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 α-vectors. We would like to find the best possible policy that can be expressed using a fixed number N of α-vectors. We call this the N-POMDP problem. The existing solver α-min approximately solves finite-horizon POMDPs with a controllable number of α-vectors. However α-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 α-min-2. These three algorithms are able to approximately solve N-POMDPs. α-min-2-fast (heuristic) and α-min-2-p (with performance guarantees) are designed to complement an existing POMDP solver, while α-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 α-vectors required. Indeed, current algorithms require many α-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 α-vectors. The recently-published algorithm α-min was a first attempt at approximately solving finite-horizon POMDPs given a limit N on the number of α-vectors (Dujardin, Dietterich, and Chadès 2015). We call this problem N-POMDP. α-min is however a greedy algorithm and does not provide satisfying performance guarantees. Here, we propose three new algorithms, based on a common approach, α-min-2, that directly searches for subsets of α-vectors of size N. α-min-2-fast (heuristic) and α-min-2-p (with performance guarantees) are designed to work in complement of an existing POMDP solver, and α-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 α-min.

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

POMDPs, N-POMDPs and the α-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 $\{S, A, O, \tau_a, \Omega_a, R, H\}$, where

- $H = \{0, ..., T-1\}$ are called "time steps", $T \in \mathbb{N}$. H 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.

- τ_a is the transition matrix for action a. Its elements are $\tau_a\left(s_t, s_{t+1}\right)$.
- Ω_a is the observation matrix for action a. Its elements are $\Omega_a(s_{t+1}, z_{t+1})$.
- R is the reward matrix. Its elements are $R\left(a_t, s_t\right)$.

For sake of clarity, we define the following notation:

- For action $a \in A$ and for observation $z \in O$, let $M_{a,z}$ be the matrix of dimension $S \times S$ such that $M_{a,z}(s_{t+1}, s_t) = \Omega_a(z, s_{t+1})\tau_a(s_{t+1}, s_t)$.
- For every $a \in A$, the vector $\boldsymbol{r_a} = \boldsymbol{R(a, \cdot)}$ 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 (Åström 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. Π_0 maximizes the expected sum of rewards $E[\sum_{t \in H} \boldsymbol{r_{a_t}} \cdot \boldsymbol{b_t}]$ over the time horizon H ($\cdot$ denotes the scalar product). For each time step t, for a given belief state $\boldsymbol{b_t}$ and a given policy $\Pi_t = \cup_{t' \in \{t,...,T-1\}} \pi_{t'}$, the expected sum $E[\sum_{t' \in \{t,...,T-1\}} \boldsymbol{r_{a_t}} \cdot \boldsymbol{b_t}]$ is also referred to as the value function $V_{t,\Pi_t}(\boldsymbol{b_t})$. A value function allows us to rank policies by assigning a real value to each belief $\boldsymbol{b_t}$. An optimal policy Π_t^* is a policy such that, $\forall\ \boldsymbol{b_t} \in B$, $\forall\ \Pi_t$, $V_{t,\Pi_t^*}(\boldsymbol{b_t}) \geq V_{t,\Pi_t}(\boldsymbol{b_t})$. 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): $\forall \boldsymbol{b_t} \in B$,

$$V_t(\boldsymbol{b_t}) = \max_{a_t \in A} \{\boldsymbol{r_{a_t}} \cdot \boldsymbol{b_t} + \sum_{z_t \in O} p(z_{t+1}|a_t, \boldsymbol{b_t}) V_{t+1}(\boldsymbol{b_{t+1}})\} \tag{1}$$

where each component $b_{t+1}(s_{t+1})$ of the new belief can be computed as follows:

$$b_{t+1}(s_{t+1}) = \frac{\Omega_a(s_{t+1}, z_{t+1}) \sum_{s_t \in S} \tau_a(s_t, s_{t+1}) b_t(s_t)}{\sum_{s_t, s'_{t+1} \in S} \Omega_a(s'_{t+1}, z_{t+1}) \tau_a(s_t, s'_{t+1}) b_t(s_t)} \tag{2}$$

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 α-vectors (Smallwood and Sondik 1973). For every $t \in H$, there exists a finite set Γ_t of vectors of dimension $|S|$ (the so-called α-vectors) that define entirely V_t such as $|\Gamma_t|$ is minimal and

$$\forall t \in H, \forall \boldsymbol{b_t} \in B, V_t\left(\boldsymbol{b_t}\right) = \max_{\boldsymbol{\alpha_t} \in \Gamma_t} \boldsymbol{\alpha_t} \cdot \boldsymbol{b_t}. \tag{3}$$

Given that $\Gamma_{T-1} = \{\boldsymbol{r_a} |\ a \in A,\ \boldsymbol{r_a}$ is not dominated$\}$, one can in theory build a set of α-vectors defining the value function V_t from Γ_{t+1} at any time step $t \in H' = \{0, ..., T-2\}$, because every $\boldsymbol{\alpha_t} \in \Gamma_t$ can be written:

$$\boldsymbol{\alpha_t} = [\boldsymbol{r_{a_t}} + \sum_{z_{t+1} \in O} (\boldsymbol{\alpha_{t+1}^{a_t, z_{t+1}}})^T M_{a_t, z_{t+1}}]^T \tag{4}$$

where $a_t \in A$ and $\boldsymbol{\alpha_{t+1}^{a_t, z_{t+1}}}, z_{t+1} \in O$ are elements of Γ_{t+1}.

The set of α-vectors G_t given by Equation 4 is such that $\Gamma_t \subseteq G_t$. Note that some α-vectors of G_t could be dominated by others and therefore do not belong to Γ_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 Γ_t from a given Γ_{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 Γ_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 α-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).

$$\max_{\Gamma_0, ..., \Gamma_{T-1} \in \Theta,\ s.t. \forall t \in H, |\Gamma_t| \leq N} V_{0,\Gamma}(\boldsymbol{b_0}) \tag{5}$$

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

(Littman, Cassandra, and Kaelbling 1995), it is stated that, unless $P = NP$, no representation of the optimal value function V_t given an optimal value function V_{t+1} can be found in polynomial time in the instance and the number of α-vectors needed to describe V_t (i.e. in N in our case). If an N-POMDP could be solved in polynomial time in the instance of the POMDP and N, one could then set N to the actual number of α-vectors needed to describe V_t and find an optimal policy of the corresponding POMDP in polynomial time. □

The α-min principle

Dujardin, Dietterich, and Chadès (2015) show that an approximate solution for N-POMDPs can be found by minimizing the Bellman error iteratively at every time step (Problem 6).

$$\min_{\tilde{\Gamma}_t \subseteq BL(\tilde{\Gamma}_{t+1}), |\tilde{\Gamma}_t| \leq N} \max_{\boldsymbol{b} \in B} [BL(\tilde{V}_{t+1})(\boldsymbol{b}) - \tilde{V}_t(\boldsymbol{b})] \quad (6)$$

where BL is the Bellman operator, $\tilde{\Gamma}_t$ is the set of approximate α-vectors for $BL(\tilde{\Gamma}_{t+1})$ and $\tilde{V}_t$ is the value function corresponding to $\tilde{\Gamma}_t$, $t \in H$.

To approximately solve Problem 6, α-min solves Problem 7 N times at each time step using a mixed integer linear program.

$$\max_{\boldsymbol{b} \in B} [BL(\tilde{V}_{t+1})(\boldsymbol{b}) - \tilde{V}_t(\boldsymbol{b})] \quad (7)$$

In each iteration, the α-vector that maximizes $\alpha \cdot b^*$ is added to the current $\tilde{\Gamma}_t$ (Figure 1).

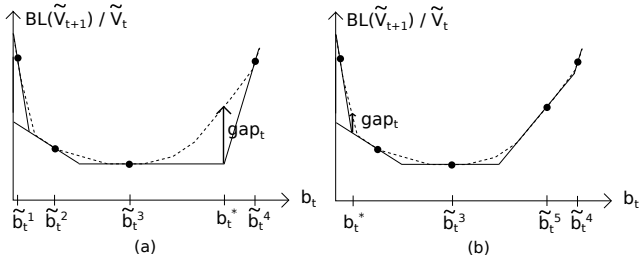


Figure 1: Two successive iterations (a) and (b) of α-min for a given time step t. In both figures, solid lines represent $\tilde{V}_t$ while dashed lines represent $BL(\tilde{V}_{t+1})$, and belief $\boldsymbol{b_t^*}$ corresponds to the current solution of Problem 7. (Reproduced from (Dujardin, Dietterich, and Chadès 2015))

α-min shares some similarities with the linear support algorithm (Cheng 1988) because it iteratively constructs new α-vectors, but it is much quicker than Cheng's algorithm and can solve bigger problems. Neither α-min nor linear support are designed to solve N-POMDPs exactly. The main reason is that new α-vectors are added greedily at each iteration, and therefore the optimal combination of α-vectors is unlikely to be found.

One can easily construct a 2-state example ($S = \{0, 1\}$) where the greedy strategy can be arbitrary bad. We consider $\Gamma = \{\alpha_1, \alpha_2, \alpha_3\}$ with $\alpha_1 = (x, -x), \alpha_2 = (0, 0), \alpha_3 = (-x, x), x \in \mathbb{R}_+$. α_1 is a dominating vector for $b \in \{b \in B \mid b(0) \leq b(1)\}$, α_2 is dominating for $b \in \{b \in B \mid b(0) = b(1)\}$ and α_3 is dominating for $b \in \{b \in B \mid b(0) \geq b(1)\}$. A greedy approach starting with α_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 $\tilde{\Gamma}^* = \{\alpha_1, \alpha_3\}$.

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

The α-min-2 algorithms

Given that good POMDPs 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 Γ without any size limitation and then to select the best combination of α-vectors of Γ minimizing the gap between the initial policy and the new policy (Problem 8).

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

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

The α-min-2-fast algorithm

Suppose we have at hand a policy Γ, and we want to solve Problem 8.

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

We have

$$\begin{aligned} \max_{\boldsymbol{b} \in B} [V(\boldsymbol{b}) - \tilde{V}(\boldsymbol{b})] &\leq \max_{b \in B}(\max_{\alpha \in \Gamma} \alpha \cdot b - \max_{\tilde{\alpha} \in \tilde{\Gamma}} \tilde{\alpha} \cdot b) \\ &\leq \max_{\alpha \in \Gamma} \min_{\tilde{\alpha} \in \tilde{\Gamma}} \max_{b \in B(\alpha)} (\alpha \cdot b - \tilde{\alpha} \cdot b) \\ &\leq \max_{\alpha \in \Gamma} \min_{\tilde{\alpha} \in \tilde{\Gamma}} s(\tilde{\alpha}, \alpha) \end{aligned}$$

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

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

for $N \geq 1$.

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.

$$
\begin{array}{lll}
\min & g & \\
s.t. & g & \geq s(\alpha,\alpha')y_{\alpha,\alpha'},\ \alpha,\alpha' \in \Gamma \\
& \sum_{\alpha\in\Gamma} y_{\alpha,\alpha'} & \geq 1,\ \alpha' \in \Gamma \\
& x_\alpha & \geq y_{\alpha,\alpha'},\ \alpha,\alpha' \in \Gamma \\
& \sum_{\alpha\in\Gamma} x_\alpha & \leq N \\
& y_{\alpha,\alpha'} & \in \{0,1\} \\
& x_\alpha & \in \{0,1\}
\end{array}
\tag{10}
$$

where every 0-1 variable x_α is such that $x_\alpha = 1$ iff α is in $\tilde{\Gamma}$ and $y_{\alpha,\alpha'}$ are 0-1 variables such that $y_{\alpha,\alpha'} = 1$ iff $\alpha \in \tilde{\Gamma}$ and $\alpha' \in \Gamma$ minimizes $s(\alpha,\alpha')$.

Linear Program 10 is a mixed integer linear programs with many 0-1 variables ($|\Gamma|^2$) and is consequently slow to solve. We propose a faster way to solve Problem 9, using pure integer linear programs and with only $|\Gamma|$ 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** FAST($\Gamma, p, N \geq 1$)
2: $\epsilon^+ = \epsilon_{ub}, \epsilon^- = 0, \delta = \epsilon^+ - \epsilon^-$
3: **while** $\delta > p$ **do**
4: $\epsilon = \frac{\epsilon^+ + \epsilon^-}{2}$
5: **for** $\alpha, \alpha' \in \Gamma$ **do**
6: **if** $s_{\alpha,\alpha'} \leq \epsilon$ **then** $c_{\alpha,\alpha'} = 1$
7: **else** $c_{\alpha,\alpha'} = 0$
8: $C = \{c_{\alpha,\alpha'} \mid \alpha, \alpha' \in \Gamma\}$
9: $\tilde{\Gamma} \leftarrow LP_{FAST}(C, N)$
10: **if** $LP_{FAST}(C, N)$ has no solution **then** $\epsilon^+ = \epsilon$
11: **else** $\epsilon^- = \epsilon$
12: $\epsilon^+ = \epsilon_{ub}, \epsilon^- = 0, \delta = \epsilon^+ - \epsilon^-$
return $\tilde{\Gamma}, \epsilon$

Here, ϵ_{ub} is an upper bound of the solution value ϵ of Problem 9 and $\tilde{\Gamma}$ (line 9) is the solution returned by LP_{FAST}, given that $\alpha \in \tilde{\Gamma} \iff x^*_\alpha = 1$.

$$
\begin{array}{lll}
\min & \sum_{\alpha\in\Gamma} x_\alpha & \\
s.t. & \sum_{\alpha\in\Gamma} c_{\alpha,\alpha'} x_\alpha & \geq 1,\ \alpha' \in \Gamma \\
& \sum_{\alpha\in\Gamma} x_\alpha & \leq N \\
& x_\alpha & \in \{0,1\}
\end{array}
\tag{LP_{FAST}}
$$

Remark. *Linear Program 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 $\alpha_0 \in \Gamma$ and set $\epsilon_{ub} = \max_{\alpha\in\Gamma} s(\alpha_0, \alpha)$ (can be solved with linear programming). We have then $\epsilon_{ub} \geq \max_{\alpha\in\Gamma} \min_{\tilde{\alpha}\in\tilde{\Gamma}} s(\tilde{\alpha}, \alpha), \tilde{\Gamma} \subseteq \Gamma, |\tilde{\Gamma}| \geq 1$, so $\epsilon_{ub} \geq \min_{\tilde{\Gamma}\subseteq\Gamma, |\tilde{\Gamma}| \leq N} \max_{\alpha\in\Gamma} \min_{\tilde{\alpha}\in\tilde{\Gamma}} s(\tilde{\alpha}, \alpha)$ for $N \geq 1$.

At the first iteration, LP_{FAST} is feasible because the column $c_{\alpha_0,\alpha'}$ only contains ones so it is enough to set $x_{\alpha_0} = 1$ to get a feasible solution (given that $N \geq 1$). If 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 $\tilde{\Gamma} \subseteq \Gamma, |\tilde{\Gamma}| \leq N$ (vectors corresponding to $x^*_\alpha = 1$) such that $\forall \alpha \in \Gamma, \exists \tilde{\alpha} \in \tilde{\Gamma}, s(\tilde{\alpha}, \alpha) \leq \epsilon$. This implies $\forall \alpha \in \Gamma,\ \min_{\tilde{\alpha}\in\tilde{\Gamma}} s(\tilde{\alpha}, \alpha) \leq \epsilon$ and consequently $\max_{\alpha\in\Gamma} \min_{\tilde{\alpha}\in\tilde{\Gamma}} s(\tilde{\alpha}, \alpha) \leq \epsilon$. Similarly, if LP_{FAST} is not feasible, then one cannot find any solution $\tilde{\Gamma}$ where every $\alpha \in \Gamma$ is controlled by $\tilde{\alpha} \in \tilde{\Gamma}$, 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 $O(\log(\frac{\epsilon_{ub}}{p}) P(|\Gamma|, log(N)) 2^{|\Gamma|})$, where P is a polynomial, due to the binary search and the Branch&Bound for solving the 0-1 integer linear program LP_{FAST}.

□

The α-min-2-p algorithm

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

In order to get a better approximation of $g_r(V, \tilde{V})$, we now consider V to be represented both by α-vectors and β-points in $\Delta = \{(b, V(b)), b \in B_\Delta\}$, where B_Δ a finite subset of B, while $\tilde{V}$ 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-points to Δ as necessary to get a better representation of V and a better approximation $g^*(\Delta)$ of the optimal gap g^*:

$$
g^*(\Delta) = \min_{\tilde{\Gamma}\subseteq\Gamma, |\tilde{\Gamma}|\leq N} \max_{\beta\in\Delta} \min_{\tilde{\alpha}\in\tilde{\Gamma}} s'(\tilde{\alpha}, \beta) \tag{11}
$$

where $N \geq 1$ and $s'(\alpha, \beta(b, V(b))) = \alpha \cdot b - V(b)$.

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

For any given set of β-points Δ, it is easy to adapt Algorithm 1 to solve Problem 11. One just needs to replace Γ by $\Gamma \times \Delta$, pairs $(\alpha, \alpha') \in \Gamma^2$ by pairs $(\alpha, \beta) \in \Gamma \times \Delta$, and s by s'. Let us call the corresponding procedure $FAST_\beta$. In Algorithm 2, the construction of Δ is the same as in α-min: the while loop aims to iteratively adding β-points corresponding to the current biggest gap $g_r(V, \tilde{V})$ between V and $\tilde{V}$, until $g_r(V, \tilde{V})$ and $g^*(\Delta)$ are close enough.

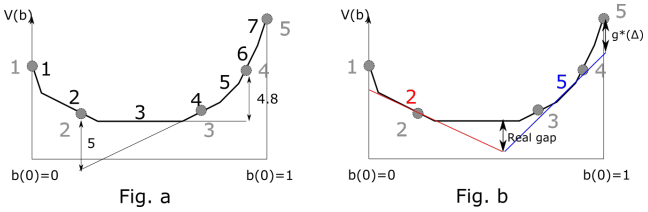


Fig. a Fig. b

Figure 2: Fig. a and Fig. b show an example of a mixed representation of V, using α-vectors (Γ) and β-points (Δ). α-vectors are in dark and β-points are in grey. Fig. a shows $s'(\alpha_4, \beta_2) = 5$ and $s'(\alpha_3, \beta_4) = 4.8$. On Fig. b one can see the 'real gap' $g_r(V, \tilde{V})$ between Γ and $\tilde{\Gamma} = \{\alpha_2, \alpha_5\}$, and the gap $g^*(\Delta)$ between Δ and $\tilde{\Gamma}$.

Algorithm 2 α-min-2-p

1: **procedure** PRECISE(Γ, p, N)
2: Let Δ be the set of the β-points corresponding to extreme beliefs of B: $(1, 0, \cdots, 0), (0, 1, \cdots, 0), \cdots, (0, \cdots, 0, 1)$
3: $\delta \leftarrow \inf$, $g_{ub} \leftarrow \inf$
4: **while** $\delta > \frac{p}{2}$ **do**
5: $(\tilde{\Gamma}, g_\Delta) \leftarrow FAST_\beta(\Gamma \times \Delta, \frac{p}{2}, N)$
6: $b^* \leftarrow \arg\max_{b \in B}(V(b) - \tilde{V}(b))$
7: $g_{ub} \leftarrow \min(g_{ub}, V(b^*) - \tilde{V}(b^*))$
8: $\delta \leftarrow g_{ub} - g_\Delta$
9: $\Delta \leftarrow \Delta \cup \{b^*\}$
return $\tilde{\Gamma}, g_{ub}$

Proposition 3. *Algorithm 2 solves Problem 8 within precision p in a finite number of iterations.*

Proof. Given that $FAST_\beta(\Gamma \times \Delta, \frac{p}{2}, N)$ provides an optimal solution to Problem 11 within $\frac{p}{2}$, upon termination we have $\tilde{\Gamma}$, Δ and g_Δ such as $g_\Delta \leq g^*(\Delta) + \frac{p}{2}$. Additionally we have $g^*(\Delta) \leq g^*$ because $\Delta \subseteq \{(b, \tilde{V}(b)), b \in B\}$. This leads to $g_\Delta \leq g^* + \frac{p}{2}$. g_{ub} is always set to a real gap $g_r(V, \tilde{V}) = V(b^*) - \tilde{V}^*(b)$ or its value does not change (line 7). Therefore, we always have $g^* \leq g_{ub}$ (because $g^* \leq g_r(V, \tilde{V}), \forall \tilde{V}$). Finally, at the end of the algorithm we have $g_{ub} - g_\Delta \leq \frac{p}{2}$, so $g_{ub} \leq g_\Delta + \frac{p}{2} \leq (g^* + \frac{p}{2}) + \frac{p}{2} \leq g^* + p$. Overall we have $g^* \leq g_{ub} \leq g^* + p$.

Algorithm 2 terminates after a finite number of iterations. At each iteration, if the $\tilde{\Gamma}$ provided by $FAST_\beta$ (line 5) is the same as a previously-generated $\tilde{\Gamma}$, we obtain $\delta = 0$ by construction. So, in the worst case, $\tilde{\Gamma}$ will be successively equal to every possible combination of N α-vectors before exiting the while loop. The number of iterations is therefore bounded by $|\Gamma|^N$. $FAST_\beta$ has time complexity $O(\log(\frac{\epsilon_{ub}}{p})P(|\Gamma|, log(N))2^{|\Gamma|})$, where ϵ_{ub} is defined in the proof of Proposition 2 and P is a polynomial. Finally, the computation in line 6 can be performed in polynomial time ($|\Gamma|$ is considered as part of the instance): $\max_{b \in B}(V(b) - \tilde{V}(b)) = \max_{\alpha \in \Gamma} \max_{b \in B}(\alpha \cdot b - \tilde{V} \mid \tilde{V} \geq \tilde{\alpha} \cdot b, \tilde{\alpha} \in \tilde{\Gamma})$ can be solved using $|\Gamma|$ linear programs.

Overall, the complexity of Algorithm 2 is $O(|\Gamma|^N \log(\frac{\epsilon_{ub}}{p})P(|\Gamma|, log(N))2^{|\Gamma|})$ where P is a polynomial. □

The α-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 α-min-2-p would be very hard to compute if V were replaced by $BL(V_{t+1})$. However, a simple heuristic based on α-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 α-min-2-solve. Algorithm α-min-2-solve has the advantage that it can be directly compared to the solver α-min, while the two previous algorithms cannot because they are not solvers. α-min-2-solve has the same time complexity as the previous algorithms since the main complexity term comes from the integer linear program $FAST_\beta$. α-min-2-solve should be faster than α-min because it does not require to solve Problem 6. Moreover, the performances of α-min-2-solve should be competitive because it looks for best *combinations* of α-vectors, while α-min greedily adds the "best" α-vectors *one by one*. The infinite-horizon case is harder because one cannot be sure of the convergence of the process. That is why α-min-2-solve only deals with the finite-horizon case.

Experiments

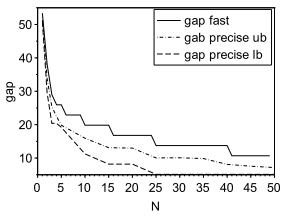
We compare α-min-2-fast to α-min-2-p, and α-min-2-solve to α-min. α-min-2-fast and α-min-2-p are not POMDP solvers because they require an initial policy calculated by an external POMDP solver. Therefore, they cannot be directly compared to α-min and α-min-2-solve.

α-min-2-fast and α-min-2-p

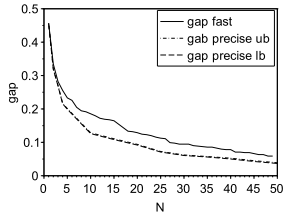
To compare α-min-2-fast and α-min-2-p, we conducted experiments on three benchmark problems[1] given an initial policy Γ provided by SARSOP and the corresponding lower bound $V(b_0)$, where b_0 the initial belief (at $t = 0$). We denote by $(|S|, |A|, |O|, |\Gamma|, V(b_0)))$ the characteristics of the problems. We have milos-aaai997 $\equiv$ (20,6,8,184,574.8), 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 α-min-2-fast, only an upper bound for g^* is provided, called 'gap fast'. For α-min-2-p the upper bound g_{ub} and the lower bound $g(\Delta) - \frac{p}{2}$ of g^* are provided, respectively called 'gap precise ub' and 'gap precise lb'.

[1] http://www.pomdp.org/examples/



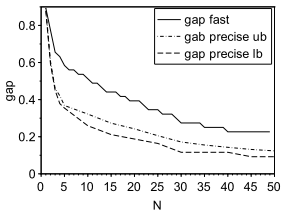
(a) Gaps milos-aaai97



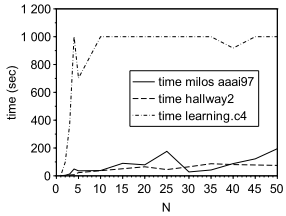
(b) Gaps hallway2 (gap_{ub} and gap_{lb} overlap almost perfectly)

Figure 3

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



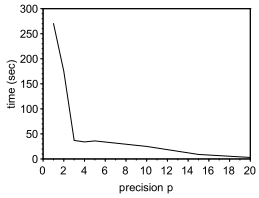
(a) Gaps learning.c4



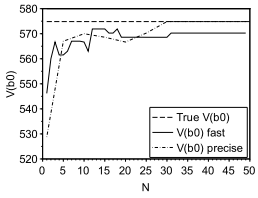
(b) Computation times (α-min-2-p)

Figure 4

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



(a) Time function of precision - milos-aaai97, $N = 5$.



(b) $\tilde{V}(b_0)$ function of N - milos-aaai97

Figure 5

α-min-2-fast and α-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 α-min-2-fast, the computing time was always less than 3 seconds and good performance was obtained, with $\tilde{V}(b_0) \geq 0.98V(b_0)$, but with large gaps (greater than 10 even for N=50). α-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).

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

Problem ($\|S\|, \|A\|, \|O\|$)	**Algo.**	N	**LB**	**Time(s)**
aloha.10	sarsop	190	64.87	1000
(30,9,3)	α-min	30	62.66	$\leq$1000
	α-min-2	30	**63.51**	$\mathbf{< 1}$
learning.c3	sarsop	11433	1.36	1000
(24,12,3)	α-min	24	1.96	$\leq$1000
	α-min-2	24	**2.09**	$\mathbf{< 1}$
cheng.D4-5	sarsop	15	77.29	1000
(4,4,4)	α-min	4	77.85	$\leq$1000
	α-min-2	4	**77.90**	$\mathbf{< 1}$
milos-aaai97	sarsop	122	41.48	1000
(20,6,8)	α-min	20	50.31	$\leq$1000
	α-min-2	20	**54.76**	$\mathbf{< 1}$
dujardin-ijcai15	α-min	7	207.23	37.8
(16,13,16)	α-min-2	7	**208.45**	$\mathbf{< 1}$

α-min-2-solve

In order to compare α-min-2-solve to α-min, we ran α-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 β-points for α-min-2-solve was 100. Table 1 shows that α-min-2-solve is always faster while providing a better $\tilde{V}_0(b_0)$ (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.

α-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]: α-min-2-solve is clearly faster while providing a better LB.

Discussion

The three α-min-2 algorithms presented in this paper approximately solve N-POMDPs with different levels of approximation or different assumptions. α-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 α-vectors. α-min-2-p uses both an upper and a lower bound, which enables it to provide solutions within any chosen precision. However, it is clearly slower than α-min-2-fast. Finally, because poor initial POMDP policies can produce poor final solutions, α-min-2-solve was written in order to solve N-POMDPs without any initial POMDP policy. However, α-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 Γ of α-vectors seems to play a key role

[2] available at https://sites.google.com/site/ijcaialphamin/home

for α-min-2-p's computation time. This is because α-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 α-min-2-p, because the number of β-points needed to obtain sufficient precision can be very large. Future work will be conducted to reduce the worst-case time complexity of the α-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 α-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 α-min-2 will contribute to bridging the gap between POMDP solutions and their applications.

Acknowledgements

We thank Sam Nicol (CSIRO) for providing valuable feedbacks on an earlier version of this manuscript.

References

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.

Chadès, I.; McDonald-Madden, E.; McCarthy, M. A.; Wintle, B.; Linkie, M.; and Possingham, H. P. 2008. When to stop managing or surveying cryptic threatened species. *Proceedings of the National Academy of Sciences* 105(37):13936–13940.

Cheng, H.-T. 1988. *Algorithms for Partially Observable Markov Decision Process*. Ph.D. Dissertation, University of British Columbia.

Dujardin, Y.; Dietterich, T.; and Chadès, I. 2015. α-min: A compact approximate solver for finite-horizon POMDPs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 15)*, 2582–2588.

Fackler, P., and Pacifici, K. 2014. Addressing structural and observational uncertainty in resource management. *Journal of Environmental Management* 133:27 – 36.

Gonzalez, T. F. 1985. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science* 38:293–306.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101(1):99–134.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. Sarsop: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, volume 2008. Zurich, Switzerland.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1995. Efficient dynamic-programming updates in partially observable Markov decision processes. Technical report.

Madani, O.; Hanks, S.; and Condon, A. 2003. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence* 147(1):5–34.

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.

Nicol, S., and Chadès, I. 2012. Which states matter? An application of an intelligent discretization method to solve a continuous POMDP in conservation biology. *PLoS ONE* 7(2):e28993.

Nicol, S.; Buffet, O.; Iwamura, T.; and Chades, I. 2013. Adaptive management of migratory birds under sea level rise. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2955–2957.

Nicol, S.; Fuller, R. A.; Iwamura, T.; and Chadès, I. 2015. Adapting environmental management to uncertain but inevitable change. *Proceedings of the Royal Society of London B: Biological Sciences* 282(1808):20142984.

Papadimitriou, C. H., and Tsitsiklis, J. N. 1987. The complexity of Markov decision processes. *Mathematics of operations research* 12(3):441–450.

Pineau, J.; Gordon, G.; and Thrun, S. 2006. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research* 335–380.

Poupart, P.; Kim, K.-E.; and Kim, D. 2011. Closing the gap: Improved bounds on optimal POMDP solutions. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*, 194,201.

Åström, K. J. 1965. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications* 10(1):174.

Regan, T. J.; Chadès, I.; and Possingham, H. P. 2011. Optimally managing under imperfect detection: a method for plant invasions. *Journal of Applied Ecology* 48(1):76–85.

Shani, G.; Pineau, J.; and Kaplow, R. 2012. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems* 27(1):1–51.

Sigaud, O., and Buffet, O. 2013. *Markov decision processes in artificial intelligence*. John Wiley & Sons.

Smallwood, R. D., and Sondik, E. J. 1973. The optimal control of partially observable Markov processes over a finite horizon . *Operations Research* 21(5):1071–1088.

Spaan, M. T., and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for pomdps. *Journal of artificial intelligence research* 24:195–220.

Springborn, M., and Sanchirico, J. N. 2013. A density projection approach for non-trivial information dynamics: Adaptive management of stochastic natural resources. *Journal of Environmental Economics and Management* 66(3):609–624.

Tulloch, V. J.; Tulloch, A. I.; Visconti, P.; Halpern, B. S.; Watson, J. E.; Evans, M. C.; Auerbach, N. A.; Barnes, M.; Beger, M.; Chadès, I.; et al. 2015. Why do we map threats? Linking threat mapping with actions to make better conservation decisions. *Frontiers in Ecology and the Environment* 13:91–99.