b_0(s) z(\mathbf{q^0}, s)$, subject to

The Bellman constraints:

$\forall \mathbf{q}, s\ z(\mathbf{q}, s) =$

$$\sum_{\mathbf{a}} \left(\prod_i x(q_i, a_i) \left[R(s, \mathbf{a}) + \gamma \sum_{s'} P(s'|s, \mathbf{a}) \sum_{\mathbf{o}} O(\mathbf{o}|s', \mathbf{a}) \sum_{\mathbf{q'}} \prod_i y(q_i, a_i, o_i, q_i') z(\mathbf{q'}, s') \right] \right)$$

And probability constraints for each agent i:

$\forall q_i \sum_{a_i} x(q_i, a_i) = 1, \quad \forall q_i, o_i, a_i \sum_{q_i'} y(q_i, a_i, o_i, q_i') = 1$

$\forall q_i, a_i\ x(q_i, a_i) \geq 0, \quad \forall q_i, o_i, a_i\ y(q_i, a_i, o_i, q_i') \geq 0$

For each agent i, variable $x(q_i, a_i)$ represents $P(a_i|q_i)$, variable $y(q_i, a_i, o_i, q_i')$ represents $P(q_i'|q_i, a_i, o_i)$ and variable $z(\mathbf{q}, s)$ represents $V(\mathbf{q}, s)$ where $\mathbf{q^0}$ represents the initial controller node for each agent

Table 4 The more scalable NLP representation for POMDPs

For variables: $x_F(q', a, o)$, $x_R(q', q, o)$ and $z(q, s)$

Maximize $\sum_s b_0(s) z(q_0, s)$, subject to

The Bellman constraints:

$\forall q \neq q_0, s\ z(q_0, s) =$

$$\sum_a \left[\left(\sum_{q'} x_F(q', a, o_k) \right) R(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_o O(o|s', a) \sum_{q'} x_F(q', a, o) z(q', s') \right]$$

$\forall q \neq q_0, s\ z(q, s) =$

$R(s, a_q) + \gamma \sum_{s'} P(s'|s, a_q) \sum_o O(o|s', a_q) \sum_{q' \neq q_0} x_R(q', q, o) z(q', s')$

Probability constraints:

$\forall o \sum_{q'} x_F(q', a, o) = 1, \quad \forall q \neq q_0, o \sum_{q'} x_R(q', q, o) = 1$

$\forall q', a, o\ x_F(q', a, o) \geq 0, \quad \forall q', q \neq q_0, o\ x_R(q', q, o) \geq 0\ \forall o, a \sum_{q'} x_F(q', a, o) = \sum_{q'} x_F(q', a, o_k)$

Variable $x_F(q', a, o)$ represents $P(q', a|q_0, o)$, variable $x_R(q', q, o)$ represents $P(q'|q_0, o)$ for a fixed action, a_q at node q, variable $z(q, s)$ represents $V(q, s)$, q_0 is the initial controller node and o_{k} is an arbitrary fixed observation

Proof Like the POMDP case, the optimality of the controllers follows from the NLP constraints and maximization of given initial nodes at the initial state distribution. We will show that an optimal solution of the NLP provides a valid set of finite-state controllers and that the value of these controllers is optimal for the given size. The controller parameters are defined in the NLP as $P(a_i|q_i) = x(q_i, a_i)$ and $P(q_i'|q_i, a_i, o_i) = y(q_i', a_i, q_i, o_i)$ with $q, q', q_0 \in Q$. The probability constraints for each agent guarantee that these values are nonnegative and sum to one. Because the parameters are independent and valid for each agent, the NLP defines a valid set of stochastic finite-state controllers. The validity of these probabilities ensures that the value produced by Bellman constraints is correct for each set of nodes $\mathbf{q}$ and state s. Because the value of the set of controllers is maximized at the initial state distribution of the problem, an optimal solution of NLP must provide an optimal set of stochastic controllers for the initial distribution and given size. □

4.3 Scaling up the NLP formulations

In order to solve NLP representations with a larger number of nodes, we can fix the actions chosen at some nodes of the controller. An example of a POMDP formulation with fixed actions is given in Table 4. The initial node of the controller contains both stochastic action selection and stochastic transitions, but the remaining nodes have fixed actions. Because the remaining nodes maintain stochastic transitions, there can be very little or no expressiveness lost in this representation. This is because the action selection probabilities can be merged with the transition probabilities and then this single parameter can be found. If the actions are fixed correctly, the resulting controller will be equivalent to one with stochastic actions and transitions.

The DEC-POMDP NLP formulation can be similarly adapted by beginning at a fully stochastic node in each agent's controller and then transitioning to nodes that have fixed actions. In both cases, a more efficient representation can be constructed by never allowing the initial node to transition to more than one node with a given fixed action. It makes no difference in the controller if the initial node transitions to any specific node with that fixed action as the nodes can be reordered without affecting the value. Because these NLPs have fewer variables, high quality solutions may be found for controller sizes that are too large when using the representation from Tables 2 and 3.

5 Incorporating correlation in the decentralized model

In order to improve the performance of a set of controllers, a shared source of randomness in the form of a *correlation device* can be added. As an improvement to DEC-BPI, Bernstein et al. show that higher-valued controllers can sometimes be achieved when incorporating this approach without sharing any local information. As an example of a correlation device, imagine that before each action is taken, a coin is flipped and both agents have access to the outcome. Each agent can then use the new information to affect their choice of action. Along with stochasticity, correlation is another means of increasing value when memory is limited. For instance, assume the agents' actions are randomized so that each chooses action A 50% of the time and action B 50% of the time. This results in the joint actions of (A, A), (A, B), (B, A) and (B, B) each 25% of the time. When a coin is added, the agents can correlate their actions based on what is seen, such as allowing each agent to perform action A when "heads" is seen and and action B when "tails" is seen. As each signal occurs 50% of the time, this permits joint actions (A, A) and (B, B) to also take place 50% of the time. This type of policy is impossible without the correlation device showing that the policies can be randomized and correlated to allow higher value to be attained.

It is worth noting that there are many ways a correlation device can be implemented in practice. These implementations include each agent using a random number generator with a common seed, or agreeing before execution starts on a list of random numbers. In each of these examples, each agent has access to the random signal at each step, but no local information is shared and thus the execution remains decentralized. While a correlation device permits higher value to be found given a fixed-size controller, a larger set of uncorrelated controllers can represent the same value. Essentially, this is done by incorporating the correlation signal into the agents' controllers. More details can be found in [4].

More formally, a correlation device provides extra signals to the agents and operates independently of the DEC-POMDP. That is, the correlation device is a tuple $\langle C, \psi \rangle$, where C is a set of states and $\psi : C \rightarrow \Delta C$ is a stochastic transition function that we will represent as $P(c'|c)$. At each step of the problem, the device transitions and each agent can observe its state.

The independent local controllers defined above (in Table 3) can be modified to make use of a correlation device. This is done by making the parameters dependent on the signal from the correlation device. For agent i, action selection is then $P(a_i|q_i, c)$ and node transition

Table 5 The nonlinear program for including a correlation device in our DEC-POMDP NLP formulation

For variables of each agent i: $x(q_i, a_i, c)$, $y(q_i, a_i, o_i, q_i', c)$, $z(\mathbf{q}, s, c)$ and $w(c, c')$

Maximize $\sum_s b_0(s) z(\mathbf{q^0}, s, c)$, subject to

The Bellman constraints:

$\forall \mathbf{q}, s\ z(\mathbf{q}, s, c) = \sum_{\mathbf{a}} \prod_i x(q_i, a_i, c) \cdot$

$\left[R(s, \mathbf{a}) + \gamma \sum_{s'} P(s'|s, \mathbf{a}) \sum_{\mathbf{o}} O(\mathbf{o}|s', \mathbf{a}) \sum_{\mathbf{q'}} \prod_i y(q_i, a_i, o_i, q_i', c) \sum_{c'} w(c, c') z(\mathbf{q'}, s', c)\right]$

For each agent i, variable $x(q_i, a_i, c)$ represents $P(a_i|q_i, c)$, variable $y(q_i, a_i, o_i, q_i', c)$ represents $P(q_i'|q_i, a_i, o_i, c)$, variable $z(\mathbf{q}, s, c)$ represents $V(\mathbf{q}, s, c)$, $\mathbf{q^0}$ represents the initial controller node for each agent and $w(c, c')$ represents $P(c'|c)$. The other constraints are similar to those above with the addition of a sum to one constraint for the correlation device

is $P(q_i'|q_i, a_i, o_i, c)$. For n agents, the value of the correlated joint controller beginning in nodes $\mathbf{q}$, state s and correlation device state c is defined as

$$V(\mathbf{q}, s, c) = \sum_{\mathbf{a}} \prod_i P(a_i|q_i, c) \left[R(s, \mathbf{a}) + \gamma \sum_{s'} P(s'|\mathbf{a}, s) \sum_{\mathbf{o}} O(\mathbf{o}|s', \mathbf{a}) \sum_{\mathbf{q'}} \prod_i \right.$$
$$\left. \times P(q_i|q_i, a_i, o_i, c) \sum_{c'} P(c'|c) V(\mathbf{q'}, s', c') \right]$$

Our NLP formulation can be extended to include a correlation device. The optimization problem, shown in Table 5 *without* the probability constraints, is very similar to the previous NLP. A new variable is added for the transition function of the correlation device and the other variables now include the signal from the device. The Bellman equation incorporates the new correlation device signal at each step, but the other constraints remain the same. A new probability constraint is added to ensure that the transition probabilities for each state of the correlation device sum to one.

6 Methods for solving the NLP

Many efficient constrained optimization algorithms can be used to solve large NLPs. Constrained optimization seeks to minimize or maximize an objective function based on equality and inequality constraints. When the objective and all constraints are linear, this is called a linear program (LP). As our POMDP formulation has a linear objective function, but contains some quadratic constraints, it is a quadratically constrained linear program. The DEC-POMDP representation also has a linear objective, but has some nonlinear constraints whose degree depends on the number of agents in the problem. Unfortunately, both of these problems are nonconvex. Essentially, this means that there may be multiple local maxima as well as global maxima, thus finding globally optimal solution is often very difficult. In the next two subsections, we discuss a method to find these globally optimal solutions as well as a more practical method for finding locally optimal solutions.

6.1 Finding a globally optimal solution for our NLP

One method that could be used to find optimal solutions for our NLP is *difference of convex functions* (DC) programming [14]. DC programming uses a general formulation that allows nonlinear and nonconvex constraints and is studied within the field of global optimization. This formulation requires an objective function and constraints that are a difference of two convex functions. Under mild conditions, it is possible to devise DC programming algorithms that converge to the globally optimal solution.

More formally, a function f is called DC if it is real-valued and defined over a convex set $C \subset \mathbb{R}^n$ and $\forall\, x \in C$, f can be decomposed such that $f(x) = p(x) - q(x)$ where p and q are convex functions on C. A *DC programming problem* then has the form

$$\min f(x) \quad \text{such that} \quad x \in C \quad \text{and} \quad g_i(x) \leq 0$$

where $C \subset \mathbb{R}^n$ is convex and objective function f and each constraint g_i are DC on C.

Proposition 1 *Optimal solution of the POMDP and DEC-POMDP NLP formulations can be defined as DC programming problems.*

Proof First, the maximization in our formulations can be changed to a minimization by multiplying by a factor of -1. We can then define x and y over all nonnegative Real numbers and z over all Real numbers. This represents a convex set over which the variables are defined. We can represent each equality constraint in the NLP formulation as inequality constraints with opposite sign to achieve the form above. Horst and Tuy show that any function whose second partial derivatives are continuous is also DC (corollary I.1). Because the NLP objective function and all constraints are polynomials, this means they are infinitely differentiable and these derivatives are continuous. Thus, the variables are defined over a convex set and the objective and constraints are DC in both formulations. This allows our NLP representations to be written as DC programming problems. □

Therefore, POMDPs and DEC-POMDPs can be formulated as DC programming problems. This has mostly theoretical significance; we doubt that this formulation could be effective in practice. In principle, global optimization algorithms such as branch and bound can be used, but because of the large size of the resulting DC programing problem, it is unlikely that current optimal solvers can handle even small POMDPs or DEC-POMDPs. Nevertheless, it would be interesting to identify classes of these problems for which DC optimization is practical.

6.2 Finding a locally optimal solution for our NLP

Since it may not be possible or feasible to solve the NLP optimally, locally optimal methods are often more useful in practice. A wide range of nonlinear programming algorithms have been developed that are able to efficiently solve nonconvex problems with many variables and constraints. Locally optimal solutions can be guaranteed, but at times, globally optimal solutions can also be found. For example, merit functions, which evaluate a current solution based on fitness criteria, can be used to improve convergence and the problem space can be made convex by approximation or domain information. These methods are much more robust than simpler methods such as GA, while retaining modest efficiency in many cases.

For this paper, we used a freely available nonlinearly constrained optimization solver called *snopt* [10] on the NEOS server (www-neos.mcs.anl.gov). The algorithm finds solutions by a method of successive approximations called sequential quadratic programming (SQP). SQP uses quadratic approximations which are then solved with quadratic programming (QP) until a solution to the more general problem is found. A QP is typically easier to solve, but must have a quadratic objective function and linear constraints. In snopt, the objective and constraints are combined and approximated to produce the QP. A merit function is also used to guarantee convergence from any initial point.

We are also exploring other techniques that take advantage of the structure of POMDPs and DEC-POMDPs in order to increase the scalability of certain solution methods. One promising idea is to use constraint partitioning [35] to break up a problem and solve the resulting pieces in such a way that allows a feasible solution to be found. For this approach, we are exploring different methods for breaking up our representations and more efficient ways of solving the subproblems. We are also examining ways to simplify our nonlinear program formulations and determine solution techniques that match these representations well, thereby increasing performance.

7 Experiments

For experimental comparison, we present an evaluation of the performance of our formulation using an off-the-shelf nonlinear program solver, snopt, as well as leading POMDP and DEC-POMDP approximation techniques. In these experiments we seek to determine how well our formulation performs when used in conjunction with a generic solver such as snopt. The formulation is very general and many other solvers may be applied. As mentioned above, we are currently developing a customized solver that would take further advantage of the inherent structure of the NLPs and increase scalability. While the POMDP results are intended to show that a simple solution of the NLP is competitive with current solution methods and can outperform point-based approaches on certain problems, the DEC-POMDP results show consistent improvement in solution quality over previous approaches in a range of problems.

7.1 POMDP results

In this section, we compare the results obtained using our new formulation with those of biased BPI, PBVI, PERSEUS, HSVI and PBPI. GA was also implemented, but produced significantly worse results and required substantially more time than the other techniques. Thus we omit the details of GA and focus on the more competitive techniques.

We also implemented a version of the fixed action formulation described in Table 4. For controller sizes larger than the number of available actions, we cycled through the actions and assigned the next available action to the next node. For smaller controller sizes, we randomly assigned actions to nodes. Throughout this section, we will refer to the optimization of our NLP using snopt as *NLO* for the fully stochastic case and as *NLO fixed* for the optimization with fixed actions.

Each of our NLP methods was initialized with ten random deterministic controllers and we report mean values and times for solving the given controller size. To slightly increase the performance of the solver, upper and lower bounds of $R_{\max}/(1-\gamma)$ and $R_{\min}/(1-\gamma)$ were added. The values $R_{\max} = \max_{s,a} R(s,a)$ and $R_{\min} = \min_{s,a} R(s,a)$ and the bounds represent repeating the best and worst possible immediate reward for the infinite steps of the problem. The results were found by using the NEOS server, which provides a set of machines with varying CPU speeds, but we expect the times to vary by only a small constant.

7.1.1 Benchmark problems

We first provide a comparison of our NLP formulations with leading POMDP approximation methods on common benchmark problems. The first three domains, which were introduced by Littman, Cassandra and Kaelbling [16], are grid problems in which an agent must navigate to a goal square. The set of actions consists of turning left, right or around, moving forward and staying in place. The observations consist of the different views of the walls in the each domain based on each agent's ability to observe walls in four directions and a unique observation for the goal state. There are also three additional observable landmarks in the Hallway problem. Both the observations and transitions are extremely noisy and the start state is a random non-goal state. Only the goal has a reward of 1 and the discount factor used was 0.95. Following the convention of previously published results, we consider the versions of the Hallway problems that stop after the goal has been reached and for the Tiger-grid problem reward only accumulates for the first 500 steps.

The other benchmark problem is a larger grid problem in which the goal is for the agent to catch and tag an opponent which attempts to move away [21]. The tagging agent has perfect knowledge of its state, but it cannot observe the oppenent unless it is located in the same square. The agent can deterministically move in each of four directions and can use the "tag" action. The oppenent attempts to move away from the agent in a fixed way, but may stay in the same location with probability 0.2. If the agent succeeds in tagging the oppenent, a reward of 10 is given, but if the oppenent is not in the same square when the tag action is taken a reward of -10 is given. All other actions result in a -1 reward and like the other benchmarks, the problem stops once the goal is reached and a discount factor of 0.95 was used.

Table 6 shows the results from previously published algorithms and our NLP formulations. We provide the mean values and times for the largest controller size that is solvable with less than (approximately) 400 MB of memory and under 8 h as these limits are imposed on the NEOS server. The values of PBPI in the table are the highest values reported for each problem, but the authors did not provide results for the Hallway problem. Because the experiments were conducted on different computers, solution times give a general idea of the speed of the algorithms. It is also worth noting that while most of the other approaches are highly optimized, a generic solver is used with our approach in these experiments.

In general, we see that the nonlinear optimization approach is competitive both in running time and value produced, but does not outperform the other techniques. Our method achieves 91, 92, 83% of maximum value in the first three problems, but does so with a much smaller representation size. A key aspect of the NLP approach is that high quality solutions can be found with very concise controllers. This allows limited representation size to be utilized very well, likely better than the other approaches in most problems.

The values in the table as well as those in Fig. 1 show that our approach is currently unable to find solutions for large controller sizes. For example, in the Tag problem, a solution could only be found for a two node controller in the general case and a seven node controller when the actions were fixed. As seen in the figures, there is a near monotonic increase of value in the Hallway problem when compared with either controller size or time. As expected, fixing the actions at each node results in faster performance and increased scalability. This allows higher valued controllers to be produced, but scalability remains limited. As improved solvers are found, higher values and larger controllers will be solvable by both approaches. The other benchmark problems display similar trends of near monotonic improvement and the need for increased scalability to outperform other methods on these problems.

7.1.2 Infinite-horizon domains

Because the above benchmark problems are indefinite-horizon (or technically finite-horizon in the case of the Tiger-grid problem) methods such as point-based approaches that build up

Table 6 Values, representation sizes and running times (in seconds) for the set of benchmark problems

	Value	Size	Time
Tiger-grid $\lvert S\rvert = 36$, $\lvert A\rvert = 5$, $\lvert\Omega\rvert = 17$			
HSVI	2.35	4,860	10341
PERSEUS	2.34	134	104
HSVI2	2.30	1,003	52
PBVI	2.25	470	3448
PBPI	2.24	3,101	51
Biased BPI	2.22	120	1000
NLO fixed	2.14	32	2411
BPI	1.81	1,500	163420
NLO	1.79	14	1174
Q_{MDP}	0.23	N/A	2.76
Hallway $\lvert S\rvert = 60$, $\lvert A\rvert = 5$, $\lvert\Omega\rvert = 21$			
PBVI	0.53	86	288
HSVI2	0.52	147	2.4
HSVI	0.52	1,341	10836
Biased BPI	0.51	43	185
PERSEUS	0.51	55	35
BPI	0.51	1,500	249730
NLO fixed	0.49	25	644
NLO	0.47	12	362
Q_{MDP}	0.27	N/A	1.34
Hallway2 $\lvert S\rvert = 93$, $\lvert A\rvert = 5$, $\lvert\Omega\rvert = 17$			
PERSEUS	0.35	56	10
HSVI2	0.35	114	1.5
PBPI	0.35	320	3.1
HSVI	0.35	1,571	10010
PBVI	0.34	95	360
Biased BPI	0.32	60	790
NLO fixed	0.29	18	251
NLO	0.28	13	420
BPI	0.28	1,500	274280
Q_{MDP}	0.09	N/A	2.23
Tag $\lvert S\rvert = 870$, $\lvert A\rvert = 5$, $\lvert\Omega\rvert = 30$			
PBPI	−5.87	818	1133
PERSEUS	−6.17	280	1670
HSVI2	−6.36	415	24
HSVI	−6.37	1,657	10113
Biased BPI	−6.65	17	250
NLO fixed	−8.14	7	5669
BPI	−9.18	940	59772
PBVI	−9.18	1,334	180880
NLO	−13.94	2	5596
Q_{MDP}	−16.9	N/A	16.1

Results for other algorithms were taken from the following sources: BPI [23], biased BPI [22], HSVI [28], HSVI2 [29], PERSEUS [31], PBPI [15], PBVI [21], Q_{MDP} [31]

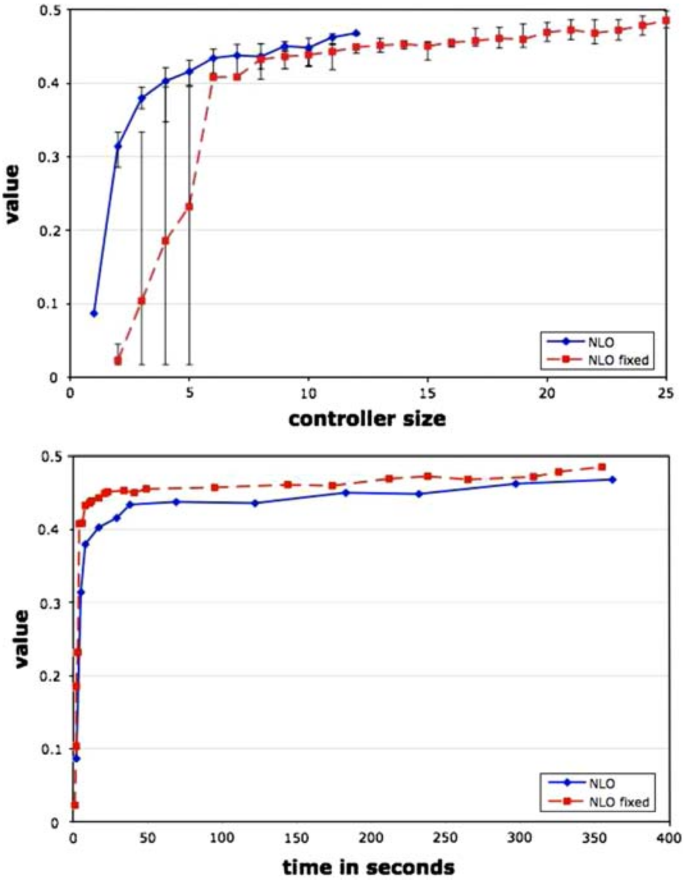

Fig. 1 Hallway problem values for various controllers sizes (*top*) and running times (*bottom*) using the NLP formulation with and without fixed actions after the first node. *Bars* are used in the top figure to represent the minimum and maximum values for each size

policies one step at a time will often perform well. Controller-based approaches may also perform well in indefinite-horizon problems because the controller may concisely represent a solution, but they cannot easily be used for finite-horizon problems. For the Tiger-grid problem, because the discount factor is 0.95, after 500 steps, the maximum reward is less than 1×10^{-11} so it can effectively be considered an infinite-horizon problem without affecting the final solution quality.

True infinite-horizon problems with higher discount rates pose a tougher challenge for point-based methods. In these problems, the effective horizon will be high and thus methods based on value iteration will require a large number of steps in order to construct a policy. Conversely, controller-based methods will have an advantage over value iteration methods. This is because the controller can concisely represent an infinite-horizon solution and generating this controller often requires similar resources regardless of discount factor. In cases where an optimal or near-optimal policy is periodic, a controller-based method can represent a very high quality solution without the repetition in the policy that would be required for value iteration approaches.

Table 7 Values, representation sizes and running times (in seconds) for the original Hallway and Hallway2 problems that reset after the goal has been reached

	Value	Size	Time
Hallway $\|S\| = 60$, $\|A\| = 5$, $\|\Omega\| = 21$			
NLO fixed	51.98	18	185
NLO	48.62	7	232
HSVI2	48.45	112	3622
Hallway2 $\|S\| = 93$, $\|A\| = 5$, $\|\Omega\| = 17$			
NLO fixed	1.97	13	309
NLO	1.66	6	163
HSVI2	1.18	2,540	3627

To demonstrate this phenomenon, we examined the performance of our NLP approaches and HSVI2 in the original versions of the Hallway and Hallway2 problems which reset to the initial state after the goal has been reached. The discount factors for the Hallway domain were 0.999 and 0.99 for the Hallway2 domain. HSVI2 was chosen for comparison because of very high performance in each of the benchmark domains and availability of the algorithm. Software for HSVI2 was used from web site of Trey Smith which was run with a time limit of 1 hour.

On these problems, which we provide results for in Table 7, our NLP formulation provides results that have higher value, smaller representation and require less time than HSVI2. While the improvement is modest in the Hallway domain, it is much more substantial in the Hallway2 domain. As stated above, this is due to the high discount factor and the ability of a concise, periodic policy to perform well. Using fixed actions at each node after the first allows further performance gains. This occurs because the fixed action formulation is more scalable, permitting larger controllers to be solved. Overall, these results demonstrate the advantages of controller-based approaches in general and our NLP approach in particular in true infinite-horizon domains with discount factors close to one.

7.2 DEC-POMDP results

We also tested our nonlinear programming approach in four DEC-POMDP domains. In each experiment, we compare our NLP solutions using snopt with Bernstein et al.'s DEC-BPI and Szer and Charpillet's BFS for a range of controller sizes. We also implemented the NLP and DEC-BPI techniques with a correlation device of size two. Like the POMDP version, to improve scalability we also solved the NLP with a stochastic action at the initial node and fixed actions at the other nodes. The action assignments were made as described above with each action in turn fixed to each node when the number of nodes was greater than the number of actions and random action assignments in each node for smaller controller sizes.

Each NLP and DEC-BPI algorithm was run until convergence was achieved with ten different random deterministic initial controllers, and the mean values and running times are reported. Unlike the figure in the POMDP section, the figures in this section do not include maximum and minimum value bars. While these could have been provided, they are very similar for each of our NLP methods and are omitted to increase readability. Also, we do not always show the results for the largest solvable controller once the values plateau. The optimal deterministic controller returned by BFS is also reported if it could be found in under 8 h. The times reported for each NLP method can only be considered estimates due to running each algorithm on external machines with uncontrollable load levels, but we expect that they vary by only a small constant factor. Throughout this section we will refer to the optimization of our NLP with snopt as *NLO*, the optimization with the correlation device with two states as *NLO corr*, and these optimizations with fixed actions as *NLO fixed* and *NLO fixed corr*.

7.2.1 Recycling robots

As a first comparison, we have extended the Recycling Robot problem [32] to the multiagent case. The robots have the task of picking up cans in an office building. They have sensors to find a can and motors to move around the office in order to look for cans. The robots are able to control a gripper arm to grasp each can and then place it in an on-board receptacle. Each robot has three high level actions: (1) search for a small can, (2) search for a large can or (3) recharge the battery. In our two agent version, the larger can is only retrievable if both robots pick it up at the same time. Each agent can decide to independently search for a small can or to attempt to cooperate in order to receive a larger reward. If only one agent chooses to retreive the large can, no reward is given. For each agent that picks up a small can, a reward of 2 is given and if both agents cooperate to pick the large can, a reward of 5 is given. The robots have the same battery states of high and low, with an increased likelihood of transitioning to a low state or exhausting the battery after attempting to pick up the large can. Each robot's battery power depends only on its own actions and each agent can fully observe its own level, but not that of the other agent. If the robot exhausts the battery, it is picked up and plugged into the charger then continues to act on the next step with a high battery level. The two robot version used in this paper has 4 states, 3 actions and 2 observations. Also, a discount factor of 0.9 was used for all problems.

As seen in Fig. 2, in this domain all the methods performed fairly well. While the DEC-BPI techniques converge over time to a value around 27, the other approaches converge to at least 30. BFS, our general formulation (NLO) and our formulation with fixed actions and a correlation device (NLO fixed corr) perform particularly well on this problem, converging to a value near 32 with very small controllers. We also see that these values were found quickly, often taking less than a minute for most methods. BFS could solve at most a controller of size three, but in this problem that is sufficient to produce a high-valued solution. We also see that while correlation helps in the case of fixed actions (NLO fixed corr), it does not improve solution quality in the general case (NLO corr). Fixing actions in our NLP approach (NLO fixed) allows larger controllers to be solved, but an increase in value is not found.

7.2.2 Multiagent tiger problem

Another domain with 2 states, 3 actions and 2 observations called the multiagent tiger problem was introduced by Nair et al. [19]. In this problem, there are two doors. Behind one door is a tiger and behind the other is a large treasure. Each agent may open one of the doors or listen. If either agent opens the door with the tiger behind it, a large penalty is given. If the door with the treasure behind it is opened and the tiger door is not, a reward is given. If both agents choose the same action (i.e., both opening the same door) a larger positive reward or a smaller penalty is given to reward this cooperation. If an agent listens, a small penalty is given and an observation is seen that is a noisy indication of which door the tiger is behind. While listening does not change the location of the tiger, opening a door causes the tiger to be placed behind one of the doors with equal probability.

In this problem, DEC-BPI performed very poorly (never producing values higher than −50) causing us to remove the values from Fig. 3 for the sake of clarity. We can also see

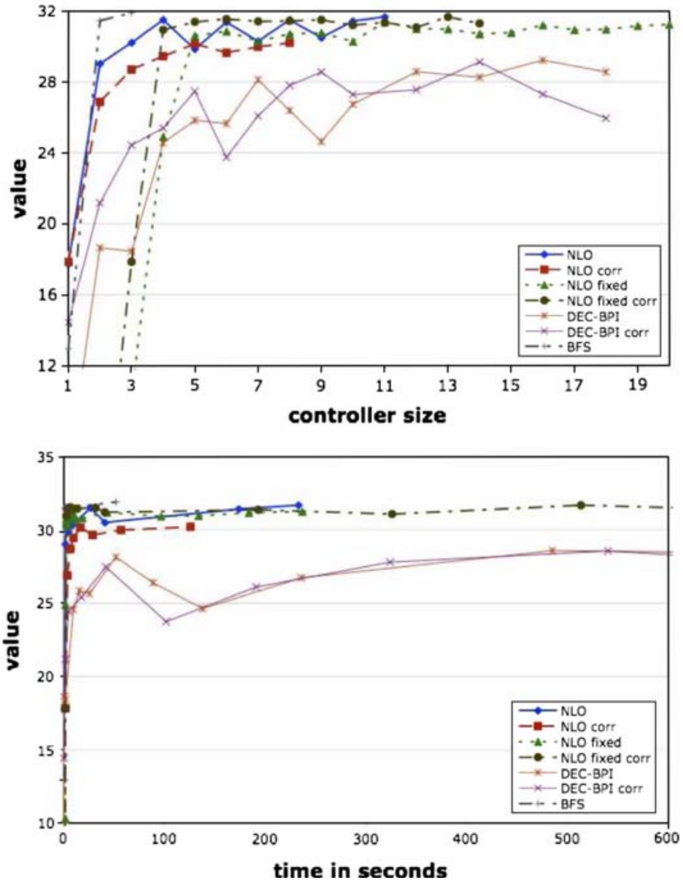

Fig. 2 Recycling robots values for various controller sizes (*top*) and running times (*bottom*) using NLP methods and DEC-BPI with and without a 2 node correlation device and BFS

the limitation of BFS in the figures. While it can do very well for three nodes, it cannot find solutions for larger controllers. In contrast, the NLP approaches demonstrate a trend of higher-valued solutions as controller size increases. This suggests that as more scalable solution methods for our formulation are found, the increase in solution quality will persist. The NLP approaches also make better use of time, quickly producing high solution quality and then value increases more slowly. We also see that while correlation is helpful, fixing actions does not improve solution quality. This is likely because a high quality solution for this domain will consist of many listen actions, but very few open actions. Our simple method of choosing actions does not take this information into account and thus requires larger controllers to produce a high value. While displaying an upward trend, the solutions for this problem, particularly with fixed actions, are somewhat irregular. This is likely due to the fact that there is a large penalty for opening the wrong door which can significantly lower the average presented in the figure. Thus, there is a large variance in solution quality but the high performance of the NLP methods is clear.

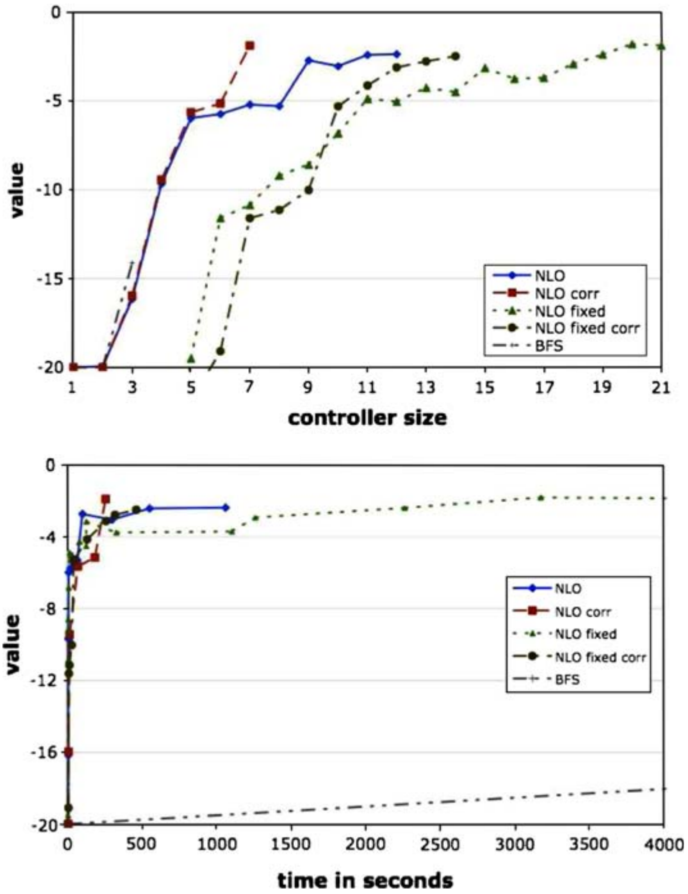

Fig. 3 Multiagent tiger problem values for various controller sizes (*top*) and running times (*bottom*) using NLP methods with and without a 2 node correlation device and BFS

7.2.3 Meeting in a grid

A larger domain with 16 states, 5 actions and 2 observations was also introduced by Bernstein et al. In this problem, two agents must meet in a 2 x 2 grid with no obstacles. The agents start diagonally across from each other and available actions are move left, right, up, or down and stay in place. Only walls to the left and right can be observed, resulting in each agent knowing only if it is on the left or right half. The agents cannot observe each other and do not interfere with other. Action transitions are noisy with the agent possibly moving in another direction or staying in place and a reward of 1 is given for each step the agents share the same square.

The results for this example can be seen in Fig. 4. The figures show that, in general, our NLP techniques and BFS outperform DEC-BPI. We also see that the fixed action NLO methods (NLO fixed and NLO fixed corr) perform especially well, producing the highest values. The standard NLP methods also perform well, as they provide nearly the same value as the fixed action methods. The figure on the bottom shows that for all methods, a near maximum value was reached quickly by all methods (∼1 min) and then, if larger controllers could be solved, value increased very slightly thereafter. We also see that correlation

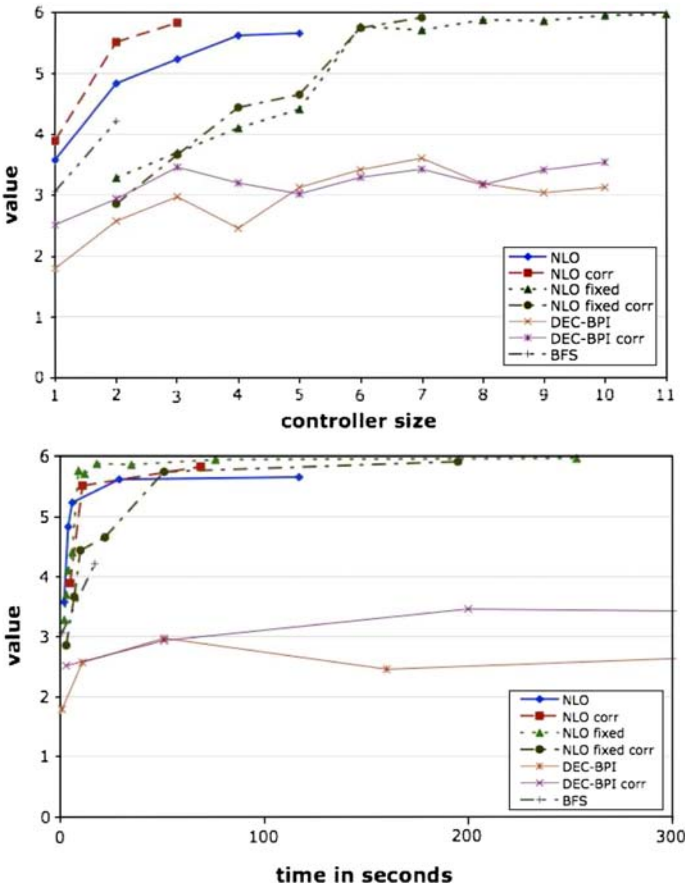

Fig. 4 Meeting in a grid problem values for various controller sizes (*top*) and running times (*bottom*) using NLP methods and DEC-BPI with and without a 2 node correlation device and BFS

increases solution quality in this problem, particularly when using the general formulation (NLO corr).

7.2.4 Box pushing

A much larger domain was introduced by Seuken and Zilberstein [25]. This problem, with 100 states, 4 actions and 5 observations consists of two agents that can gain reward by pushing different boxes. The agents begin facing each other in the bottom corners of a 4 x 3 grid with the available actions of turning right, turning left, moving forward or staying in place. There is a 0.9 probability that the agent will succeed in moving and otherwise will stay in place, but the two agents can never occupy the same square. The middle row of the grid contains two small boxes and one large box. This large box can only be moved by both agents pushing at the same time. The upper row of the grid is considered the goal row. The possible deterministic observations consist of seeing an empty space, a wall, the other agent, a small box or the large box. A reward of 100 is given if both agents push the large box to the goal row and 10 is given for each small box that is moved to the goal row. A penalty of −5 is given for

Table 8 Box pushing problem values and running times (in seconds) for each controller size using NLP methods and DEC-BPI with and without a 2 node correlation device and BFS

	NLO	NLO corr	NLO fix	NLO fix corr	DEC-BPI	DEC-BPI corr	BFS
Value							
1	−1.580	6.267	N/A	N/A	−10.367	−10.012	−2
2	31.971	39.825	−6.252	−10.865	3.293	4.486	x
3	46.282	50.834	5.097	5.300	9.442	12.504	x
4	50.640	x	18.780	25.590	7.894	x	x
5	x	x	53.132	53.132	14.762	x	x
6	x	x	73.247	x	x	x	x
7	x	x	80.469	x	x	x	x
Time							
1	20	21	N/A	N/A	26	87	1696
2	115	314	18	21	579	3138	x
3	683	1948	27	37	4094	15041	x
4	5176	x	44	108	11324	x	x
5	x	x	92	12189	27492	x	x
6	x	x	143	x	x	x	x
7	x	x	256	x	x	x	x

An ’x’ signifies that the approach was not able to solve the problem and because our fixed action formulation cannot solve 1 node controllers, an ’N/A’ is given

each agent that cannot move and −0.1 is given for each time step. Once a box is moved to the goal row, the environment resets to the original start state.

The results for this problem, which is an order of magnitude larger than previously published infinite-horizon problems, are given in Table 8. Because this problem is so large and given the time and memory limitations, BFS is only able to provide a solution for a one node controller and both DEC-BPI and our general approach with a correlation device (NLO corr) could not solve the formulation for a four node controller. Nonetheless, our techniques perform very well. The general formulations (NLO and NLO corr) never produce lower value than DEC-BPI or BFS and often produce much higher value. Fixing actions allows solutions to be found very quickly, but because random actions are used for controller sizes 2–4, performance is poor for these sizes. Because larger controllers can be solved, these fixed action formulations (NLO fixed and NLO fixed corr) allow greater value to be found. The results from all domains suggest that the general formulation should be used when controller size is less than the number of actions in the problem, but the fixed action method can provide benefits for larger controller sizes. Also, all NLP methods required much less time than the DEC-BPI techniques when the problems were solvable. And in general, correlation provides only a slight increase in value for the various methods.

8 Conclusions

We introduce a new approach for solving POMDPs and DEC-POMDPs by defining optimal fixed-size solutions as nonlinear programs. This permits these problems to be solved by a wide range of powerful NLP solution methods. Because our approach applies nonlinear optimization to a stochastic controller in the single agent case or a set of stochastic controllers for the multiagent case, it is able to make better use of the limited representation space than other approaches. Thus, for a large number of POMDPs and DEC-POMDPs this new formulation may allow very concise high quality solutions.

We show that by using a generic nonlinear program solver, our formulation can provide POMDP solutions that are competitive with leading techniques in terms of running time and solution quality. The NLP method also outperforms one such method (HSVI2) in two domains with high discount factors and no goal state (in which the problem terminates). In infinite-horizon domains with high discount factors, point-based algorithms and other approaches based on value iteration will require many backup steps to represent a policy and thus performance suffers. On the contrary, controller-based methods such as our formulation represent infinite-horizon policies, eliminating the need for these backups. In many of these infinite-horizon domains, the stochastic and periodic policies of controller-based methods are likely to perform well.

Our approach has some other distinctive characteristics and advantages. In particular, it produces high value with significantly less memory. And because the controller is more compact, the approach may use less time despite the use of nonlinear optimization. This work opens up new promising research directions that could produce further improvement in both quality and efficiency. As the solver in our experiments has not been optimized for the NLP it is likely that even higher performance can be achieved in the future.

In the DEC-POMDP case, we demonstrate that our approach is able to produce solutions that are significantly higher-valued than the other infinite-horizon DEC-POMDP algorithms for a range of problems. In all problems, a variant of the NLP approach produces the highest value that is generated for any controller size of any method. This included a problem that is an order of magnitude larger than any used previously in infinite-horizon DEC-POMDP literature. The combination of start state knowledge and more advanced optimization techniques allows us to make efficient use of the limited space of the controllers. In addition, incorporating a correlation device as a shared source of randomness can further increase solution quality. Also, the running times needed to produce solutions for each problem show that the NLP approach is frequently the most efficient method to achieve any given value. These results show that our approach can quickly find compact, high-valued solutions for several different types of DEC-POMDPs. Lastly, as the number of agents in a given problem grows, the number of local maxima is likely to increase. Because solution of our NLP can avoid many of these suboptimal points, our representation should also perform well when the number of agents is increased beyond two.

In the future, we plan to explore more specialized algorithms that can be tailored for our optimization problem. While the performance achieved with a standard nonlinear optimization algorithm is good, specialized solvers will likely further increase solution quality and scalability. For instance, in both POMDPs and DEC-POMDPs different optimization methods should be able to take advantage of the inherent problem structure. Also, we are interested in identifying important subclasses for which globally optimal solutions can be efficiently provided. We also plan to characterize the circumstances under which introducing a correlation device is cost effective. For important classes of POMDPs and DEC-POMDPs, these improvements could help the algorithm find optimal or near optimal fixed-size solutions.

Acknowledgments We would like to thank Marek Petrik and the anonymous reviewers for their helpful comments. Support for this work was provided in part by the National Science Foundation under grants IIS-0535061 and IIS-0812149, and by the Air Force Office of Scientic Research under grants FA9550-05-1-0254 and FA9550-08-1-0181.

References

1. Amato, C., Bernstein, D. S., & Zilberstein, S. (2007). Solving POMDPs using quadratically constrained linear programs. In: *Proceedings of the twentieth international joint conference on artificial intelligence*. (pp. 2418–2424.) Hyderabad, India.
2. Becker, R., Zilberstein, S., Lesser, V., & Goldman, C.V. (2004). Solving transition-independent decentralized Markov decision processes. *Journal of AI Research, 22*, 423–455.
3. Bernstein, D. S., Hansen, E., & Zilberstein, S. (2005). Bounded policy iteration for decentralized POMDPs. In: *Proceedings of the nineteenth international joint conference on artificial intelligence*. (pp. 1287–1292). Edinburgh, Scotland.
4. Bernstein, D.S., Amato, C., Hansen, E.A., & Zilberstein, S. (2009). Policy iteration for decentralized control of Markov decision processes. *Journal of AI Research, 34*, 89–132.
5. Bertsekas, D. P. (2004). *Nonlinear programming*. Belmont, MA: Athena Scientific.
6. Cassandra, A. R. (1998a). A survey of POMDP applications. In: AAAI fall symposium: Planning with POMDPs. Orlando, FL.
7. Cassandra, A. R. (1998b). Exact and approximate algorithms for partially observable Markov decision processes. PhD thesis. Brown University Providence, RI.
8. Eckles, J.E. (1968). Optimum maintenance with incomplete information. *Operations Research, 16*, 1058–1067.
9. Emery-Montemerlo, R., Gordon, G., Schneider, J., & Thrun, S. (2004). Approximate solutions for partially observable stochastic games with common payoffs. In: *Proceedings of the third international joint conference on autonomous agents and multiagent systems* (pp. 136–143). New York, NY.
10. Gill, P. E., Murray, W., & Saunders, M. (2005). Snopt: An SQP algorithm for large-scale constrained optimization. *SIAM Review, 47*, 99–131.
11. Hansen, E. A. (1998). Solving POMDPs by searching in policy space. In: *Proceedings of the fourteenth conference on uncertainty in artificial intelligence*. (pp. 211–219). Madison, WI.
12. Hansen, E. A., Bernstein, D. S., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In: *Proceedings of the nineteenth national conference on artificial intelligence*. (pp. 709–715). San Jose, CA.
13. Hauskrecht, M., & Fraser, H. (1998). Modeling treatment of ischemic heart disease with partially observable Markov decision processes. In: *Proceedings of American medical informatics association annual symposium on computer applications in health care*. (pp. 538–542). Orlando, Florida.
14. Horst, R., & Tuy, H. (1996). *Global optimization: Deterministic approaches*. New York: Springer.
15. Ji, S., Parr, R., Li, H., Liao, X., & Carin, L. (2007). Point-based policy iteration. In: *Proceedings of the twenty-second national conference on artificial intelligence*. (pp. 1243–1249). Vancouver, Canada.
16. Littman, M.L., Cassandra, A.R., & Kaelbling, L.P. (1995). *Learning policies for partially observable environments: Scaling up. Technical report CS-95-11*. Providence, RI: Brown University, Department of Computer Science.
17. Marecki, J., Gupta, T., Varakantham, P., Tambe, M., & Yokoo, M. (2008). Not all agents are equal: Scaling up distributed POMDPs for agent networks. In: *Proceedings of the seventh international joint conference on autonomous agents and multiagent systems*. (pp. 485–492). Estoril, Portugal.
18. Meuleau, N., Kim, K. E., Kaelbling, L. P., & Cassandra, A. R. (1999). Solving POMDPs by searching the space of finite policies. In: *Proceedings of the fifteenth conference on uncertainty in artificial intelligence*. (pp. 417–426). Stockholm, Sweden.
19. Nair, R., Pynadath, D., Yokoo, M., Tambe, M., & Marsella, S. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In: *Proceedings of the nineteenth international joint conference on artificial intelligence*. (pp. 705–711). Acapulco, Mexico.
20. Petrik, M., & Zilberstein, S. (2007). Average-reward decentralized Markov decision processes. In *Proceedings of the twentieth international joint conference on artificial intelligence* (pp. 1997–2002). Hyderabad, India.
21. Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In: *Proceedings of the eighteenth international joint conference on artificial intelligence*. (pp. 1025–1032). Acapulco, Mexico.
22. Poupart, P. (2005). *Exploiting structure to efficiently solve large scale partially observable Markov decision processes*. PhD thesis. University of Toronto.
23. Poupart, P., & Boutilier, C. (2004). Bounded finite state controllers. In: S. Thrun, L. Saul, & B. Schölkopf (Eds.), *Advances in neural information processing systems 16*, Cambridge, MA: MIT Press.
24. Seuken, S., & Zilberstein, S. (2007a). Memory-bounded dynamic programming for DEC-POMDPs. In: *Proceedings of the twentieth international joint conference on artificial intelligence*. (pp. 2009–2015). Hyderabad, India.

25. Seuken, S., & Zilberstein, S. (2007b). Improved memory-bounded dynamic programming for decentralized POMDPs. In: *Proceedings of the twenty-third conference on uncertainty in artificial intelligence*. Vancouver, Canada.
26. Simmons, R., & Koenig, S. (1995). Probabilistic navigation in partially observable environments. In: *Proceedings of the fourteenth international joint conference on artificial intelligence*. (pp. 1080–1087). Montral, Canada.
27. Singh, S., Jaakkola, T., & Jordan, M. (1994). Learning without state-estimation in partially observable Markovian decision processes. In: *Proceedings of the eleventh international conference on machine learning*. (pp. 284–292). New Brunswick, NJ.
28. Smith, T., & Simmons, R. (2004). Heuristic search value iteration for POMDPs. In: *Proceedings of the twentieth conference on uncertainty in artificial intelligence*. (pp. 520–527). Banff, Canada.
29. Smith, T., & Simmons, R. (2005). Point-based POMDP algorithms: Improved analysis and implementation. In: *Proceedings of the twenty-first conference on uncertainty in artificial intelligence*. Edinburgh, Scotland.
30. Sondik, E. J. (1971). *The optimal control of partially observable Markov processes*. PhD thesis. Stanford University.
31. Spaan, M. T. J., & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *Journal of AI Research, 24*, 195–220.
32. Sutton, R. S., & Barto, A. G. (1995). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.
33. Szer, D., & Charpillet, F. (2005). An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In: *Proceedings of the sixteenth European conference on machine learning*. (pp. 389–399). Porto, Portugal.
34. Szer, D., Charpillet, F., & Zilberstein, S. (2005). MAA*: A heuristic search algorithm for solving decentralized POMDPs. In: *Proceedings of the twenty-first conference on uncertainty in artificial intelligence*. (pp. 576–583). Edinburgh, Scotland.
35. Wah, B. W., & Chen, Y. (2005). Solving large-scale nonlinear programming problems by constraint partitioning. In: *Proceedings of the eleventh international conference on principles and practice of constraint programming*. (pp. 697–711).