Three New Algorithms to Solve N-POMDPs

Yann Dujardin
CSIRO
yann.dujardin@csiro.au

Tom Dietterich
School of EECS
Oregon State University
tgd@oregonstate.edu

Iadine Chadès
CSIRO
iadine.chades@csiro.au

Abstract

In many fields in computational sustainability, applications of POMDPs are inhibited by the complexity of the optimal solution. One way of delivering simple solutions is to represent the policy with a small number of $\alpha$-vectors. We would like to find the best possible policy that can be expressed using a fixed number $N$ of $\alpha$-vectors. We call this the $N$-POMDP problem. The existing solver $\alpha$-min approximately solves finite-horizon POMDPs with a controllable number of $\alpha$-vectors. However $\alpha$-min is a greedy algorithm without performance guarantees, and it is rather slow. This paper proposes three new algorithms, based on a general approach that we call $\alpha$-min-2. These three algorithms are able to approximately solve $N$-POMDPs. $\alpha$-min-2-fast (heuristic) and $\alpha$-min-2-p (with performance guarantees) are designed to complement an existing POMDP solver, while $\alpha$-min-2-solve (heuristic) is a solver itself. Complexity results are provided for each of the algorithms, and they are tested on well-known benchmarks. These new algorithms will help users to interpret solutions to POMDP problems in computational sustainability.

Introduction

Partially Observable Markov Decision Processes (POMDP) provide a powerful tool for sequential decision-making under uncertainty. In computational sustainability, POMDPs are increasingly being applied to help solve important problems in biodiversity conservation. For example, rules of thumb were derived using POMDPs to best manage and monitor cryptic species such as the critically endangered Sumatran Tigers (Chadès et al. 2008; McDonald-Madden et al. 2011), or the invasive parasitic plant species branched broomrape with long-lasting microscopic seeds (Regan, Chadès, and Possingham 2011). More recently, POMDPs were employed to adaptively manage renewable natural resources (Fackler and Pacifici 2014; Springborn and Sanchirico 2013) and migratory shorebirds threatened by the uncertain consequences of sea level rise (Nicol et al. 2015; 2013). Despite this growing interest, POMDP solutions are often too complex to be analyzed and communicated efficiently to stakeholders, which prevents the deployment of POMDP solutions (Tulloch et al. 2015; Nicol and Chadès 2012). Until recently, POMDP algorithms have focused on providing near-optimal solutions regardless of the number of $\alpha$-vectors required. Indeed, current algorithms require many $\alpha$-vectors even when solving small problems (Poupart, Kim, and Kim 2011). Here, we contribute to bridging the gap between POMDP solvers and their end users by providing approximate solutions that can be represented with a small number of $\alpha$-vectors. The recently-published algorithm $\alpha$-min was a first attempt at approximately solving finite-horizon POMDPs given a limit $N$ on the number of $\alpha$-vectors (Dujardin, Dietterich, and Chadès 2015). We call this problem $N$-POMDP. $\alpha$-min is however a greedy algorithm and does not provide satisfying performance guarantees. Here, we propose three new algorithms, based on a common approach, $\alpha$-min-2, that directly searches for subsets of $\alpha$-vectors of size $N$. $\alpha$-min-2-fast (heuristic) and $\alpha$-min-2-p (with performance guarantees) are designed to work in complement of an existing POMDP solver, and $\alpha$-min-2-solve (heuristic) is a solver itself. The new approach allows to solve infinite-horizon or finite-horizon problems and is able to outperform $\alpha$-min.

The paper is organized as follows. First, we introduce POMDPs and the $\alpha$-min principles. Next we formalize the $N$-POMDP problem. Then we present the $\alpha$-min-2 principle and the three related algorithms: $\alpha$-min-2-fast, $\alpha$-min-2-p and $\alpha$-min-2-solve. Finally we apply the three algorithms to well-known benchmark problems and discuss the results.

POMDPs, N-POMDPs and the $\alpha$-min principle

POMDPs and related work

POMDPs are models of sequential decision-making when the decision-maker does not have complete information about the current state of the system (Sigaud and Buffet 2013). Formally, a discrete finite-horizon POMDP is specified as a tuple $\{S, A, O, \tau_o, \Omega_a, R, H\}$, where

- $H = \{0, \ldots, T - 1\}$ are called “time steps”, $T \in \mathbb{N}$ is called the time horizon.
- $S, \forall t \in H, s_t \in S$ is the state of the system at $t$.
- $A, \forall t \in H, a_t \in A$ is the action taken at $t$.
- $O, \forall t \in H, z_t \in O$ is the observation at $t$. 

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
• \( \tau_a \) is the transition matrix for action \( a \). Its elements are 
\[ \tau_a(s_t, s_{t+1}) \]
• \( \Omega_a \) is the observation matrix for action \( a \). Its elements are 
\[ \Omega_a(s_{t+1}, \omega_{t+1}) \]
• \( R \) is the reward matrix. Its elements are 
\[ R(a_t, s_t) \]

For sake of clarity, we define the following notation:

• For action \( a \in A \) and for observation \( \omega \in O \), let 
\[ M_{a,\omega} \] be the matrix of dimension \( S \times S \) such that 
\[ M_{a,\omega}(s_{t+1}, s_t) = \Omega_a(\omega, s_{t+1}) \tau_a(s_{t+1}, s_t) \]
• For every \( a \in A \), the vector \( r_a = R(a, \cdot) \) corresponds to 
the row of the matrix \( R \) corresponding to the action \( a \).

The optimal decision at time \( t \) may depend on the complete history of past actions and observations. Because it is neither practical nor tractable to use the history of the action-observation trajectory to compute an optimal solution, belief states (also called beliefs), i.e., probability distributions over states, are used to summarize and overcome the difficulties of imperfect detection (Aström 1965). A POMDP can be cast into a fully observable Markov decision process defined over the continuous belief state space, \( B \).

Solving exactly a finite-horizon POMDP means finding an optimal policy \( \Pi_0 = \cup_{t \in H} \pi_t \), where \( \pi_t : B \rightarrow A \) maps belief states at time \( t \) to actions. \( \Pi_0 \) maximizes the expected sum of rewards \( E[\sum_{t \in H} r_{a_t} \cdot b_t] \) over the time horizon \( H \) (\( \cdot \) denotes the scalar product).

For each time step \( t \), for a given belief state \( b_t \) and a given policy \( \Pi_t = \cup_{t \in \{t, \ldots, T-1\}} \pi_t \), the expected sum 
\[ E[\sum_{t \in \{t, \ldots, T-1\}} r_{a_t} \cdot b_t] \] is also referred to as the value function \( V_t(\Pi_t)(b_t) \). A value function allows us to rank policies by assigning a real value to each belief \( b_t \). An optimal policy \( \Pi^*_t \) is a policy such that, \( \forall b_t \in B, \forall \Pi_t, V_t(\Pi^*_t)(b_t) \geq V_t(\Pi_t)(b_t) \).

Several policies can be optimal and share the same optimal value function \( V_t \), which can be computed using Bellman’s principle of optimality (Bellman 1957): 
\[
V_t(b_t) = \max_{a_t \in A} \left\{ r_{a_t} \cdot b_t + \sum_{\omega \in O} \pi_t(\omega \mid a_t, b_t) V_{t+1}(b_{t+1}) \right\}
\]

where each component \( b_{t+1}(s_{t+1}) \) of the new belief can be computed as follows:
\[
b_{t+1}(s_{t+1}) = \sum_{s_t, \omega_t} \pi_t(s_t, \omega_t) \tau_a(s_t, s_{t+1}) b_t(s_t)
\]

Equation 1 can be rewritten \( V_t = BL(V_{t+1}) \), where \( BL \) is the Bellman operator (Shani, Pineau, and Kaplow 2012), sometimes also called backup operator (Pineau, Gordon, and Thrun 2006). While various algorithms from the operations research and artificial intelligence literature have been developed over the past years, exact solving of POMDPs is intractable: finite-horizon POMDPs are PSPACE-complete (Papadimitriou and Tsitsiklis 1987) and infinite-horizon POMDPs are undecidable (Madani, Hanks, and Condon 2003).

Equation 1 can be solved by directly manipulating \( \alpha \)-vectors: (POMDPs and Sondik 1973). For every \( t \in H \), there exists a finite set \( \Gamma_t \) of vectors of dimension \( |S| \) (the so-called \( \alpha \)-vectors) that define entirely \( V_t \) such that \( |\Gamma_t| \) is minimal and
\[
\forall t \in H, \forall b_t \in B, V_t(b_t) = \max_{\alpha_t \in \Gamma_t} \alpha_t \cdot b_t.
\]

Given that \( \Gamma_{T-1} = \{ r_a \mid a \in A, r_a \text{ is not dominated} \} \), one can in theory build a set of \( \alpha \)-vectors defining the value function \( V_t \) from \( \Gamma_{t+1} \) at any time step \( t \in H' = \{0, \ldots, T-2\} \), because every \( \alpha_t \in \Gamma_t \) can be written:
\[
\alpha_t = [r_{a_t} + \sum_{a_{t+1} \in A} (\alpha_{t+1})^T M_{a_t, a_{t+1}}] T
\]
where \( a_t \in A \) and \( \alpha_{t+1} \) are elements of \( \Gamma_{t+1} \).

The set of \( \alpha \)-vectors \( G_t \) given by Equation 4 is such that \( \Gamma_t \subseteq G_t \). Note that some \( \alpha \)-vectors of \( G_t \) could be dominated by others and therefore do not belong to \( \Gamma_t \).

In most of exact algorithms, complexity when calculated is exponential in at least one component of the instance (Kaelbling, Littman, and Cassandra 1998). For example, a natural way to compute \( \Gamma_t \) from a given \( \Gamma_{t+1} \), is first to compute \( G_t \) entirely, and then prune the dominated vectors that are not useful for representing the value function. This approach has a complexity exponential in the number of observations \( |O| \) (Sigaud and Buffet 2013, sections 7.3 and 7.4). Conversely, the one-pass algorithm (Smallwood and Sondik 1973), the linear-support algorithm (Cheng 1988), and the relaxed-region algorithm (Cheng 1988) all try to generate \( \Gamma_t \) directly. In particular, the linear-support algorithm (Cheng 1988) systematically looks for new beliefs allowing to iteratively compute new vectors of \( BL(\Gamma_{t+1}) \). Although not proven by the authors, the complexity is then controlled by the number of explored beliefs, which in turn is at least exponential in \( |\Gamma_t| \). The witness algorithm (Kaelbling, Littman, and Cassandra 1998) has a running time exponential in \( |A| \).

N-POMDPs

Definition 1. An N-POMDP is a POMDP with an additional parameter \( N \) that defines the maximum size of any admissible policy represented by a set of \( \alpha \)-vectors at each time step (for the infinite-horizon case, we consider that there is only one time step).

Solving an N-POMDP means finding the best possible policy of size at most \( N \) at each time step (Problem 5).

\[
\max_{\Gamma_0, \ldots, \Gamma_{T-1} \in \Theta, s.t. \forall t \in H, |\Gamma_t| \leq N} V_{0, \Gamma}(b_0)
\]

where \( V_{0, \Gamma}(b_0) \) is the value function corresponding to the policy \( \Gamma = \cup_{t \in H} \Gamma_t \) for the initial belief \( b_0 \). \( \Theta \) is the set of all possible sets of \( \alpha \)-vectors. In the infinite case, we only consider one set \( \Gamma \) of \( \alpha \)-vectors.

Proposition 1. The N-POMDP problem is an NP-hard problem.

Proof. Because N-POMDPs concern both finite and infinite horizon cases, one just needs to prove the NP-hardness for one case. For the finite-horizon case, this is a direct consequence of the NP-hardness of the exact backup operation: in
A greedy approach starting with $\alpha_2$ would return the policy $\tilde{\Gamma} = \{\alpha_2, \alpha_1\}$ or $\tilde{\Gamma} = \{\alpha_2, \alpha_3\}$ with a maximum gap of $x$, which can be arbitrary large, while the optimum gap is 0 and corresponds to the policy $\bar{\Gamma}^* = \{\alpha_1, \alpha_3\}$.

The goal of the $\alpha$-min-2 algorithms is to solve Problem 6 by looking for the best combinations of $\alpha$-vectors instead of using the $\alpha$-min greedy algorithm.

The $\alpha$-min-2 algorithms

Given that good POMDP solvers already exist (Spaan and Vlassis 2005; Kurniawati, Hsu, and Lee 2008), a natural approximate approach to solve an N-POMDP is to first generate a good initial policy $\Gamma$ without any size limitation and then to select the best combination of $\alpha$-vectors of $\Gamma$ minimizing the gap between the initial policy and the new policy (Problem 8).

$$g^* = \min_{\tilde{\Gamma} \subseteq \Gamma, |\tilde{\Gamma}| \leq N} \max_{b \in B} [V(b) - \bar{V}(b)] \tag{8}$$

In doing so we hope that if $\Gamma$ is good enough, $\tilde{\Gamma}$ should be good enough too. This approach constitutes the main principle for both algorithms $\alpha$-min-2-fast (heuristic) and $\alpha$-min-2-p (approximate approach with performance guarantees). Unfortunately, $\alpha$-min-2-fast and $\alpha$-min-2-p are unable to solve N-POMDPs when no initial policy is given. If $\Gamma$ is not good enough or not available, one would like to solve Problem 5 directly. $\alpha$-min-2-solve aims to do that.

The $\alpha$-min-2-fast algorithm

Suppose we have at hand a policy $\Gamma$, and we want to solve Problem 8.

Let $s$ be a positive semi-definite function such as $s(\tilde{\alpha}, \alpha) = \max_{\alpha \in B(\alpha)} (\alpha \cdot b - \tilde{\alpha} \cdot b)$, $\alpha, \tilde{\alpha} \in \Gamma$, where $B(\alpha)$ is the subspace of $B$ where $\alpha$ dominates the other $\alpha$-vectors.

We have

$$\max_{b \in B} [V(b) - \bar{V}(b)] \leq \max_{b \in B} (\max_{\alpha \in \Gamma} \alpha \cdot b - \max_{\tilde{\alpha} \in \Gamma} \tilde{\alpha} \cdot b)$$

$$\leq \max_{\alpha \in \Gamma} \min_{\tilde{\alpha} \in \Gamma} (\alpha \cdot b - \tilde{\alpha} \cdot b)$$

$$\leq \min_{\alpha \in \Gamma} \max_{\tilde{\alpha} \in \Gamma} s(\tilde{\alpha}, \alpha)$$

Solving Problem 9 then provides an upper bound for Problem 8.

$$\min_{\tilde{\Gamma} \subseteq \Gamma, |\tilde{\Gamma}| \leq N} \max_{\alpha \in \Gamma} \min_{\tilde{\alpha} \in \Gamma} s(\tilde{\alpha}, \alpha) \tag{9}$$

for $N > 1$.

Problem 9 is very similar to the $k$-center clustering problem (Gonzalez 1985) but cannot be solved using classic algorithms because $s$ is not a distance.

However, Problem 9 can easily be modelled by a mixed integer linear program, Linear Program 10, with $|\Gamma|^2 0$-1 variables and constraints, where $s(\tilde{\alpha}, \alpha)$ are pre-calculated using linear programs with $|S|$ variables and $|\Gamma|$ constraints each.
where every 0-1 variable $x_α$ is such that $x_α = 1$ iff $α$ is in $Γ$ and $y_{α,α′}$ are 0-1 variables such that $y_{α,α′} = 1$ iff $α ∈ Γ$ and $α′ ∈ Γ$ minimizes $s(α, α′)$.

Linear Program 10 is a mixed integer linear programs with many 0-1 variables ($|Γ|^2$) and is consequently slow to solve. We propose a faster way to solve Problem 9, using pure integer linear programs and with only $|Γ|$ variables, within an arbitrary precision $p$ (Algorithm 1). The principle of Algorithm 1 is to perform a binary search on a decision version of Linear Program 10.

**Algorithm 1 $α$-min-2-fast**

1: procedure $\text{Fast}(Γ, p, N ≥ 1) $
2: \hspace{1cm} $ε^+ = ε_{ub}, ε^- = 0, δ = ε^+ - ε^-$
3: \hspace{1cm} while $δ > p$ do
4: \hspace{2cm} $ε = \frac{ε^+ + ε^-}{2}$
5: \hspace{2cm} for $α, α′ ∈ Γ$ do
6: \hspace{3cm} if $s_{α,α′} ≤ ε$ then $c_{α,α′} = 1$
7: \hspace{3cm} else $c_{α,α′} = 0$
8: \hspace{2cm} $C = \{c_{α,α′} | α, α′ ∈ Γ\}$
9: \hspace{2cm} $Γ ← \text{LP}_{FAST}(C, N)$
10: \hspace{1cm} if $\text{LP}_{FAST}(C, N)$ has no solution then $ε^+ = ε$
11: \hspace{1cm} else $ε^- = ε$
12: $ε^+ = ε_{ub}, ε^- = 0, δ = ε^+ - ε^-$
13: return $Γ, ε$

Here, $ε_{ub}$ is an upper bound of the solution value $ε$ of Problem 9 and $Γ$ (line 9) is the solution returned by $\text{LP}_{FAST}$, given that $α ∈ Γ$ $\iff x_α = 1$.

min $\sum_{α ∈ Γ} x_α$

s.t. $\sum_{α ∈ Γ} c_{α,α′} x_α ≥ 1, α′ ∈ Γ$

$\sum_{α ∈ Γ} x_α = N$

$x_α ∈ \{0, 1\}$

($\text{LP}_{FAST}$)

**Remark.** Linear Program $\text{LP}_{FAST}$ actually does not need any objective function since only the feasibility is checked in Algorithm 1. An objective function has been added to obtain a formal integer linear program.

**Proposition 2.** Algorithm 1 solves Problem 9 within precision $p$ in a finite number of iterations.

**Proof.** Computing a first upper bound $ε_{ub}$ of $g^*$ is easy: choose randomly $α_0 ∈ Γ$ and set $ε_{ub} = \max_{α ∈ Γ} s(α_0, α)$ (can be solved with linear programming). We have then $ε_{ub} ≥ \max_{α ∈ Γ} \min_{α′ ∈ Γ} s(α, α′), Γ ⊆ Γ$, $|Γ| ≥ 1$, so $ε_{ub} ≥ \min_{ε ∈ Γ, |Γ| ≤ N} \max_{α ∈ Γ} \min_{α′ ∈ Γ} s(α, α′)$ for $N ≥ 1$.

At the first iteration, $\text{LP}_{FAST}$ is feasible because the column $c_{α,α′}$ only contains ones so it is enough to set $x_{αα} = 1$ to get a feasible solution (given that $N ≥ 1$).

If $\text{LP}_{FAST}$ is feasible for a given $ε$, then $ε$ is an upper bound of the optimal solution value of Problem 9. Indeed, we have $Γ ⊆ |Γ| ≤ N$ (vectors corresponding to $x_α = 1$) such that $∀α ∈ Γ, \exists α′ ∈ Γ, s(α, α′) ≤ ε$. This implies $∀α ∈ Γ, \min_{α′ ∈ Γ} s(α, α′) ≤ ε$ and consequently $\max_{α ∈ Γ} \min_{α′ ∈ Γ} s(α, α′) ≤ ε$. Similarly, if $\text{LP}_{FAST}$ is not feasible, then one cannot find any solution $Γ$ where every $α ∈ Γ$ is controlled by $α′ ∈ Γ$, so $ε$ in this case is a lower bound.

A binary search is then possible on $ε$ to solve Problem 9, by setting the first upper bound $ε^+$ to $ε_{ub}$ and $ε^-$ to 0. This is exactly what Algorithm 1 performs.

Algorithm 1 has time complexity $O(\log(\frac{ε_{ub}}{p}) P(|Γ|, \log(N))^2|Γ|)$, where $P$ is a polynomial, due to the binary search and the Branch&Bound for solving the 0-1 integer linear program $\text{LP}_{FAST}$.

**The $α$-$min$-2-$p$ algorithm**

Solutions provided by $α$-min-2-fast have no performance guarantees and can produce, in theory, very bad solutions, because, for every $Γ$, the real gap $g_r(V, V) = \max_{b ∈ B} [V(b) - V(b)]$ between $V$ and $V$ is approximated by an upper bound only.

In order to get a better approximation of $g_r$, we now consider the vector $V$ to be represented both by $α$-vectors and $β$-vectors in $Δ = \{b, V(b)\}, b ∈ B_{Δ}$, where $B_{Δ}$ a finite subset of $B$, while $V$ remains represented by $α$-vectors only. According to the new representation of $V$, Problem 8 can be approximated by Problem 11, with the new possibility of adding as many $β$-vectors to $Δ$ as necessary to get a better representation of $V$ and a better approximation $g^*(Δ)$ of the optimal gap $g^*$:

$$g^*(Δ) = \min_{Γ ⊆ |Γ| ≤ N} \max_{α ∈ Δ} \min_{α′ ∈ Γ} s(α, α′)$$

where $N ≥ 1$ and $s(α, β(b, V(b))) = α ⊖ b - V(b)$.

Figure 2 gives an example where $V$ is represented with 7 $α$-vectors (in black) and 5 $β$-vectors (in gray).

For any given set of $β$-vectors $Δ$, it is easy to adapt Algorithm 1 to solve Problem 11. One just needs to replace $Γ$ by $Γ × Δ$, pairs $(α, α′) ∈ Γ^2$ by pairs $(α, β) ∈ Γ × Δ$, and $s$ by $s'$. Let us call the corresponding procedure $\text{Fast}_β$. In Algorithm 2, the construction of $Δ$ is the same as in $α$-min; the while loop aims to iteratively adding $β$-vectors corresponding to the current biggest gap $g_r(V, V)$ between $V$ and $V$, until $g_r(V, V)$ and $g^*(Δ)$ are close enough.
The $\alpha$-min-2-solve algorithm

Solving Problem 5 directly without any initial policy at hand is harder to perform. In the finite-horizon case, one has to solve Problem 6 instead of Problem 8, and $BL(V_{t+1})$ is generally very hard to compute. In particular, line 6 of $\alpha$-min-2-p would be very hard to compute if $V'$ were replaced by $BL(V_{t+1})$. However, a simple heuristic based on $\alpha$-min-2-p can be used, where instead of computing the best belief $b^*$ at each iteration (line 6), one generates a random belief. With this change, a new $\tilde{\Gamma}$ is not yet guaranteed to be found at each iteration, so the new stopping criterion is to stop when the number of sampled beliefs has reached a specified limit. This is $\alpha$-min-2-solve. Algorithm $\alpha$-min-2-solve has the advantage that it can be directly compared to the solver $\alpha$-min, while the two previous algorithms cannot because they are not solvers. $\alpha$-min-2-solve has the same time complexity as the previous algorithm since the main complexity term comes from the integer linear program $FAST_\beta$. $\alpha$-min-2-solve should be faster than $\alpha$-min because it does not require to solve Problem 6. Moreover, the performances of $\alpha$-min-2-solve should be competitive because it looks for best combinations of $\alpha$-vectors, while $\alpha$-min greedily adds the “best” $\alpha$-vectors one by one. The infinite-horizon case is harder because one cannot be sure of the convergence of the process. That is why $\alpha$-min-2-solve only deals with the finite-horizon case.

Experiments

We compare $\alpha$-min-2-fast to $\alpha$-min-2-p, and $\alpha$-min-2-solve to $\alpha$-min. $\alpha$-min-2-fast and $\alpha$-min-2-p are not POMDP solvers because they require an initial policy calculated by an external POMDP solver. Therefore, they cannot be directly compared to $\alpha$-min and $\alpha$-min-2-solve.

$\alpha$-min-2-fast and $\alpha$-min-2-p

To compare $\alpha$-min-2-fast and $\alpha$-min-2-p, we conducted experiments on three benchmark problems\footnote{http://www.pomdp.org/examples/} given an initial policy $\Gamma$ provided by SARSOP and the corresponding lower bound $V(b_0)$, where $b_0$ the initial belief (at $t = 0$). We denote by $(\alpha \cdot b - V, V(b_0))$ the characteristics of the problems. We have milos-aaai997 $\equiv (20, 6.8, 184, 574.88)$, hallway2 $\equiv (92.5, 17, 139, 0.25)$ and learning-c4 $\equiv (48, 16.3, 332, 3.3)$.

Figures 3a, 3b and 4a show the encouraging profile of the calculated gaps and bounds as a function of $N$. For $\alpha$-min-2-fast, only an upper bound for $g^*$ is provided, called ‘gap fast’. For $\alpha$-min-2-p the upper bound $g_{ub}$ and the lower bound $g_0$ are provided, respectively called ‘gap precise ub’ and ‘gap precise lb’.

Figures 4b provides the computation times for the $\alpha$-min-2-p approach on each benchmark. The computation times of $\alpha$-min-2-fast are negligible (always less than 1 second) and are not shown.
Figure 3

Figure 5a and Figure 5b show respectively the computation time of $\alpha$-min-2-p as a function of the required precision $p$ ($N$ fixed), and $\hat{V}(b_0)$ as a function of $N$, for milos-aaai97.

$\alpha$-min-2-fast and $\alpha$-min-2-p are able to solve larger problems. Both algorithms were tested on the benchmark TagAvoid $\equiv (870, 5, 30, 615, -6.76)$ (Kurniawati, Hsu, and Lee 2008). For $\alpha$-min-2-fast, the computing time was always less than 3 seconds and good performance was obtained, with $\hat{V}(b_0) \geq 0.98V(b_0)$, but with large gaps (greater than 10 even for $N=50$). $\alpha$-min-2-p provided much smaller gaps, with performance guarantees: for $N=10$, gap=5.98 within precision 0.7. This comes with a high cost in computation time (3389 seconds for $N=10$).

$\alpha$-min-2-solve

In order to compare $\alpha$-min-2-solve to $\alpha$-min, we ran $\alpha$-min-2-solve on several benchmark problems from (Dujardin, Dietterich, and Chadès 2015). The time-horizon chosen for the experiments was $T = 10$. The number of $\beta$-points for $\alpha$-min-2-solve was 100. Table 1 shows that $\alpha$-min-2-solve is always faster while providing a better $V_0(b_0)$ (LB). Computation times (Time) are the average computation times per time step, in seconds. Note that SARSOP is not a N-POMDP solver. Therefore, results are presented for information only.

$\alpha$-min-2-solve has also been tested on the same computational sustainability problem as in (Dujardin, Dietterich, and Chadès 2015): the four populations Sumatran tigers non-stationary problem $^2$: $\alpha$-min-2-solve is clearly faster while providing a better LB.

Table 1: Comparison of $\alpha$-min-2-solve and $\alpha$-min. LB is the provided lower bound of $V_0(b_0)$. SARSOP and $\alpha$-min results are from (Dujardin, Dietterich, and Chadès 2015).

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algo.</th>
<th>$N$</th>
<th>LB</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>abola.10</td>
<td>sarsop</td>
<td>190</td>
<td>64.87</td>
<td>1000</td>
</tr>
<tr>
<td>(30,9,3)</td>
<td>$\alpha$-min</td>
<td>30</td>
<td>62.66</td>
<td>$\leq 1000$</td>
</tr>
<tr>
<td>$\alpha$-min-2</td>
<td>30</td>
<td>$\alpha$-min-2</td>
<td>63.51</td>
<td>$&lt; 1$</td>
</tr>
<tr>
<td>learning.c3</td>
<td>sarsop</td>
<td>11433</td>
<td>1.36</td>
<td>$1000$</td>
</tr>
<tr>
<td>(24,12,3)</td>
<td>$\alpha$-min</td>
<td>24</td>
<td>1.96</td>
<td>$\leq 1000$</td>
</tr>
<tr>
<td>$\alpha$-min-2</td>
<td>24</td>
<td>$\alpha$-min-2</td>
<td>2.09</td>
<td>$&lt; 1$</td>
</tr>
<tr>
<td>cheng.D4-5</td>
<td>sarsop</td>
<td>15</td>
<td>77.29</td>
<td>$1000$</td>
</tr>
<tr>
<td>(4,4,4)</td>
<td>$\alpha$-min</td>
<td>4</td>
<td>77.85</td>
<td>$\leq 1000$</td>
</tr>
<tr>
<td>$\alpha$-min-2</td>
<td>4</td>
<td>$\alpha$-min-2</td>
<td>77.90</td>
<td>$&lt; 1$</td>
</tr>
<tr>
<td>milos-aaai97</td>
<td>sarsop</td>
<td>122</td>
<td>41.48</td>
<td>$1000$</td>
</tr>
<tr>
<td>(20,6,8)</td>
<td>$\alpha$-min</td>
<td>20</td>
<td>50.31</td>
<td>$\leq 1000$</td>
</tr>
<tr>
<td>$\alpha$-min-2</td>
<td>20</td>
<td>$\alpha$-min-2</td>
<td>54.76</td>
<td>$&lt; 1$</td>
</tr>
<tr>
<td>dujardin-ijcai15</td>
<td>$\alpha$-min</td>
<td>7</td>
<td>207.23</td>
<td>$37.8$</td>
</tr>
<tr>
<td>(16,13,16)</td>
<td>$\alpha$-min-2</td>
<td>7</td>
<td>208.45</td>
<td>$&lt; 1$</td>
</tr>
</tbody>
</table>

**Discussion**

The three $\alpha$-min-2 algorithms presented in this paper approximately solve N-POMDPs with different levels of approximation or different assumptions. $\alpha$-min-2-fast minimizes an upper bound of the gap between an initial policy, provided by an external solver, and a policy using only $N$ vectors. $\alpha$-min-2 uses both an upper and a lower bound, which enables it to provide solutions within any chosen precision. However, it is clearly slower than $\alpha$-min-2-fast. Finally, because poor initial POMDP policies can produce poor final solutions, $\alpha$-min-2-solve was written in order to solve N-POMDPs without any initial POMDP policy. However, $\alpha$-min-2-solve can only solve finite-horizon N-POMDPs. For all three algorithms, the computation time is not very sensitive to the parameter $N$, although the worst-case time complexity is exponential in $N$. In practice, the size of the initial set $\Gamma$ of $\alpha$-vectors seems to play a key role for $\alpha$-min-2-p’s computation time. This is because $\alpha$-min-2-p has to solve $|\Gamma|$ linear programs at each iteration to update the upper bound of the gap. The number of states also plays a role for $\alpha$-min-2-p, because the number of $\beta$-points needed to obtain sufficient precision can be very large. Future work will be conducted to reduce the worst-case time complexity.
of the \(\alpha\)-min-2 algorithms.

Interpreting policies for planning problems arising in ecology is very important (Tulloch et al. 2015). While existing POMDP solvers generally provide policies of unlimited size, making interpretation difficult, the \(\alpha\)-min-2 algorithms provide policies of a desired size \(N\) which are as close as possible to optimal policies. In doing so, we hope that \(\alpha\)-min-2 will contribute to bridging the gap between POMDP solutions and their applications.

**Acknowledgements**

We thank Sam Nicol (CSIRO) and Martin Péron (QUT/CSIRO) for providing valuable feedbacks on an earlier version of this manuscript.

**References**


McDonald-Madden, E.; Chadès, I.; McCarthy, M. A.; Linkie, M.; and Possingham, H. P. 2011. Allocating conservation resources between areas where persistence of a species is uncertain. *Ecological Applications* 21(3):844–858.


