Point-based Incremental Pruning Heuristic for Solving Finite-Horizon DEC-POMDPs

Jilles S. Dibangoye
University of Caen
Caen, France
gdibango@info.unicaen.fr

Abdel-Illah Mouaddib
University of Caen
Caen, France
mouaddib@info.unicaen.fr

Brahim Chai-draa
Laval University
Québec, Qc, Canada
chaib@aid.ift.ulaval.ca

ABSTRACT

Recent scaling up of decentralized partially observable Markov decision process (DEC-POMDP) solvers towards realistic applications is mainly due to approximate methods. Of this family, MEMORY BOUNDED DYNAMIC PROGRAMMING (MBDP), which combines in a suitable manner top-down heuristics and bottom-up value function updates, can solve DEC-POMDPs with large horizons. The performances of MBDP, can be, however, drastically improved by avoiding the systematic generation and evaluation of all possible policies which result from the exhaustive backup. To achieve that, we suggest a heuristic search method, namely POINT BASED INCREMENTAL PRUNING (PBIP), which is able to distinguish policies with different heuristic estimates. Taking this insight into account, PBIP searches only among the most promising policies, finds those useful, and prunes dominated ones. Doing so permits us to reduce clearly the amount of computation required by the exhaustive backup. The computation experiment shows that PBIP solves DEC-POMDP benchmarks up to 800 times faster than the current best approximate algorithms, while providing solutions with higher values.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation, Performance

Keywords

artificial intelligence, decentralized pomdps, branch-and-bound, point-based solver, planning under uncertainty

1. INTRODUCTION

Many interesting complex problems that involve two or more agents that cooperate to optimize a joint reward function, while having different local observations, can be modeled as DEC-POMDPs. These problems arise naturally in various quantitative disciplines including Computer Science (*e.g.* control of multiple robots for space exploration) , Economics (*e.g.* decentralized supply chains) and Operations Research (*e.g.* network traffic routing). Unfortunately, finding either optimal or even ε-approximate solutions of such problems has been shown to be particularly hard [2, 10]. To date, most DEC-POMDP algorithms are assumed not to be able to scale to real-world-size problems [6, 13, 14].

There are two distinct, but closely related, reasons for the limited scalability of DEC-POMDP solvers. The more widely known reason is the so-called *curse of dimensionality* [1]: in a problem with $|S|$ physical states, n agents, and for agent i, the set of teammates' strategies Q_{-i}, DEC-POMDP planners must reason about an $|S^n \times Q_{-i}|$-dimensional continuous belief space that grows exponentially with the number of agents. This is because each agent has to reason about the other agents' policies evaluated over the joint state space S^n. This explains why most DEC-POMDP algorithms cannot solve problems with a number of agents larger than two, and a few dozen states. The other well known reason for the computational burden of DEC-POMDPs is the *curse of history* where the number of joint histories grows double-exponentially with the planning horizon and the number of agents [9]. In most domains, the curse of history affects DEC-POMDP algorithms far more strongly than the curse of dimensionality. This suggests that if we can avoid the curse of history, there are many real-world DEC-POMDPs where the curse of dimensionality is *not* a problem. Recent attempts have been made to whittle down the set of histories considered [3, 4, 11, 12, 13], but so far the state-of-the-art technique, MBDP [12], still remains constrained by the curse of history.

For the general finite-horizon DEC-POMDPs that we are interested in, MBDP [12] is currently the most successful approximate algorithm. MBDP, in comparison to other DYNAMIC PROGRAMMING (DP) algorithms, has two benefits: first, it is *bounded*, that is, it does not have to keep exponential many policies in memory but only a fixed number denoted by parameter $maxTrees$. Moreover, it chooses top-down heuristics to determine a set of beliefs that enables the best policies to be selected. However, an undesirable effect of this strategy is that it requires the exhaustive enumeration of all possible joint policies of the current iteration using joint policies from the last iteration. This is achieved by means of the so called *exhaustive backup*. Unfortunately, in the worst case the resulting set of joint policy-trees requires an exponential space with respect to the number of observations and the number of agents. This issue motivated us to search for new heuristic methods in order to avoid the exhaustive enumeration of all joint policy-trees at each iteration.

The main contribution of this paper is to introduce a novel technique that is able to select the best policy-trees for a given iteration with respect to an initial belief state and policy-trees from the last iteration. This method aims at replacing the time and memory consuming operator of all dynamic programming methods, namely the exhaustive backup. To do so, we suggest the POINT BASED INCREMENTAL PRUNING heuristic (PBIP). PBIP circumvents the problem

Cite as: Point-based Incremental Pruning Heuristic for Solving Finite-Horizon DEC-POMDPs, Jilles S. Dibangoye, Abdel-Illah Mouaddib, Brahim Chai-draa, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 569–576

of searching in the entire space of joint policy-trees by expanding, at each iteration, only the most promising joint policy-trees, and pruning dominated ones. Since we often expand only a few joint policy-trees, we may either converge much faster to an approximate solution or use larger parameter $maxTrees$ to improve the solution quality. This is achieved by means of a threefold method: (1) identifying a bijection from a subspace of joint policy-trees of the underlying multi-agent POMDP (MPOMDP) to the space of joint policy-trees of the original DEC-POMDP; (2) using the beliefs selected by top-down heuristics to compute exact and upper-bound estimates of joint policy-trees from the last iteration; (3) using these estimates to traverse the exponential space of joint policy-trees of the underlying MPOMDP towards the small space of relevant ones.

2. BACKGROUND AND RELATED WORK

In this section, we present the DEC-POMDP model and some of the state-of-the-art approaches.

2.1 An overview of DEC-POMDPs

The DEC-POMDP framework is a generalized model for a cooperative group of agents that operates in domains involving hidden states and uncertain action effects. A n-agent DEC-POMDP is a tuple $(I, S, \{A_i\}_i, P, R, \{\Omega_i\}_i, O, T, b_0)$. Let I be a finite set of agents indexed by $1 \cdots n$. Let S be a finite set of states. Let $A_i = \{a_1, a_2, \cdots\}$ be a finite set of actions available for agent i, and $A = \otimes_{i \in I} A_i$ is the finite set of joint actions a, where $a = (a^1, \cdots, a^n)$ and variable a^i denotes the value of an action executed by agent $i \in I$. Let $P(s'|s,a)$ be a function of transition probabilities. Let $R(s,a)$ be a real-valued reward function. Let $\Omega_i = \{o_1, o_2, \cdots\}$ define a finite set of observations available for agent i, and $\Omega = \otimes_{i \in I} \Omega_i$ is the finite set of joint observations $o \in \Omega$, where $o = (o^1, \cdots, o^n)$ and variable o^i denotes the value of an observation received by agent $i \in I$. Let $O(o|a,s)$ be a function of observation probabilities. Let T be the finite-horizon. Let b_0 be the initial belief state of the system.

Given a DEC-POMDP, we aim at finding a solution that yields the highest long-term reward, $E[\sum_{t=0}^{T-1} R(s_t, a_t) |\, b_0]$, for a given initial belief b_0. A policy for a single agent i, *policy-tree*, can be represented as a decision tree denoted q_i, where nodes are labeled with actions $a_i \in A_i$ and arcs are tagged with observations $o_i \in \Omega_i$. Let Q_i^t be the set of horizon-t policy-trees available for agent i. A solution to a horizon-t DEC-POMDP can then be seen as a vector of horizon-t policy-trees. We denote a vector of policy-trees by $\vec{q}_t = (q_1^t, \cdots, q_n^t)$, one policy-tree for each agent. The set of vectors of policy-trees is referred to as $Q^t = \otimes_{i \in I} Q_i^t$. We also define $V(s, \vec{q}_t)$ as the expected value of executing the vector of policy-trees $\vec{q}_t$ starting in state s. This value can be easily computed using dynamic programming:

$$V(s, \vec{q}_t) = R(s, \alpha(\vec{q}_t)) + \sum_{o,s'} P(s'|s, \vec{q}_t, o) V(s', \eta(\vec{q}_t, o)) \quad (1)$$

where $\eta(\vec{q}_t, o)$ is the vector of policy-trees executed by agents after receiving joint observation o. We use $\alpha(\vec{q}_t)$ to denote the root node of vector of policy-trees $\vec{q}_t$; We denote by $P(s'|s, \vec{q}_t, o)$ the probability $P(s'|s, \alpha(\vec{q}_t)) O(o|s', \alpha(\vec{q}_t))$ of being in state s' after executing joint policy $\vec{q}_t$ from state-observation pair (s, o). An optimal vector of policy-trees $\vec{q}_{\star T} = (q_1^T, \cdots, q_n^T)$ for a given initial belief state b_0 can then be determined as follows:

$$\vec{q}_{\star T} = \arg\max_{\vec{q}_T \in Q^T} \sum_s b_0(s) V(s, \vec{q}_T) \quad (2)$$

A number of algorithms have been proposed to build either optimal or approximate vector of policy-trees $\vec{q}_T$ using top-down [14] or bottom-up [6, 13] techniques.

2.2 Top-down techniques

Many attempts have been made to use top-down techniques for solving DEC-POMDPs [14, 15]. These heuristic search methods can handle large domains effectively, by means of combining lower and upper bounds and knowledge of the initial information (starting belief point). Szer *et* al. [14] provided the first heuristic search algorithm namely MAA* for finite-horizon DEC-POMDPs. This algorithm is based on the combination of classical heuristic method A* and decentralized control theory. It seeks to compute globally optimal solutions. Although MAA* is a great improvement on earlier techniques, it suffers from a major drawback: its ability to constrain the search space depends on lower and upper bounds accuracy. Unfortunately tighter bounds often incur considerable computation efforts. While this algorithm fails to scale to DEC-POMDPs with large horizons, it shows promising results demonstrating the potential of forward search methods. The key difference between PBIP and MAA* is that while PBIP improves the joint policy-tree one fringe at a time, MAA* proceeds by improving the joint policy-tree one time step at a time (simultaneously for all agent involved). By improving the joint policy-tree one fringe at a time we provide a more accurate pruning mechanism than MAA*. This is a strong argument that outlines the soundness of the proposed approach. Varakantham *et* al. [15] suggested a forward heuristic technique, SPIDER, that exploits the weak interaction between teammates to leverage MAA* limitations. More precisely, SPIDER improves the joint policy one agent at a time in the order of teammate interactions. Doing so permits SPIDER to scale up to distributed POMDP with larger number of agents. While interesting, SPIDER usefulness depends on the assumption that teammate interactions are largely loosely coupled. For the finite-horizon DEC-POMDPs that we are interested in, *i.e.,* where teammate interactions are either unknown or strongly coupled, SPIDER offers no benefits.

2.3 Bottom-up techniques

In accordance with Szer, the major limitation of bottom-up approaches is the explosion in memory and time requirements, since each step requires first generating and evaluating all joint policy-trees for the next horizon, *i.e.,* the exhaustive backup, before beginning the step of pruning. While this is similar to POMDPs, it appears to be far more limiting in DEC-POMDPs, because value functions are now defined over a much larger policy space that includes all policies of all agents.

2.3.1 Dynamic programming.

The DYNAMIC PROGRAMMING algorithm (DP) [6] consists of two steps: the *exhaustive backup* and the pruning of dominated policies. For each agent, the exhaustive backup takes as input an initial set of horizon-$(t-1)$ policy-trees. Then, it builds all possible horizon-t policy-trees such that each tree of the new set has only subtrees[1] that appear in the initial set. The resulting sets of policy-trees are exponential with respect to the number of observations. Finally, the pruning step eliminates the dominated policy-trees in each new set. Finding dominated policy-trees, however, can be expensive since checking whether a policy-tree is dominated requires solving a linear program. Despite clever strategies including heuristic search methods [14] and point-based techniques [13], exact algorithms do not scale to larger problems. The intractability of exact approaches has led to the development of a wide variety of approximate methods. Of this family, MBDP [12] and its extensions (IMBDP [11] and MBDP-OC [5]) are the only one that scale to much

[1] throughout the paper '*vectors of policy-trees from the last iteration*' will be referred as to '*subtrees*' for the sake of simplicity.

longer horizons.

2.3.2 Approximate dynamic programming solvers.

Recent bottom-up techniques, MBDP, IMBDP and MBDP-OC somewhat mitigate DP limitations by means of: (1) selecting only a small number ($maxTrees$) of policies for each agent at each horizon; (2) reducing the exponential role of observations by sampling them. However, choosing the *right* $maxTrees$ for a given DEC-POMDP is not obvious and dynamically adjusting $maxTrees$ may raise non-negligible computation costs. Moreover, the usefulness of the second enhancement is domain-dependant. Even more importantly, in the worst case the solution sets at a given horizon are still built in exponential time with respect to the number of relevant observations denoted $maxObs$ and the number of agents $|I|$, *i.e.,* $|S^2||A||\Omega|maxTrees^{maxObs|I|+1}$. The problem with MBDP and its extensions is that they traverse the entire exponential space of vectors of policy-trees to determine the best vectors of policy-trees for a given set of belief states. To better understand the complexity of MBDP, let $maxTrees$ be the number of policy-trees in the previous solution set of each agent. The first step creates $|A_i|maxTrees^{|\Omega_i|}$ policy-trees for agent i; and the second step evaluates $|A|maxTrees^{\sum_i|\Omega_i|}$ vectors of policy-trees for each belief point. Therefore the new solution sets are built in exponential time $|S^2||A||\Omega|maxTrees^{\sum_i|\Omega_i|+1}$, where $|\Omega|$, $|A|$ and $|S|$ grow exponentially with $|I|$. More importantly, it does so even though only a few vectors of policy-trees are relevant to achieve optimal or near-optimal behavior. Nevertheless, not enough efforts have been made to exploit this insight. Currently all bottom-up approaches trying to solve DEC-POMDPs make use of the exhaustive backup. They apply essentially the same strategies as solving a DEC-POMDP where all vectors of policy-trees have the same heuristic estimate. So these approaches do not have a *smart* subroutine to distinguish vectors of policy-trees with a low heuristic estimate from those with a high heuristic estimate. Considering this intuition, we would like to design a general algorithm that is able to identify such vectors of policy-trees in order to avoid the exhaustive backup. Our proposed approach provides the basis of incremental pruning techniques in DEC-POMDPs suitable for either finite or infinite horizon cases. In contrast to all MBDP, IMBDP and MBDP-OC, PBIP allows using larger parameter $maxTrees$ (improving the solution quality by the way) and *automatically* prunes some of the work.

3. INCREMENTAL PRUNING HEURISTIC

In this section, we describe a general overview of the proposed solution approach, optimizations and main properties are discussed in depth in the next sections. Before going into further detail, we need to introduce some additional definitions.

3.1 The Space of Joint Policy-Trees

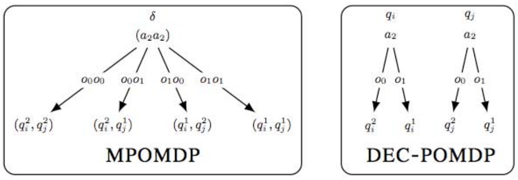

Figure 1: MPOMDP and DEC-POMDP joint policy-trees.

Since our goal is to determine a vector of policy-trees for a DEC-POMDP, it seems clear that the search space should be the space of vectors of policy-trees. However, for the sake of simplicity, we consider a slightly different representation of these policies. In fact, unlike the classical representation (vector of policy-trees), we use a single decision tree for the entire group of agents, namely *joint policy-tree* and denoted δ. In the remainder of this paper, a joint policy-tree is a decision tree, where the root node $\alpha(\delta) = (\alpha(q_1), \cdots, \alpha(q_n))$ is labeled by a joint action; arcs are tagged by joint observations; and subtrees are vectors of policy-trees $\eta(\delta, o) = (\eta(q_1, o^1), \cdots, \eta(q_n, o^n))$ from the last iteration. The right hand side of Figure 1 illustrates such a representation. The reader will note that this representation corresponds exactly to the policy-tree of the underlying MPOMDP.

3.2 Problem Reformulation

A desirable effect follows from the above observation: the problem of finding the optimal joint policy-tree δ^t, for a given belief b and subtrees Q^{t-1}, is *equivalent* to the problem of determining joint action $\alpha(\delta^t)$ and subtrees $\eta(\delta^t, o)$ such that the resulting joint policy-tree δ^t is both *valid* and *optimal*. A joint policy-tree is said to be valid if there exists a vector of policy-trees $(q_1, \cdots, q_n)$ which satisfies the following constraints $(c1)$ and $(c2)$:

- $(c1)$ $\alpha(\delta) = (\alpha(q_1), \cdots, \alpha(q_n))$,
- $(c2)$ $\eta(\delta, o) = (\eta(q_1, o^1), \cdots, \eta(q_n, o^n))$.

more precisely, δ^t is a valid joint policy-tree if it corresponds to a unique vector of policy-trees. Otherwise, the joint policy-tree is said to be non-valid, when it is a policy-tree of the underlying MPOMDP that does not match to a vector of policy-trees.

3.3 The Heuristic Approach

At this point, we are left with two problems: first, how subtrees help us to compute the heuristic estimates of joint policies; moreover, how can these estimates be used to traverse our search space towards valid and useful joint policy-trees.

3.3.1 Computing the heuristic estimates

In order to compute the heuristic estimate of a joint policy-tree δ, for a given belief b and subtrees, we proceed as follows: select both joint action $\alpha(\delta)$, and subtrees $\eta(\delta, o)$ that yield the highest *contributions* [9], where:

- the contribution of subtree $\eta(\delta, o)$ is given by,

$$g_{b,o}^{\delta} = \textstyle\sum_{s'} P(s'|b, \delta, o)V(s', \eta(\delta, o)) \tag{3}$$

  where $P(s'|b, \delta, o) = \sum_s b(s)P(s'|s, \delta, o)$. The evaluation of all contributions requires a polynomial time. To better understand this complexity, let's $maxTrees$ be the maximum number of policy trees kept in memory for each agent at each horizon. We create $|A||\Omega|maxTrees^{|I|}$ projections (in time $O(|S|^2|A||\Omega|maxTrees^{|I|})$) that correspond to the number of all possible contributions of subtree $\eta(\delta, o)$. Given the fact that $\sum_i |\Omega_i| >> |I|$, the complexity of this step is negligible in comparison to the complexity of the exhaustive backup.
- finally, the exact value of joint policy-tree δ is,

$$f(b, \delta) = \textstyle\sum_{s \in S} b(s)R(s, \alpha(\delta)) + \sum_{o \in \Omega} g_{b,o}^{\delta} \tag{4}$$

It is fairly easy, however, to demonstrate that the solution that consists in selecting, for each joint observation the subtree with the highest contribution, will lead in general to a non-valid joint policy-tree. Nevertheless, the good news is that the exact value of non-valid joint policy-trees can be seen as a heuristic estimate of the exact value of valid joint policy-trees. More generally, joint policy-trees of the underlying MPOMDP can be used to define an upper-bound estimate of either partial or complete valid joint policy-trees.

Indeed, the heuristic estimate of a joint policy-tree is based on the decomposition of the evaluation function (Equation 1) into two estimates. The first estimate, $g(b, \delta)$, is the exact value coming from subtrees selected in accordance with $(c2)$, *i.e.* valid subtrees. A non-valid subtree is therefore a subtree that does not satisfy $(c2)$. The second estimate, $h(b, \delta)$, is the upper-bound value of the remaining subtrees selected in the order of their contribution, *i.e.* non-valid subtrees. We introduce Ω^1 as the set of joint observations that leads to a valid subtree and Ω^2 the remaining joint observations which means $\Omega = \Omega^1 \cup \Omega^2$. This allows us to decompose the heuristic estimate of any joint policy-tree δ for a given belief b into the exact contribution coming from joint observations Ω^1, and the upper-bound of the completion Ω^2:

$$\bar{f}(b, \delta) \quad = \quad \underbrace{b \cdot r_{\alpha(\delta)} + \sum\nolimits_{o \in \Omega^1} g^{\delta}_{b,o}}_{g(b,\delta)} + \underbrace{\sum\nolimits_{o \in \Omega^2} \bar{g}^{b}_{\alpha(\delta),o}}_{h(b,\delta)} \qquad (5)$$

where $\bar{g}^{b}_{\alpha(\delta),o} = \max_\delta g^{\delta}_{b,o}$ and $r_{\alpha(\delta)}$ is a vector of $R(s, \alpha(\delta))$ for all s *i.e.,* $r_{\alpha(\delta)}(s) = R(s, \alpha(\delta))$. Of course, the heuristic estimate $\bar{f}$ is an admissible heuristic function. Indeed, for any joint policy-tree δ and belief b the following holds by construction: $\bar{f}(b, \delta) \geq f(b, \delta)$.

3.3.2 A Branch & Bound Method

In the following, we describe a straightforward heuristic solution to the problem of determining the best joint policy-tree given a belief state and subtrees. Our complete algorithm, that includes important enhancements to its straightforward counterpart, is discussed later. As suggested by [8], since A* can be viewed as a special case of Branch and Bound algorithm (B&B), one can resort our proposed approach to Branch and Bound instead.

Similarly to POMDP incremental pruning techniques, the heuristic splits up the selection of the best joint policy-tree into the selection of several subtrees one for each joint observation $o \in \Omega$. Unfortunately, as already mentioned a DEC-POMDP is a MPOMDP with additional constraints. Thus, in order to satisfy these constraint requirements, we combine the incremental pruning mechanisms to a heuristic search subroutine. The resulting heuristic method is able to efficiently walks through the search tree that represents the space of joint policy trees. As illustrated in the section of the search tree Figure 1, each node in the search tree is either a *root* node δ_0, an *internal* node δ_1, or a *leaf* node δ_2. A root node of a search tree is initialized with a joint policy-tree δ, where $\alpha(\delta)$ can be any of the joint actions $a \in A$ and $\eta(o, \delta)$ be the subtree with the highest contribution no matter if it is a valid subtree or not. Each internal node at depth h ($h \geq 1$) of the search tree represents a joint policy-tree δ where only $(h - 1)$ valid subtrees have been attached at δ's leaf nodes (not to be confused with the leaf nodes of the search tree). Each joint policy-tree at leaf node of the search tree, where valid subtrees have been attached, is a valid joint policy-tree. Notice that during the course of the search, leaf nodes of the the search tree that do not represent a valid joint policy-tree turn out to be either the root node or future internal nodes of the search tree.

What PBIP does, is evaluate leaf nodes of the search tree, select the node with the highest heuristic estimate, and expand this node, thus descending one step further in the search tree. The search then proceeds by expanding joint policy-trees δ at leaf nodes of the search tree in descending order of $\bar{f}$-values, until the best solution is identified. Expanding a leaf node consists in expanding the attached joint policy-tree δ, which means: building all possible joint policy-trees that extend δ by replacing *one* non-valid subtree by *one* valid subtree; then making the resulting joint policy-trees as child nodes of δ. When a search tree has been completely explored, the heuristic starts a new search tree with another joint action $\alpha(\delta)$, until all joint actions have been selected. The resulting best joint policy-tree is then kept in memory. One can prioritize search trees in decreasing order of the heuristic estimate of their root nodes and prune all partially expanded joint policy-trees with heuristic estimates less or equal to the lower bound. The exact value of the joint policy-tree, that is both completely expanded (all subtrees are valid) and the current best joint policy-tree (that yields the highest exact value), can be set as the lower bound and denoted $\underline{f}(b)$.

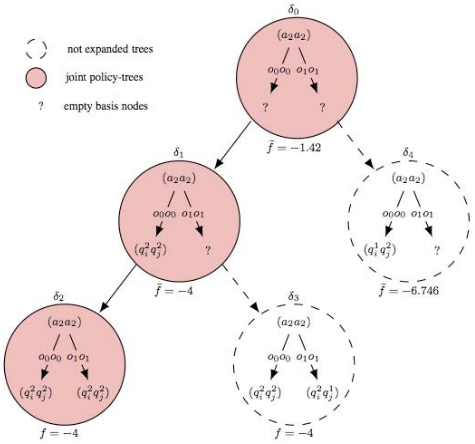

Figure 2: A section of PBIP search tree.

4. PBIP ALGORITHM

The heuristic method discussed above suffers from three major drawbacks: computation overhead, memory overhead, and inefficiency of the pruning strategy. To overcome these drawbacks, we suggest additional optimizations.

4.1 Reducing computation overhead

First of all, the above heuristic requires checking whether or not $(c2)$ and $(c1)$ are satisfied for each joint policy-tree δ. However, checking $(c2)$ incurs significant computation overhead since it requires in the worst case a time complexity $O(n \cdot |\Omega^1| \cdot |Q^{t-1}|)$, where a single subtree requires $O(n \cdot |\Omega^1|)$ operations. To handle this issue we rely on the fact that only a few non-valid subtrees of a joint policy-tree δ have to be replaced by valid subtrees to build a valid joint policy-tree as illustrated in the example below.

EXAMPLE 1. *Back to the example of Figure 1, given two valid subtrees of* δ, $\eta(\delta, (o_0, o_0)) = (q_i^2, q_j^2)$ *and* $\eta(\delta, (o_1, o_1)) = (q_i^1, q_j^1)$. *Then, using* $(c2)$ *it turns out that* $\eta(q_i, o_0) = q_i^2$ *and* $\eta(q_i, o_1) = q_i^1$ *for agent i;* $\eta(q_j, o_0) = q_j^2$ *and* $\eta(q_j, o_1) = q_j^1$ *for agent j. Therefore, the complete joint policy-tree* δ *is built as follows:* $\eta(\delta, (o_0, o_1)) = (\eta(q_i, o_0), \eta(q_j, o_1)) = (q_i^2, q_j^1)$.

Doing so permits us to reduce the computation overhead since $(c2)$ is checked only for a small number of subtrees. As discussed in the following, even for those subtrees it is not always required to check $(c2)$. More formally, we introduce the *basis joint policy-tree* as the smallest partial joint policy-tree, where some arcs have been removed, that is sufficient to recover the complete joint policy-tree. The set of joint observations, which tag the arcs of a basis joint policy-tree is referred to as the set of *basis joint observations* and denoted Ω_B. The nodes labeled with subtrees executed after receiving basis joint observations are called basis nodes. In Figure 1, the joint policy-tree, where arcs $\{(o_0, o_1), (o_1, o_0)\}$ are removed from δ, is a basis joint policy-tree, its basis joint observations and basis nodes are sets $\{(o_0, o_0), (o_1, o_1)\}$ and $\{(q_i^2, q_j^2), (q_i^1, q_j^1)\}$, respectively. While it is fairly easy to find a joint policy-tree using a basis joint policy-tree, one may ask how to determine a basis joint policy-tree. One simple way to determine a basis joint policy-tree is first to build the set of basis joint observations Ω_B then select a subtree for each basis leaf nodes. We build Ω_B by progressively adding a new joint observation $o \in \Omega$, which is *component-wise* different or simply different from joint observations already added into Ω_B, *e.g.* joint observations (o_0, o_0) and (o_1, o_1) are component-wise different while (o_1, o_0) and (o_1, o_1) are simply different since their first components are equal. We add joint observations that are component-wise different first then complete with simply different joint observations until Ω_B is a basis. The nice property of the idea of 'working with basis joint policy-trees' lies in the fact that it reduces the number of times $(c2)$ has to be checked. Indeed, $(c2)$ is satisfied for joint observations that are component-wise different. In addition, we can prove that the number of basis nodes κ is quite small, $\kappa := \max_{i \in I} |\Omega_i|$ (by construction of Ω_B). To get a better insight of the basis set, one can look at Ω_B as the smallest set of joint observations where each individual observation is included in at least one joint observation of that set. In particular, if all agents have the same number of individual observations, we can forget about $(c2)$, *i.e.* joint observations that are component-wise different define the basis set Ω_B.

4.2 Reducing space complexity.

One major drawback with either A*-style heuristics or B&B-like techniques is that in the worst case they remember an exponential number of nodes. Indeed, in both, *branching rules* to be applied can be seen as subdivision of a part of the search space. Generating *subspaces* are then kept in memory. Unfortunately, this incurs considerable memory requirement. To cope with this, we restrict the set of possible nodes at a given depth of the search tree, as illustrated in Figure 2.

To do so, our search tree is built such that joint policy-trees that can be attached at leaf nodes are all possible assignments of subtrees to the *same* basis node. This idea is motivated by the observation that such joint policy-trees differ only from the last subtree assigned to them, see for example δ_1 and δ_4. As a result, they are processed in decreasing order of the contribution of the last subtree assigned to them, *e.g.*, (q_i^2, q_j^2) and (q_i^1, q_j^2) respectively. Moreover, contributions of subtrees are stored in a priority stack that will maintain subtrees in descending order. Then, to expand leaf node δ_1, we only need to build the best child node δ_2, *i.e.* basis joint policy-tree where the last assigned subtree (q_i^2, q_j^2) yields the highest contribution. Indeed, the remaining child nodes will be progressively built in decreasing order of the contribution of the subtree assigned to them if necessary. A straightforward implementation of PBIP's search tree will results in a linear memory space $O(\kappa)$. In particular, our PBIP implementation (see subsection 4.4) only needs to keep one node that is progressively updated.

Since the space complexity plays a major role in B&B performances, we compare PBIP to B&B techniques with different search strategies of selecting the next node: Best First (BeFS); Breadth First (BFS) and Depth First (DFS). Indeed, the strategy for selecting the next node to process usually reflects a trade-off between keeping the number of processed nodes in the search tree low, and staying within the memory capacity of the computer used. Although the DFS strategy is slightly better than BeFS and clearly better than BFS, not surprisingly PBIP is superior to all of them in all tested benchmarks. The reason turns out to be that, PBIP shares common skills with BeFS and DFS.

4.3 Improving the pruning strategy

Using basis joint policy-trees, we are able to improve significantly the pruning strategy by means of exploiting $\bar{f}$'s properties. Let us denote $\textsc{Succ}(\delta)$ the set of *successor* joint policy-trees of a joint policy-tree δ, *i.e.* child and brother nodes of δ built after δ. For example in Figure 2, the successor joint policy-trees of δ_1 are $\{\delta_2, \delta_3, \delta_4\}$. The admissibility of the heuristic function $\bar{f}$ and the order in which PBIP processes joint policy-trees allow us to state the following results.

LEMMA 1. *Let δ be a joint policy-tree. Then, the following holds: $\forall \delta' \in \textsc{Succ}(\delta)\colon \bar{f}(b, \delta) \geq \bar{f}(b, \delta')$.*

PROOF. We only need to prove this claim for child nodes, since by hypothesis δ yields a better heuristic estimate than its brother nodes include into $\textsc{Succ}(\delta)$. We proceed by induction. First, the basis case consists in proving that the next partial or complete joint policy-tree δ_1 that extends δ is such that: $\bar{f}(b, \delta) \geq \bar{f}(b, \delta_1)$. To do so, we need to look at what is new in δ_1 according to δ. The main difference lies in the fact that δ_1 has one more subtree assigned to a basis node leading from basis joint observation, *e.g.* o^1:

$$\begin{aligned} \bar{f}(b, \delta_1) &= \bar{f}(b, \delta) - \bar{g}^b_{\alpha(\delta), o^1} + g^{\delta_1}_{b, o^1} \\ &= \bar{f}(b, \delta) - \bar{g}^b_{\alpha(\delta_1), o^1} + g^{\delta_1}_{b, o^1} \quad (\alpha(\delta) = \alpha(\delta_1)) \\ &\leq \bar{f}(b, \delta) \end{aligned}$$

The last inequality holds since $\bar{g}^b_{\alpha(\delta_1), o^1} \geq g^{\delta_1}_{b, o^1}$ by definition of the upper-bound. By repeating this argument for any consecutive pair $(\delta_k, \delta_{k+1}) \in \textsc{Succ}(\delta) \times \textsc{Succ}(\delta)$, the following holds: $\bar{f}(b, \delta_k) \geq \bar{f}(b, \delta_{k+1}),\ \forall k = 1, \cdots, |\textsc{Succ}(\delta)|$. This proves the above Lemma. □

Intuitively, it is obvious that this property is a more strict requirement than *monotonicity* since $\textsc{Succ}(\delta)$ also includes δ's brother nodes. Using this property and the order PBIP processes joint policy-trees, we establish the following theorem. This theorem describes an efficient property used to avoid joint policy-trees with low heuristic estimate.

THEOREM 1. *Let δ be a joint policy-tree with heuristic estimate $\bar{f}(b, \delta)$, and δ' the current best joint policy-tree. If $\bar{f}(b, \delta) \leq f(b, \delta')$, then the following holds:*

$$f(b, \delta') \geq f(b, \delta_k), \quad \forall \delta_k \in \textsc{Succ}(\delta).$$

PROOF. In accordance with Lemma 1 we have,

$$\bar{f}(b, \delta) \quad \geq \quad \bar{f}(b, \delta_k), \ \forall \delta_k \in \textsc{Succ}(\delta)$$

Then, the following holds:

$$\begin{aligned} f(b, \delta') &\geq \bar{f}(b, \delta) && \text{(by hypothesis)} \\ &\geq \bar{f}(b, \delta_k),\ \forall \delta_k \in \textsc{Succ}(\delta) && \text{(Lemma 1)} \\ &\geq f(b, \delta_k),\ \forall \delta_k \in \textsc{Succ}(\delta) && \text{(upper-bound def.)} \quad \square \end{aligned}$$

This theorem states only that if a joint policy-tree δ has a heuristic estimate less or equal to the current lower-bound, PBIP does not build subspace $\textsc{Succ}(\delta)$. This strategy improves the pruning strategy used in A*-style algorithms since it does not require all successor nodes $\textsc{Succ}(\delta)$ to be generated and evaluated. From a practical point of view, if δ's heuristic estimate is less or equal to the current lower bound (*backtracking condition*), then there are two possibilities: (a) if δ's parent node is the root node, then the search terminates; (b) otherwise, the search backtracks to δ's parent node and tests the backtracking condition.

Algorithm 1 PBIP subroutines.

```
 1: procedure SEARCH(($b$, $a$, $\delta^t$, $\Omega_B$, $Q^{t-1}$, $Select^t$))
 2:     $open \leftarrow EmptyStack$, $k \leftarrow 0$
 3:     $\alpha(\delta) \leftarrow a$
 4:     $\forall(o, \eta(\delta, o)) \in \Omega \times Q^{t-1}$: compute $g^{\delta}_{b,o}$, $\bar{g}^{b}_{a,o}$

 5:     EXPAND                                          ▷ initialize the search tree
 6:     while $open \neq EmptyStack$ do
 7:         $(o^k, \eta(\delta, o^k)) \leftarrow open$.PEEK
 8:         if $\underline{f}(b) < \bar{f}(b, \delta)$ then
 9:             if $o^k = o^\kappa$ then
10:                 if $f(b, \delta) > \underline{f}(b)$ and $\delta \notin Select^t$ then
11:                     $\delta^t \leftarrow \delta$
12:                     $\underline{f}(b) \leftarrow f(b, \delta^t)$
13:                 EXPLORE              ▷ select a new subtree $\eta(\delta, o^k)$
14:                 else EXPAND           ▷ assign a subtree $\eta(\delta, o^{k+1})$
15:             else BACKTRACK        ▷ backtrack if pos. to $\eta(\delta, o^{k-1})$
```

```
 1: procedure EXPAND
 2:     $k \leftarrow k + 1$                              ▷ go forward to $\eta(\delta, o^{k+1})$
 3:     $\eta(\delta, o^k) \leftarrow$ EXPLORE
 4:     $open$.PUSH$(o^k, \eta(\delta, o^k))$
 5: procedure BACKTRACK
 6:     if $open$.ISNOTEMPTY then
 7:         $(o^k, -) \leftarrow open$.POP
 8:         $\eta(\delta, o^k) \leftarrow -1$          ▷ reinitialize the pointer position
 9:         if $o^k \neq o^1$ then
10:             $k \leftarrow k - 1$                 ▷ go backward to $\eta(\delta, o^{k-1})$
11:             EXPLORE
12: procedure EXPLORE
13:     INCREMENT$(\eta(\delta, o^k))$
14:     if $\eta(\delta, o^k) > |Q^{t-1}|$ then
15:         BACKTRACK                    ▷ backtrack if pos. to $\eta(\delta, o^{k-1})$
16:     return $\eta(\delta, o^k)$
```

EXAMPLE 2. *Figure 2 shows the progress of* PBIP *to find the best joint policy-tree for the search tree associated with joint action (a_2, a_2). The first node to be built starting from root node (δ_0) is δ_1, because it yields the highest heuristic estimate -4. This node in turn generates only one child node,* i.e. *δ_2. Since the resulting joint policy-tree has non-empty basis nodes,* PBIP *computes its exact value -4, and uses it as a lower bound. Then,* PBIP *backtracks to node δ_1. Because δ_1's heuristic estimate -4 is equal to the current lower bound,* PBIP *does not build its successor nodes δ_3 and δ_4. Then, the search terminates since δ_1's parent node (δ_0) is the root node of the search tree. Finally, joint policy-tree δ_2 is the best one for this search tree.*

4.4 Specific implementation

In the same vein as MBDP, PBIP combines the bottom-up and top-down heuristic methods as described in Algorithm 2. However, what sets PBIP apart from MBDP is its ability to avoid the exhaustive generation of all possible joint policy-trees at each iteration. Indeed, a single iteration of PBIP can be summarized in the following steps: first, the algorithm sets the joint policy-tree from the last iteration Q^{t-1}. Next, it chooses top-down heuristics from the portfolio H in order to generate $maxTrees$ number of belief states. Then, it uses subroutine SEARCH to determine the set of best joint policy-trees $Select^t$ for the set of generated belief states. Finally, at iteration T, the best joint policy-tree with respect to the initial belief state b_0 is returned.

Algorithm 2 Point based incremental pruning

```
 1: procedure PBIP(($maxTrees$, $T$, $H$))
 2:     $Select^1 \leftarrow$ initialize all depth-1 joint policy-trees
 3:     for all $t = 2, \cdots, T$ do
 4:         $Q^{t-1} \leftarrow Select^t$ and $Select^t \leftarrow \emptyset$
 5:         for all $k = 1, \cdots, maxTrees$ do
 6:             chooses $h \in H$ and generate belief $b$
 7:             $\delta^t \leftarrow null$ and $\underline{f}(b) \leftarrow -\infty$
 8:             for all $a \in A$ do
 9:                 SEARCH$(b, a, \delta^t, \Omega_B, Q^{t-1}, Select^t)$
10:             add best joint policy-tree $\delta^t$ to $Select^t$
11:     select the optimal joint policy-tree $\delta^{\star T}$ from $Select^T$
12:     return $\delta^{\star T}$
```

In the following, we draw attention to a single search tree of PBIP with the following entries: a belief state b; a joint action a, the current best joint policy-tree δ^t, the set of basis joint observations Ω_B, subtrees from the last iteration Q^{t-1}, and finally current best joint policy-trees $Select^t$ (Algorithm 1). Exact and upper bound contributions of subtrees in Q^{t-1} are computed and stored in a priority queue. We use $\eta(\delta, o)$ as either a subtree or its index position in the priority queue *interchangeably*. Therefore, PBIP's SEARCH subroutine can be summarized in the following steps:

First of all, SEARCH initializes an *open list* '*open*' that will contain the expanded nodes. Second, it expands the current leaf node of the search tree that yields the highest heuristic estimate (see procedure EXPAND, lines 1-5). Then, SEARCH goes through the main loop. Each loop consists in evaluating the heuristic estimate of the current either partially or completely expanded joint policy-tree (lines 6-19). If the heuristic estimate is less or equal to the lower bound (line 8), then SEARCH backtracks (if possible) to the last expanded joint policy-tree (see procedure BACKTRACK, lines 6-15). Otherwise, it updates the best joint policy-tree if it is complete and its exact value is greater than the lower bound (lines 9-13) and explores any remaining alternatives (see procedure EXPLORE, lines 16-22). If not, SEARCH backtracks and expands the current joint policy-tree δ (see procedure EXPAND, lines 1-5) by assigning a subtree $\eta(\delta, o^{k+1})$. Note that, the new best joint policy-tree has to be *component-wise* different from those already selected (line 10). This enables us to avoid the problem of selecting identical policy-trees for each agent. Finally, the search ends when the root node of the search tree has been completely explored.

5. THEORETICAL PROPERTIES

In this section, we introduce some additional theoretical properties that guarantee PBIP performances, including completeness, optimality, and complexity.

THEOREM 2. PBIP *heuristic method is complete.*

PROOF. PBIP will eventually terminate in the worst case after enumerating all possible joint policy-trees from the current iteration given the set of subtrees, which means after visiting the entire exponential space of joint policy-trees at each iteration. The best solution δ for a given belief b will be the one that yields the highest exact value $f(b, \delta)$. □

THEOREM 3. PBIP *heuristic method is optimal with respect to a given belief and the set of subtrees.*

PROOF. One can look at PBIP as an A*-style algorithm. Indeed, PBIP visits complete joint policies in best-first fashion following an admissible heuristic function $\bar{f}$. Then, if PBIP terminates and returns a joint policy-tree δ, the convergence property of A* and the admissibility of the heuristic $\bar{f}$ guarantee the optimality of the solution with respect to a given belief b and subtrees. □

Algorithm	MBDP		IMBDP		MBDP-OC		PBIP		
HORIZON (T)	AEV	CPU (sec.)	AEV	CPU (sec.)	AEV	CPU (sec.)	AEV	CPU (sec.)	%
MABC problem $maxTrees = 3$									
100	90.29	0.190	N.A.		N.A.		90.29	0.060	39
1000	900.29	2.270	N.A.		N.A.		900.29	0.940	38
10000	9000.29	51.93	N.A.		N.A.		9000.29	38.48	38
MA-TIGER problem $maxTrees = 20$									
10	12.5 ± 2.9	67.1 ± 0.13	N.A.		N.A.		13.6 ± 1.11	5.55 ± 0.88	7.83
20	25.8 ± 2.1	159 ± 0.16	N.A.		N.A.		26.8 ± 1.50	15.1 ± 2.68	12.4
50	73.9 ± 0.7	438 ± 0.86	N.A.		N.A.		74.2 ± 2.76	45.3 ± 6.49	15.3
100	149	901	N.A.		N.A.		147 ± 5.84	94.2 ± 9.53	12.9
COOPERATIVE BOX-PUSHING problem $maxTrees = 3$ $maxObs = 3$									
10	N.A.		99.59	7994.6	89.94	7994.4	102.5 ± 6.46	11.8 ± 3.61	10.9
20	N.A.		102.6	17031	135.0	17194	198.0 ± 10.8	25.0 ± 4.21	10.3
50	N.A.		82.99	44708	272.70	44210	422.2 ± 20.8	61.6 ± 10.6	9.07
100	N.A.		73.88	89471	440.08	90914	786.4 ± 31.9	113 ± 20.8	9.62
AEV = average expected value			N.A. = not applicable			% = pourcentage of expanded nodes			

Table 1: Performances of all PBIP, MBDP, IMBDP and MBDP-OC.

Notice that in the worst case PBIP still has an exponential time complexity with respect to the number of agents and κ: in the best case, however, its time complexity is $O(\kappa)$. The proof of these results relies on the following observations: first, since PBIP eventually enumerates all possible joint policies at a given iteration, in the worst case its time complexity is also exponential; but in the best case, PBIP requires only κ steps to build the first complete joint policy-tree. Nevertheless, its overall time complexity can be influenced by the pre-computing step that requires roughly $O(|S^2||A||\Omega|maxTrees^{|I|})$ operations. Although PBIP is based on A* algorithm, it incurs negligible memory overhead. Indeed, the subroutine SEARCH has a linear space complexity $O(\kappa)$, *i.e.* the length of the longest path from the root node to the best joint policy-tree. This is possible since we compute all contributions of subtrees for all joint action and joint observation pairs before the search starts. As a result, the overall space complexity of PBIP is $O(|A||\Omega|maxTrees)$.

6. EXPERIMENTS

As MBDP and its extensions are known to perform better than other approximate algorithms such as point based dynamic programming (PBDP), we compare PBIP only to MBDP, IMBDP and MBDP-OC. Comparison is made according to DEC-POMDP benchmarks from the literature: the multi-access broadcast channel (MABC) problem [6], the multi-agent tiger problem [7] and the COOPERATIVE BOX-PUSHING problem [11]. The COOPERATIVE BOX-PUSHING problem provides an opportunity for testing the scalability of different algorithms. We executed the algorithms using the same parameter controls of the selection of heuristics from the portfolio, $maxTrees$, *recursion depth* (set at 1) and $maxObs$. Except that parameter $maxObs$ is only used for IMBDP and MBDP-OC solvers.

Table 1, Figure 3 and Figure 4 present our experimental results. For each problem and method we report: *The average expected value* (AEV.) – this is because both MBDP and PBIP are partly randomized then selected belief states and thus the quality of the resulting solution may differ from one trial to another. *Execution time* – we report only the CPU time (in seconds) since all algorithms are coded in JAVA, properly optimized and run on the same machine. *Expanded nodes* (%) – where $\% = 100 \times \frac{N_{\text{PBIP}}}{N_{\text{MBDP}}}$, where N_i denotes the number of expanded joint trees by algorithm i. This value helps to estimate the benefits of our pruning heuristic.

6.1 Discussion

Table 1 presents performance results of all the algorithms on the three benchmarks. As we already notice, both PBIP and MBDP algorithms are likely to find the same solution, since they differ only in the way they select the best joint policy-trees. Our results clearly indicate that PBIP is faster than MBDP, IMBDP or MBDP-OC and scales up better. For problem with small number of observation both PBIP and MBDP provide approximately the same solution quality, but the time required by PBIP to reach the desired solution is always faster. For example, the MABC problem for horizon 1000 is solved (approximately) in 560 seconds with PBIP whereas MBDP requires 1094 seconds to solve it. The most impressive results rely on the percentage of expanded joint policy-trees. For instance, in the MA-TIGER problem at horizon 10, PBIP expands only 7.83% of what would be expanded by a brute force approach. However, when the horizon increases in general this percentage also increases and then stabilizes depending on the problem and parameter $maxTrees$. Table 1 via the COOPERATIVE BOX-PUSHING problem also presents a more focused comparison of the relative ability of the PBIP and MBDP extensions to scale up. Overall, it appears that IMBDP and MBDP-OC are slowed down considerably by the exhaustive backup computation and the management of the resulting joint policy-trees. PBIP, however, benefits from its ability to identify subspaces of joint policy-trees with low heuristic estimate, this enables it to focus only on a small space of relevant joint policy-trees. Even though PBIP does not include an observation compression subroutine, it solves the COOPERATIVE BOX-PUSHING up to 800 times faster than IMBDP and MBDP-OC, while providing a better solution quality, that is up to 1.78 times higher than the MBDP-OC's and up to 10 times higher than the IMBDP's. Notice that given the time requires to build policies using either IMBDP or MBDP-OC their results are not averaged, so the solution quality may be slightly different from one trial to another.

These differences become more pronounced as the parameter $maxTrees$ increases. As pointed out by [12], by increasing parameter $maxTrees$ one generally increases both solution quality and runtime. We illustrate the performance of MBDP and PBIP when one increases parameter $maxTrees$ for the MA-TIGER problem. On the right hand side, Figure 3 shows the runtimes of both MBDP and PBIP when parameter $maxTrees$ is increased. On the left hand side, it shows the solution value for different values of parameter $maxTrees$. The purpose of Figure 3 is therefore to show how both PBIP and MBDP perform for different values of parameter $maxTrees$. As illustrated, MBDP exceeds the time limit (1 hour) for $maxTrees = 35$ while PBIP is able to solve the MA-TIGER problem using $maxTrees = 100$ and more. Doing so enables PBIP to improve the solution quality of MA-TIGER problem. Nevertheless, even though PBIP avoids a large number of joint policy-trees, the benefits do not fully manifest in total execution time. For instance in Table 1, for MABC problem at horizon 10000, PBIP explores only 38% of the joint policy-trees, however, it took 38.48 seconds to find the best solution, while MBDP required 51.93 seconds. This is mainly because computing exact and upper-bound estimates of each subtree incurs considerable computation overhead. Of course, managing those estimates requires a number of costly operations including: subtrees re-ordering, checking the validity of a given joint policy-tree. We believe that those issues would be overcome by an appropriate data structure and/or imple-

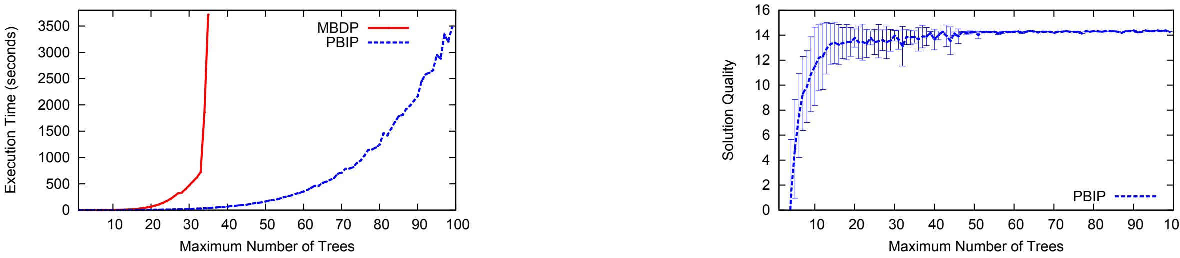

Figure 3: Performances for MA-TIGER problem with $T = 10$.

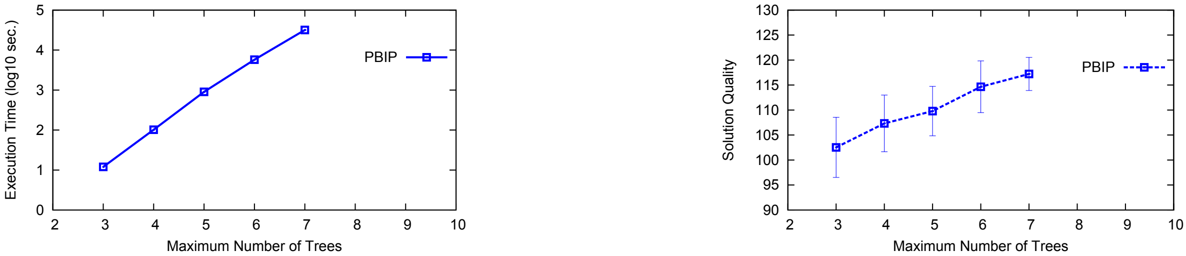

Figure 4: PBIP's performances for COOPERATIVE BOX-PUSHING problem with $T = 10$.

mentation. To confirm the scalability of PBIP, we increase parameter $maxTrees$ up to 7, in the COOPERATIVE BOX-PUSHING problem (see Figure 4). We only report the PBIP's performances since IMBDP and MBDP-OC quickly run out of time (10 hours). Overall, it appears that although PBIP scales up better than other approximate DEC-POMDP solvers, the impact of increasing $maxTrees$ parameter is not negligible. For example, if $maxTrees = 7$ PBIP requires approximately 8 hours while requiring only one hour and half when using $maxTrees = 6$. Roughly, the running time grows exponentially with parameter $maxTrees$ while the average solution quality grows only about 5% for each additional trees.

7. CONCLUSION AND FUTURE WORK

We have presented PBIP, the first heuristic method that circumvents the problem of generating and evaluating all possible joint policy-trees at each iteration of DEC-POMDP solvers. This algorithm is based on two observations: (1) most joint policy-trees resulting from the exhaustive backup yield low heuristic estimate; (2) the DEC-POMDP policy space is a subspace of the MPOMDP policy space. This insight, allows PBIP to conduct a search towards the most promising joint policies, and avoids the irrelevant ones. Experiment results show orders of magnitude improvement in the performance of all leading algorithms including MBDP and its extensions. Although the proposed approach has been illustrated within the MBDP framework, PBIP can easily be used to scale up others DEC-POMDP solvers for either finite or infinite cases. Currently, we are tracking others similarities between DEC-POMDP and POMDP models so as to reduce the algorithmic gap that separates these two closely related problems and hopefully to enable us to scale up to real-life applications.

Acknowledgments

We would like to thank Sven Seuken and Alan Carlin for making available their MBDP, IMBDP and MBDP-OC codes for solving finite horizon DEC-POMDPs. We also thank Hamid R. Chinaei and Camille Besse for the helful comments on this work.

References

R. E. Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 1957.

D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, 27(4), 2002.

C. Besse and B. Chaib-draa. Parallel rollout for online solution of dec-pomdps. In *FLAIRS Conference*, pages 619–624, 2008.

A. Boularias and B. Chaib-draa. Exact dynamic programming for decentralized pomdps with lossless policy compression. In *ICAPS*, pages 20–27, 2008.

A. Carlin and S. Zilberstein. Value-based observation compression for DEC-POMDPs. In *AAMAS*, 2008.

E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, pages 709–715, 2004.

R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *AAAI*, pages 133–139, 2005.

D. S. Nau, V. Kumar, and L. Kanal. General branch and bound, and its relation to a* and ao*. *Artif. Intell.*, 23(1):29–58, 1984.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, 2003.

Z. Rabinovich, C. V. Goldman, and J. S. Rosenschein. The complexity of multiagent systems: the price of silence. In *AAMAS*, pages 1102–1103, 2003.

S. Seuken and S. Zilberstein. Improved Memory-Bounded Dynamic Programming for DEC-POMDPs. In *UAI*, 2007.

S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *IJCAI*, pages 2009–2015, 2007.

D. Szer and F. Charpillet. Point-based dynamic programming for DEC-POMDPs. In *AAAI*, pages 16–20, July 2006.

D. Szer, F. Charpillet, and S. Zilberstein. Maa*: A heuristic search algorithm for solving decentralized POMDPs. In *UAI*, pages 568–576, 2005.

P. Varakantham, J. Marecki, M. Tambe, and M. Yokoo. Letting loose a spider on a network of POMDPs: Generating quality guaranteed policies. In *AAMAS*, May 2007.