

Robust probabilistic planning with ilao

Daniel A. M. Moreira¹ · Karina Valdivia Delgado²  · Leliane Nunes de Barros¹

Published online: 15 April 2016
© Springer Science+Business Media New York 2016

Abstract In probabilistic planning problems which are usually modeled as Markov Decision Processes (MDPs), it is often difficult, or impossible, to obtain an accurate estimate of the state transition probabilities. This limitation can be overcome by modeling these problems as Markov Decision Processes with imprecise probabilities (MDP-IPs). Robust LAO* and Robust LRTDP are efficient algorithms for solving a special class of MDP-IPs where the probabilities lie in a given interval, known as Bounded-Parameter Stochastic-Shortest Path MDP (BSSP-MDP). However, they do not make clear what assumptions must be made to find a robust solution (the best policy under the worst model). In this paper, we propose a new efficient algorithm for BSSP-MDPs, called Robust ILAO* which has a better performance than Robust LAO* and Robust LRTDP, considered the-state-of-the-art of robust probabilistic planning. We also define the assumptions required to ensure a robust solution and prove that Robust ILAO* algorithm converges to optimal values if the initial value of all states is admissible.

Keywords Bounded-parameter Markov decision process · Probabilistic planning · Heuristic search

1 Introduction

Markov Decision Process (MDP) is a mathematical model for sequential decision-making. An MDP models the interaction between an agent and its environment (fully observable): the agent chooses an action that takes him to a next state according with a probability distribution [19]. This can be repeated for a horizon of n steps.

MDPs model a wide range of problems such as: (i) medical diagnosis and treatment of diseases [18]; (ii) robot navigation with probabilistic sensor models; and (iii) supply chain management with uncertainty demand [10].

MDPs problems can have a finite, infinite or indefinite horizon. In the particular case of problems with an indefinite horizon, called Stochastic Shortest-Path Markov Decision Processes (SSP-MDPs), the objective is to reach a goal state with the minimum expected accumulated action cost. In general, probabilistic planning problems are modeled as SSP-MDPs, which are MDPs under special assumptions.

Classical dynamic programming algorithms such as *value iteration* and *policy iteration* [19], can be used to solve SSP-MDPs; however, since these algorithms evaluate the whole set of states, they are inefficient [11]. One way to overcome this kind of inefficiency is to evaluate only a subset of states that are relevant to the planning problem. Real-Time Dynamic Programming (RTDP) [1] and LAO*[10] were proposed to solve SSP-MDPs when the initial state s_0 is known, so they focus only on states that are reachable from s_0 , which means that only a subset of all states is evaluated.

✉ Karina Valdivia Delgado
kvd@usp.br

Daniel A. M. Moreira
daniel.augusto.moreira@usp.br

Leliane Nunes de Barros
leliane@ime.usp.br

¹ Institute of Mathematics and Statistics, University of São Paulo, R. do Matão 1010, Butantã, São Paulo, Brazil

² School of Arts, Sciences and Humanities, University of São Paulo, Av. Arlindo Béttio 1000, Ermelino Matarazzo, São Paulo, Brazil

Another difficulty encountered when using SSP-MDPs for probabilistic planning is that in real world applications it is difficult, or impossible, to obtain exact probabilities. In this case, it is necessary to deal with imprecise probabilities and coming up with a robust solution, i.e., the best decision for the worst-case scenario.

Example 1 [15] In a traffic-control problem at an intersection, it is difficult to estimate the "turn probabilities" for each traffic lane when drivers have the choice of going straight on or making a turn. These lane-turning probabilities may change during the day or throughout the year, depending on the volume of traffic at other intersections, and during holidays and special events. It is also difficult to obtain an accurate estimate of "arrival probabilities" at all hours of the day. In general, it is impossible to accurately model all of these complex dependencies. In this case, the ideal situation would be to have a traffic control policy optimized over a range of probabilities so that it can be robust to inherent non-stationarity turn and arrival probabilities.

Example 2 [17] In an automated navigation system, the probabilities of reaching different locations after a movement may change depending on environmental factors (such as weather and road conditions), and this can make the navigation more difficult and subject to failure. It is hard to model all these changes accurately since they can include many external dependencies. In view of this, it would be better to have a robust policy that is optimized for a range of feasible probabilities.

Example 3 [17] In the field of genetics, a genetic regulatory network and a range of activities (therapeutic interventions) can be modelled as an MDP [2, 8, 22]. This procedure should prevent the network from moving into undesirable states associated with diseases. However, modelling the exact MDP transition probabilities may not be possible when there are few samples or when exogenous events occur. In this case, it would be better to implement a robust policy for therapeutic interventions.

These three examples can be modeled as Markov Decision Processes with imprecise probabilities (MDP-IPs) [5, 14–16], a general class of MDPs, where the transition function is parameterized and subject to a set of constraints. When these constraints take the form of intervals, there is a special class of MDP-IP, called *Bounded-Parameter MDP* (BMDP) [21]. A BMDP with an indefinite horizon is a Bounded-Parameter Stochastic-Shortest Path MDP (BSSP-MDP). Two efficient algorithms for solving BSSP-MDP are Robust LAO* and Robust LRTDP. However, they do not make clear what assumptions must be made in order to find

a robust solution, i.e., searching for the best policy under the worst model.

In this paper, we propose a new efficient algorithm for BSSP-MDPs, called Robust ILAO* that has a better performance when compared to Robust LAO* and Robust LRTDP, considered the state-of-the-art of robust probabilistic planning. We also define the assumptions that must be made to guarantee a robust solution and prove that the Robust ILAO* algorithm converges to optimal values if the initial value of all the states is admissible.

2 Background

In this section, we review the basic concepts of SSP-MDPs and BSSP-MDPs including the assumptions that these processes must satisfy to ensure that an optimal policy can exist. We also describe previously proposed algorithms to solve BSSP-MDPs which are as follows: Robust Value Iteration [25], Robust LRTDP [20] and Robust LAO* [24]. In addition, we demonstrate the Robust (I)LAO* convergence under the BSSP-MDP assumptions.

2.1 Stochastic shortest-path problem

A Stochastic Shortest-Path Markov Decision Problem (SSP-MDP) [7] can be defined by the tuple $\langle S, s_0, S_g, A, p, c \rangle$, where:

- S is a finite set of states;
- $s_0 \in S$ is the initial state;
- $S_g \subseteq S$ is a set of goal states;
- A is a finite set of actions and $A(s) \subseteq A$ denotes the set of actions applicable in state s ;
- $p(s'|s, a)$ is the transition function that represents the probability of reaching the state s' when action a is applied in state s . Furthermore, any action applicable in a goal state must cause a transition to itself, i.e., $p(s|s, a) = 1, \forall s \in S_g, \forall a \in A(s)$; and
- $c(s, a)$ is a positive cost function that maps each state s and action $a \in A(s)$ to a real number.

An SSP-MDP has an indefinite horizon, i.e., the number of actions carried out by the agent to reach a goal state is finite and unknown. The objective of an SSP-MDP problem is to reach a goal state with the *minimum expected accumulated costs*. The solution for SSP-MDP is represented by a mapping from states to actions, $\pi : S \rightarrow A$, called a policy. A stationary policy π is called proper if for every state, π leads to a goal state with probability 1 (a policy that is not proper is improper).

The following assumptions are made for SSP-MDP:

Assumption 1 *There exists at least one proper policy.*

Assumption 2 Every improper policy has an infinite expected cost in all the states that cannot reach the goal state with probability 1.

The value of a policy π , known as evaluation function f^π , is defined as:

$$f^\pi(s) = \begin{cases} 0 & \text{if } s \text{ is a goal state,} \\ c(s, \pi(s)) + \sum_{s' \in S} p(s'|s, \pi(s)) f^\pi(s') & \text{otherwise.} \end{cases} \quad (1)$$

Finding an optimal solution means finding an optimal policy π^* such that $f^* = \min_\pi f^\pi$, which satisfies the Bellman optimality equation [7]:

$$f^*(s) = \begin{cases} 0 & \text{if } s \text{ is a goal state,} \\ \min_{a \in A(s)} [c(s, a) + \sum_{s' \in S} p(s'|s, a) f^*(s')] & \text{otherwise.} \end{cases} \quad (2)$$

The classical synchronous dynamic algorithm for solving SSP-MDPs, named Value Iteration, finds the optimal or ϵ -optimal evaluation function by iteratively performing backups and improving f^0 (an initial evaluation function) using the following equation:

$$f^{i+1}(s) := \min_{a \in A(s)} [c(s, a) + \sum_{s' \in S} p(s'|s, a) f^i(s')], \quad (3)$$

where i is the i th iteration. These backups are performed for all states $s \in S$ at each iteration. Value Iteration stops when the current error is less than ϵ . Finally, the ϵ -optimal policy can be extracted by:

$$\pi(s) := \arg \min_{a \in A(s)} [c(s, a) + \sum_{s' \in S} p(s'|s, a) f^i(s')], \forall s \in S. \quad (4)$$

Asynchronous dynamic algorithms only perform updates for a subset of S at each iteration. Examples of asynchronous dynamic programming algorithms for solving SSP-MDPs are: RTDP [1], LRTDP [4], BRTDP [13] and Bayesian RTDP [23]. RTDP is a heuristic search algorithm that runs simulations from the initial state until it reaches the goal state [4]. LRTDP [4], BRTDP [13] and Bayesian RTDP [23] are variants of RTDP. LAO* [10] is also a heuristic search algorithm that searches on an explicit graph, but the computation involves the states that are reachable following the best current solution starting from the initial state [11].

2.2 Bounded-parameter Markov decision process

A Bounded-Parameter SSP-MDP (BSSP-MDP) is an extension of an SSP-MDP, where S, s_0, S_g, A and c are the same as SSP-MDPs, but the transition probability function is defined over closed intervals of real numbers, i.e.

$P(s'|s, a) = [l, u]$ with $l, u \in \mathbb{R}$, where l stands for the lower bound of P and u stands for the upper bound. There are three reasons for adopting this approach: (i) to represent uncertainty in the model [3, 9, 21]; (ii) to model the occurrence of unpredictable events [26]; and (iii) to allow similar states of an MDP to be aggregated into a single aggregate state [21]. Three conditions must be satisfied to ensure that P only allows well-formed functions [21]:

- i. The interval must be in the range $0 \leq l \leq u \leq 1$.
- ii. The sum of the lower bounds of P over all the next s' states for the pair (s, a) must be less than, or equal to, 1:

$$\sum_{s' \in S} P^{lower}(s'|s, a) \leq 1, \forall s \in S, \forall a \in A, \quad (5)$$

where P^{lower} is the transition function P which only has lower bound values.

- iii. The sum of the upper bounds of P over all the next s' states for the pair (s, a) must be greater than, or equal to, 1:

$$\sum_{s' \in S} P^{upper}(s'|s, a) \geq 1, \forall s \in S, \forall a \in A, \quad (6)$$

where P^{upper} is the transition function P which only has upper bound values.

There are several optimization criteria for defining the value of a policy in a BSSP-MDP. In this work, our aim was to find the best solution in the worst-case scenario, (also known as a *robust solution*). Under this criterion, the agent (the minimizer) selects the actions that minimize its future expected cost by assuming that the adversarial agent (the maximizer) will choose the probability that maximizes the agents future expected cost.

As for SSP-MDPs, there are special assumptions that must be satisfied by BSSP-MDPs to ensure that an optimal policy can exist. Since BSSP-MDP is a subclass of SSP MDP-IP [17], the following assumptions for BSSP-MDPs must be made:

Assumption 3 There is at least one proper policy for the minimizer agent rooted at s_0 , i.e. a policy that leads to a goal state with probability 1 for any possible initial state and for any possible probability chosen by the maximizer.

Assumption 4 Any arbitrary policy π for the minimizer agent and any probability that the maximizer chooses which does not lead to the goal with probability 1, must have cost equal to ∞ .

Theorem 1 Given a BSSP-MDP that satisfies Assumptions 1 and 2, there is at least one stationary optimal policy with

a value function that is the solution of the Bellman equation, given by:

$$f^*(s) = \min_{a \in A(s)} \max_{p \in P} [c(s, a) + \sum_{s' \in S} p(s'|s, a) f^*(s')], \quad (7)$$

where p is a real number in the interval of P .

The proof of this theorem follows from the proof of the same theorem for SSP-MDP-IP (which is based on SSP Games[25]), since BSSP-MDP is a sub class of SSP-MDP-IP [17].

An optimization solver can be used to find the exact probability p in the maximization step in Equation 7, (as is used by SSP MDP-IPs algorithms [17]). Givan [21] proposed a more efficient optimal greedy algorithm, called WorstModel (Algorithm 1) [21]. Given a pair (s, a) , this algorithm sorts all the reachable states s' in decreasing order of their current evaluation function f^i (Algorithm 1, Line 2). Then, for each state s' the probabilities are [21]:

$$p(s'_i) = \begin{cases} P_i^{upper} & \text{if } i < r, \\ 1 - (\sum_{j=1}^{r-1} P_j^{upper} + \sum_{j=r+1}^k P_j^{lower}) & \text{if } i = r, \\ P_i^{lower} & \text{if } i > r, \end{cases} \quad (8)$$

where k is the total number of states s' and r is the first index that satisfies the following equation:

$$\sum_{i=1}^r P_i^{upper} + \sum_{i=r+1}^k P_i^{lower} \geq 1. \quad (9)$$

The probabilities on Equation 8 are defined, in Algorithm 1, Lines 7, 11 and 13 respectively.

Robust Value Iteration [25], Robust LRTDP [20] and Robust LAO* [24] are algorithms that call Algorithm 1 to find a robust ϵ -optimal solution for BSSP-MDPs, which are briefly described in the next sections.

2.2.1 Robust value iteration

The Robust Value Iteration algorithm [25] iteratively improves an arbitrary initial evaluation function by performing backups in the form:

$$f^{i+1}(s) = \min_{a \in A(s)} \max_{p \in P} [c(s, a) + \sum_{s' \in S} p(s'|s, a) f^i(s')]. \quad (10)$$

In Equation 10, the maximizer chooses a probability ($p \in P$) by calling Algorithm 1 for each pair (s, a) .

Algorithm 1 Worst Model for the pair (state,action) [21]

```

1: procedure WORSTMODEL( $s$ : state,  $a$ : action)
2:    $S' = (s'_1, \dots, s'_k) = s.SORTNEXTSTATES(a)$ 
3:    $i = 1$ 
4:    $bound = \sum_{s' \in S'} P^{lower}(s'|s, a)$ 
5:   while ( $bound - P_i^{lower} + P_i^{upper} < 1$ ) do
6:      $bound := bound - P_i^{lower} + P_i^{upper}$ 
7:      $p(s'_i|s, a) := P_i^{upper}$ 
8:      $i := i + 1$ 
9:   end while
10:   $r = i$ 
11:   $p(s'_r|s, a) := 1 - (bound - P_r^{lower})$ 
12:  for all  $m \in \{r + 1, \dots, k\}$  do
13:     $p(s'_m|s, a) := P_m^{lower}$ 
14:  end for
15: end procedure

```

2.2.2 Robust LRTDP

The Robust LRTDP algorithm [20] is a sample-based algorithm that performs simulations or trials, each one starting in the initial state s_0 and ending in a goal state. Each state visited in a simulation has the evaluation function updated considering the worst model, by calling Algorithm 1. Simulations are executed until the initial state s_0 has converged. This happens when all the states that are reachable through the greedy policy from s_0 have their evaluation function changed by less than ϵ .

2.2.3 Robust LAO*

Robust LAO* [24] is a heuristic search algorithm that finds an optimal policy without evaluating all the states. It executes three main steps: (i) expansion; (ii) revision; and (iii) convergence test. The idea of Robust LAO* is to find a path from the initial state s_0 to a goal state throughout expanding reachable states, according with the current greedy policy, in an explicit graph G' . This is the *expansion* step. Following this, it updates the value function $f(s)$ of each state on the best partial solution graph. This is the *cost revision* step. The evaluation function is updated by executing Robust Value Iteration (or Robust Policy Iteration) for the expanded state s and for all its ancestors (states that reach s by following the current greedy policy). When there are no more states on the fringe of the best solution graph, the *convergence* test step is performed.

The convergence proof of Robust LAO* follows from the convergence proof of LAO* [10] for SSP-MDPs. However, it should be taken into account that there are imprecise probabilities.

Proposition 1 *If the initial evaluation function f^0 used in the expansion step is admissible, the admissibility of the evaluation function is maintained during the execution of the Robust LAO*, i.e., if $f^0(s) \leq f^*(s), \forall s \in S$ then $f(s) \leq f^*(s), \forall s \in S$ at every point in the algorithm.*

Proof We will prove by induction that $f^t(s) \leq f^*(s), \forall s \in G$ at any time t , where G is the implicit graph, i.e., the graph specified by the initial state and the transition function.

Base Case: For $t = 0$, $f^0(s) \leq f^*(s), \forall s \in G$ is true owing to the admissibility of the given heuristic evaluation function.

Induction Step: By the induction hypothesis at some time t in the algorithm, $f^t(s) \leq f^*(s), \forall s \in G$, then if a backup is performed for some state s ,

$$\begin{aligned} f^{t+1}(s) &= \min_{a \in A(s)} [c(s, a) + \max_{p \in P} (\sum_{s' \in S} p(s'|s, a) f^t(s'))] \\ &\leq \min_{a \in A(s)} [c(s, a) + \max_{p \in P} (\sum_{s' \in S} p(s'|s, a) f^*(s'))] \\ &= f^*(s), \end{aligned} \tag{11}$$

i.e., $f^{t+1}(s) \leq f^*(s), \forall s \in G$ at any time t . \square

Theorem 2 *The evaluation function $f(s)$ converges within ϵ of $f^*(s)$ for every state s of the best solution graph, after a finite number of iterations of Robust LAO* if the heuristic evaluation function f^0 used in the expansion step is admissible and a Robust Value Iteration is used in the cost revision step.*

Proof This proof is similar to LAO* convergence proof [10]. Since graph G is finite, the expansion of the explicit graph G' must eventually stop looking for a solution graph. Additionally, since Robust Value Iteration is applied into the solution graph, $f(s)$ must also converge within ϵ of $f^*(s)$ for every state s of the best solution graph, after a finite number of iterations. This is a result of the proof of convergence of the Robust Value Iteration algorithm [25]. Note that the proof of convergence of Robust Value Iteration given for sequential SSP games, is also valid for BSSP-MDP, because BSSP-MDPs can be seen as a particular case of sequential SSP games [17]. \square

3 Robust ILAO*

The main drawback of Robust LAO* is the fact that many states are evaluated more often than necessary. In Robust

ILAO* two key improvements have been made to restrict the number of times that states are evaluated (similar to what has been proposed for MDPs [10]). First, we expand all the states that are on the fringe of the best partial solution graph. Second, we revised the estimated cost of the expanded states and its ancestors *only once* (i.e., without calling the Robust Value Iteration algorithm at this step).

Robust ILAO* (Algorithm 2) merges the cost revision step with the expansion step and can be summarized in two new main steps described as follows:

- i. *DepthFirstSearch* (Algorithm 3) (called in Line 6 of Algorithm 2) performs both the *expansion* and the *cost revision step*. It expands the best partial solution graph, updates the evaluation function and marks the best actions. A depth-first search is carried out in the best solution graph (Algorithm 3, Line 4-6) and some states are added to the explicit graph (Algorithm 3, Line 8). These states are all the successor states of a leaf state that is not a goal state (also called a tip state). At the end of each depth-first search, the evaluation function is updated (Algorithm 3, Line 10) and the best action is marked (Algorithm 3, Line 11) for each visited state.
- ii. A *convergence step* is performed by calling Robust Value Iteration (Algorithm 4) in Line 9 of Algorithm 2. Algorithm 4 updates the evaluation function of all the states that belong to the best solution graph. It also makes two convergence tests: (a) if the maximum error falls below a given ϵ (Algorithm 4, Line 4), this means the ϵ -optimal solution has been found; and (b) if the best solution graph has changed so that it has an unexpanded leaf state (Algorithm 4, Line 10), this means Robust ILAO* has not been converged yet and both steps of Robust ILAO* must be executed again.

Algorithm 2 Robust ILAO*

```

1: procedure ROBUSTILAO( $G'$ : Graph)
2:    $G' := (\{s_0\}, \emptyset)$ 
3:   convergence := false
4:   while (not convergence) do
5:     while ( $G'$ .hasNonTerminalTipState()) do
6:       depthFirstSearch( $G', s_0$ )
7:        $G'$ .clearVisitedMark()
8:     end while
9:     convergence := robustValueIteration( $G'$ )
10:  end while
11: end procedure

```

Algorithm 3 Depth-First Search for Robust ILAO*

```

1: procedure DEPTHFIRSTSEARCH( $G'$ : Graph,  $s$ : State)
2:    $s.visited := true$ 
3:   if ( $s.expanded$ ) then
4:     for all ( $s' \in (s.getSuccessors(s.bestAction) \cap G'.verticesNotVisited())$ ) do
5:        $depthFirstSearch(G', s')$ 
6:     end for
7:   else
8:      $expandState(G', s)$ 
9:   end if
10:   $f(s) := \min_{a' \in A(s)} Q(s, a')$ 
11:   $s.bestAction := arg \min_{a' \in A(s)} Q(s, a')$ 
12: end procedure

```

It is worth noting that the WorstModel method is called every time that $Q(s, a)$ is computed (Algorithm 5, Line 2) and the `getSuccessors` method must consider s' as a successor of s by action a if $P(s'|s, a) \neq [0, 0]$.

Example 4 [10] In the simple example showed in Fig. 1(a), a robot walks in a grid 3×3 and the states are the cells, which are labeled from 1 to 8. The agent starts in the initial state 1 and has to find a path to reach the goal state 8. In any state, the agent can choose among one of the actions, go up (U), go down (D), go left (L) and go right (R), each of which can reach its desired effect (i.e., up, down, left and right, respectively) with a probability interval $[0.5, 0.7]$; otherwise the agent remains in the same state with a probability interval $[0.3, 0.5]$. If the agent hits a wall, he will also remain in the same position. Whenever the agent chooses an action in a nonggoal state, he must pay a cost of 1 unit for actions D, L and R and a cost of 0.5 for action U.

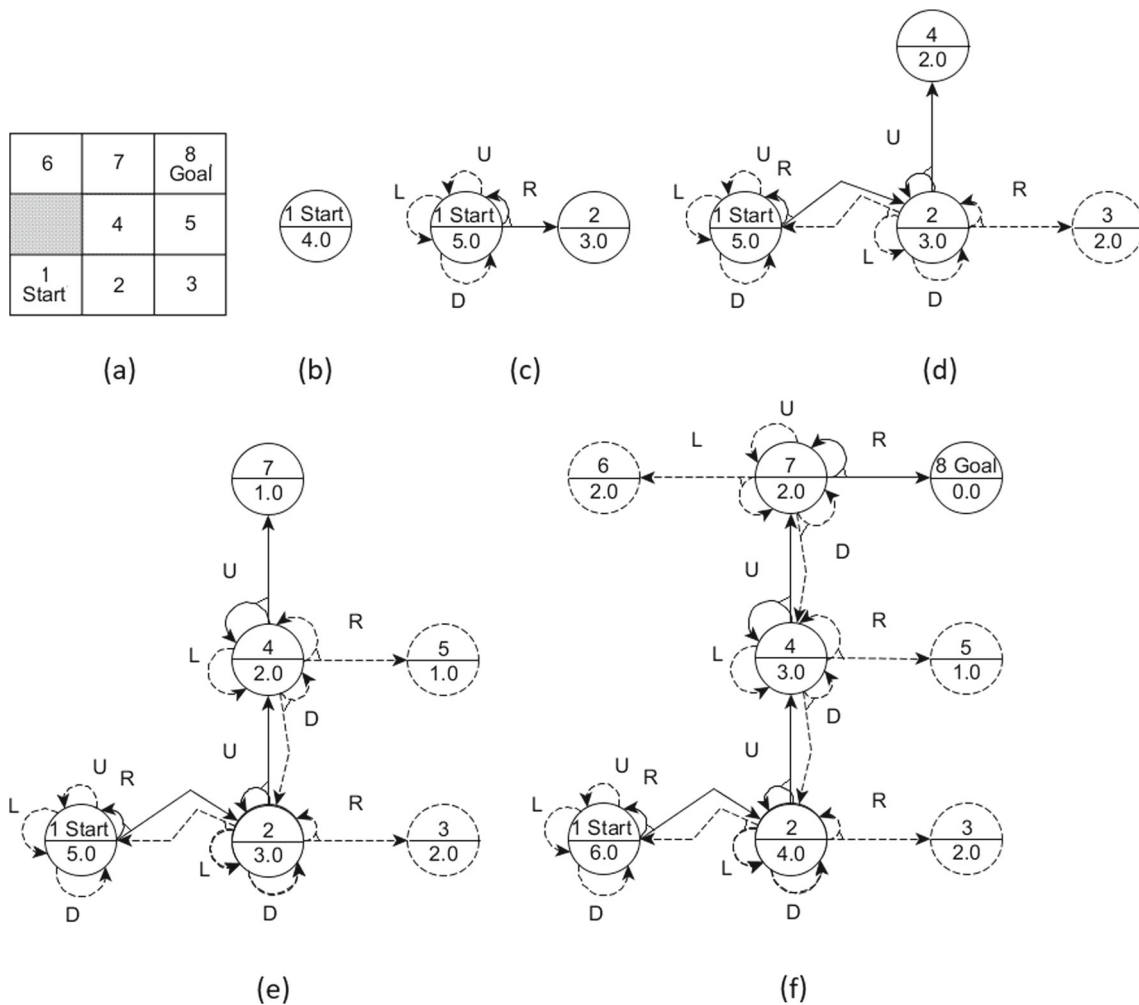


Fig. 1 (a) The grid where the agent starts in the initial state 1 and has to find a path to reach the goal state 8 (Example 4); (b) through (f) are the steps that Robust ILAO* performs to solve this problem. Each node represents an state and hyper-edges correspond to actions and

transitions. The label and the estimated cost of each state are shown in the upper and lower half of the node, respectively. Solid lines show the best partial solution graph, and dashed lines show the rest of the explicit graph created.

Algorithm 4 Robust Value Iteration for Robust ILAO*

```

1: procedure ROBUSTVALUEITERATION( $G'$ : Graph,  $\epsilon$ :
   float)
2:    $i := 0$ 
3:    $G'' := G'.bestSolutionGraph()$ 
4:   while ( $max\_error \leq \epsilon$ ) do
5:     for all ( $s \in G''$ ) do
6:        $f^{i+1}(s) := \min_{a \in A(s)} Q(s, a)$ 
7:        $s.bestAction := \arg \min_{a \in A(s)} Q(s, a)$ 
8:     end for
9:      $G'' := G'.bestSolutionGraph()$ 
10:    if ( $G''.hasUnexpandedTipState()$ ) then
      return false
11:    end if
12:     $max\_error := \max_{s \in G''} |f^{i+1}(s) - f^i(s)|$ 
13:     $i := i + 1$ 
14:  end while
15:  return true
16: end procedure

```

Algorithm 5 Some auxiliar methods of Robust ILAO*

```

1: procedure Q( $s$ : State,  $a$ : Action)
2:   WORSTMODEL( $s, a$ )
3:   return  $c(s, a) + \sum_{s' \in S} p(s'|s, a) f(s')$ 
4: end procedure
   ▸


---


5: procedure EXPANDSTATE( $G'$ : Graph,  $s$ : State)
6:    $S' := s.getAllSuccessors()$ 
7:    $f(s') := \begin{cases} 0 & \text{if } s' \text{ is a goal state} \\ h(s') & \text{otherwise} \end{cases}, \forall (s' \in S')$ 
8:    $G'.addSuccessors(s, S')$ 
9:    $s.expanded := true$ 
10: end procedure

```

Figure 1(b) through (f) shows the steps that Robust ILAO* performs to solve Example 4. In this example, it is used the length of a deterministic path from the state s to the goal state 8 as an admissible heuristic for s . Fig. 1(b) shows the explicit graph that consists of the start state 1 with 4 as its heuristic value. Fig. 1(c) shows the result of expanding the start state 1. The successor state 2 is added to the explicit graph with 3 as its heuristic value and then the evaluation function of state 1 is updated. Finally, the best action Right (R) for state 1 is marked. Fig. 1(d) through (f) shows the result of expanding state 2, state 4 and state 7, respectively. Each state is the only on the fringe of the best partial solution graph, respectively.

After expanding state 7 ILAO* terminates because the convergence test returns true, i.e., the best solution graph has not changed and the maximum error falls below $\epsilon = 10^{-16}$.

4 Experiments

The performance of Robust ILAO*, Robust LAO* [24], Robust LRTDP [20] and Robust Value Iteration [25] algorithms were compared by carrying out some experiments with modified versions of Grid World [10] and Race Track [1] benchmark planning domains, as described in Sections 4.1 and 4.2.

4.1 Grid world domain

In the Grid World domain [10], a robot walks in a grid $n \times m$, where n is the number of rows and m is the number of columns. The agent starts in the initial state $(1, 1)$ and has to find a path to reach the goal state (m, n) . In any state, the agent can choose among one of the actions, *go up*, *go down*, *go left* and *go right*, each of which can reach its desired effect (i.e., up, down, left and right, respectively) with a probability interval $[0.8, 1]$; otherwise the agent ends up in one of the adjacent cells with a probability interval $[0.05, 0.25]$. If the agent hits a wall, he will remain in the same position. Whenever the agent chooses an action in a state, he must pay a cost of 0.04 units.

4.2 Race track domain

In the Race Track domain [1], there is a race car on a track represented by a grid $n \times m$, where n is the number of rows and m is the number of columns. The car begins on the starting line and stops at the goal line. The state is determined by the position of the car (x, y) and its speed $\vec{v} = \langle v^x, v^y \rangle$ in the horizontal and vertical axis, respectively. In any state, the car can try to alter its speed by adding the vector $\vec{u} = \langle u^x, u^y \rangle$, where u^x and u^y are both in the set $\{-1, 0, 1\}$. However, the resulting velocity \vec{v} is given by:

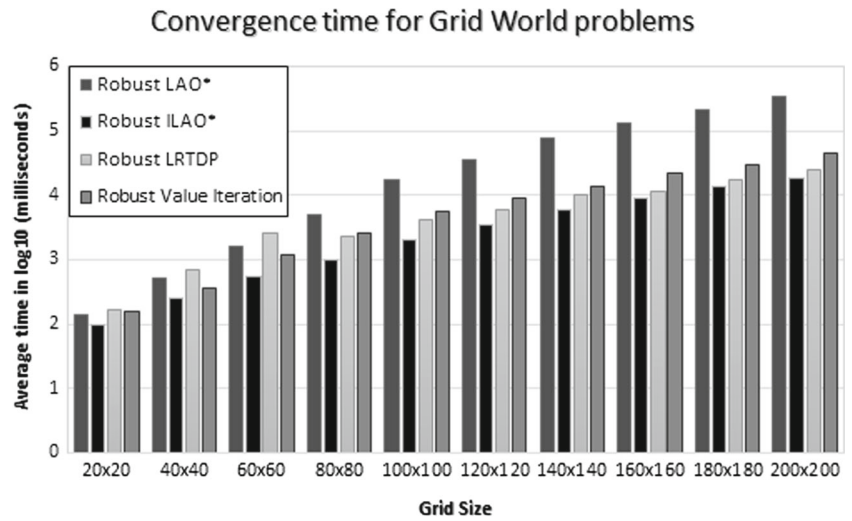
$$\vec{v} = \begin{cases} \langle 0, 0 \rangle & \text{if the car hits a wall} \\ \vec{v} & \text{with probability } [0.1, 0.3] \\ \vec{v} + \vec{u} & \text{with probability } [0.7, 1.0] \end{cases} \quad (12)$$

Moreover, if the car hits a wall, it remains at the same position. Whenever the car tries to change its velocity in a state, a cost of 1.0 unit must be paid.

4.3 Convergence time

In Fig. 2, the convergence time of the 4 algorithms is compared for 10 instances of the Grid World problem

Fig. 2 Convergence time of Robust LAO*, Robust ILAO*, Robust LRTDP and Robust Value Iteration algorithms for 10 Grid World problems. Note that the scale of the time is \log_{10} .



(20×20 to 200×200 size). Note that in this domain Robust ILAO* was up to 15.8 times faster than Robust LAO*. In fact, Robust LAO* converges slower than Robust Value Iteration for instances larger than 40×40 . This is because Robust LAO* calls a Robust Value Iteration algorithm after each expanded state, performing too many robust updates (Equation 10) which are very costly. Fig. 2 also shows that Robust ILAO* was up to 4.7 times faster than Robust LRTDP (considered the state-of-the art for BSSP-MDPs).

In Fig. 3, the convergence time of the 4 algorithms is compared for 6 instances of the Race Track domain

(problems of 1.2×10^3 to 4×10^5 states). It should also be noted that in this domain, Robust ILAO* was up to 4 times faster than Robust LRTDP and up to 22 times faster than Robust LAO*. The Race Track is a good domain to show the efficiency of heuristic algorithms that do not evaluate the entire state space. This domain has a large number of states but only a fraction of these will be part of an optimal policy. One reason why Robust ILAO* outperforms Robust LRTDP is that a sizable proportion of states in the optimal policy have low-probability and thus they are slowly evaluated by Robust LRTDP.

Fig. 3 Convergence time of Robust LAO*, Robust ILAO*, Robust LRTDP and Robust Value Iteration algorithms for six Race Track problems. Note that the scale of the time is \log_{10} .

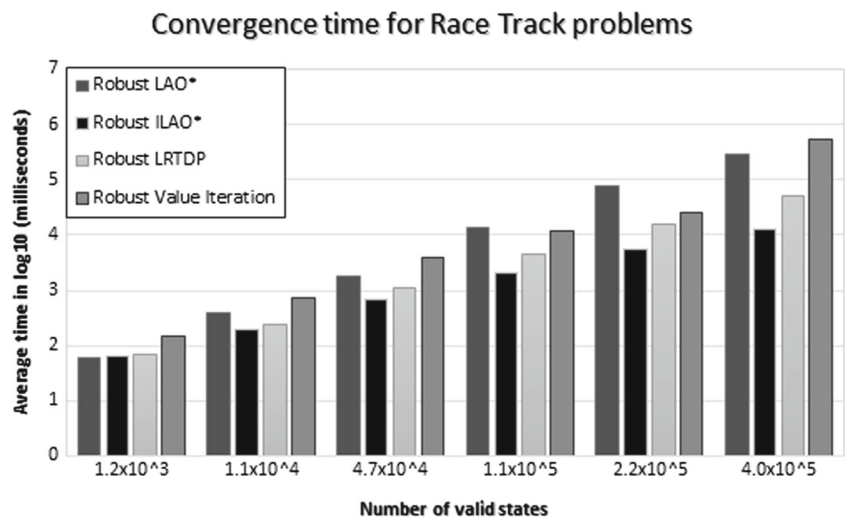
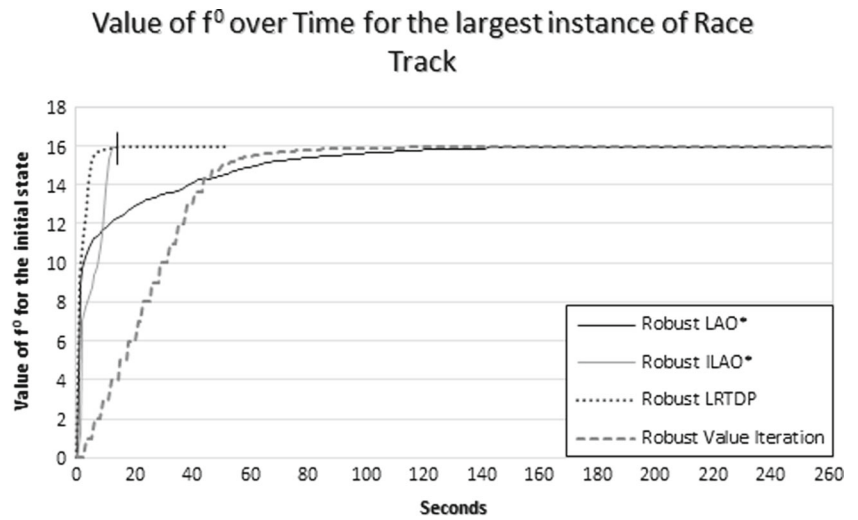


Fig. 4 A comparison of the convergence behavior of Robust LAO*, Robust ILAO*, Robust LRTDP and Robust Value Iteration on the Race Track problem with 4×10^5 states.



4.4 Convergence behavior

In Fig. 4, the convergence behavior of the 4 algorithms is compared using the largest instance of the Race Track problem. It should be noted that, although Robust LRTDP improves the solution more quickly, Robust ILAO* converges to an ϵ -optimal solution earlier, after 12 seconds (small vertical line), whereas Robust LRTDP continues to run for more 40 seconds until the convergence. The convergence behavior of Robust ILAO* and Robust LRTDP suggests that a combined strategy that focuses search in high-probability paths first, and low-probability paths when

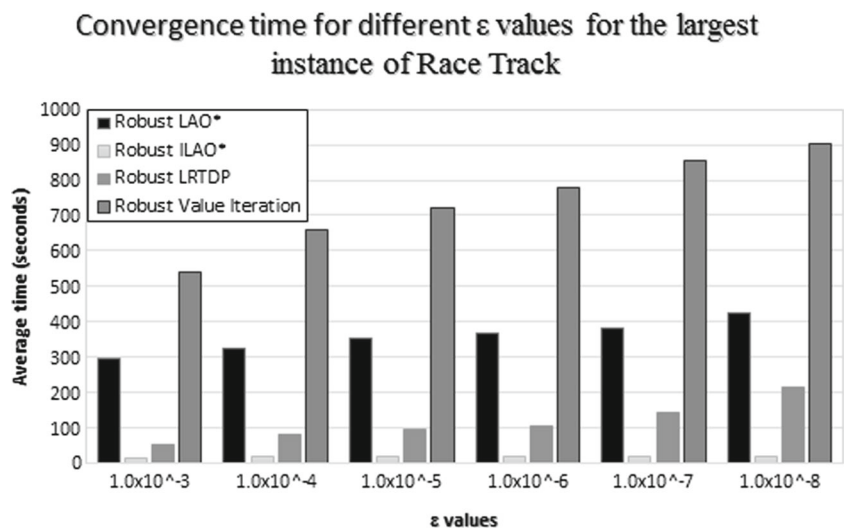
the solution is near to convergence, may result in a better overall convergence time.

In the first three experiments (Fig. 2, 3 and 4), ϵ was set to 10^{-3} for all instances and algorithms.

4.5 Convergence time versus precision (ϵ values)

In Fig. 5, the convergence time of the algorithms for the largest instance of the Race Track problem were compared, varying the value of ϵ . It was found that Robust ILAO* is faster when ϵ gets smaller. In fact, when $\epsilon = 10^{-8}$, Robust ILAO* was 10 times faster than Robust LRTDP.

Fig. 5 Convergence time of Robust LAO*, Robust ILAO*, Robust LRTDP and Robust Value Iteration algorithms for different ϵ values for the race track problem with 4×10^5 states.



5 Conclusion

In this paper, we have sought to provide a complete definition of BSSP-MDP problems, which includes the assumptions that must be made to guarantee the existence of a robust stationary optimal policy. Although other works have proposed algorithms for BSSP-MDPs, they can fail without these assumptions.

We also propose an efficient algorithm to solve these problems considering the worst-case model, called Robust ILAO*. We have been able to prove the ϵ -convergence of the Robust LAO* algorithm when the initial value of all the states is admissible. In addition, it has been shown in empirical terms that Robust ILAO* has a good performance for Grid World and Race Track benchmark domains when compared with the state-of-the-art Robust LRTDP algorithm.

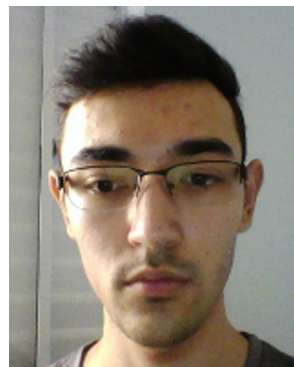
It should be pointed out that attempts made in previous work to develop a robust version of LAO* [24] have failed to prove convergence or to compare the Robust LAO* with Robust LRTDP [20]. In fact, Robust LAO* showed the worst performance in the analyzed domains, and was even worse than the Robust Value Iteration in most of the instances.

In a future work we intend to analyze the convergence behavior of a hybrid algorithm that first applies the Robust LRTDP strategy and, then employs Robust ILAO* algorithm when near convergence. This will combine the best features of each algorithm.

Acknowledgments We thank the São Paulo Research Foundation for the financial support (FAPESP grant #2015/01587-0).

References

- Barto A, Bradtke S, Singh S (1995) Learning to act using Real-Time dynamic programming. *Artif Intell* 72:81–138
- Datta A, Choudhary A, Bittner ML, Dougherty ER (2003) External control in Markovian genetic regulatory networks. *Mach Learn* 52(1-2):169–191
- Tewari A, Bartlett PL (2007) Bounded parameter Markov decision processes with average reward criterion. Springer, Berlin Heidelberg, pp 263–277. *proceedings of Learning Theory: 20th Annual Conference on Learning Theory*
- Bonet B, Geffner H, Labeled RTDP (2003) Improving the convergence of real-time dynamic programming in *Proc. AAAI Press:12–21*. 13th International Conf. on Automated Planning and Scheduling Trento: Italy:
- White III CC, El-Deib HK (1994) Markov decision processes with imprecise transition probabilities. *Oper Res* 42(4):739–749
- Bertsekas D (1995) *Programming, Dynamic Control, Optimal* Athena scientific, belmont MA
- Bertsekas DP, Tsitsiklis JN (1991) An analysis of stochastic shortest path problems. *Math Oper Res* 16(3):580–595
- Bryce D, Verdicchio M, Kim S (2010) Planning interventions in biological networks. *ACM Trans Intell Syst Technol* 1(2):111–140
- Wu D, Koutsoukos X (2008) Reachability analysis of uncertain systems using bounded-parameter Markov decision processes. *Artif Intell* 172(8):945–954
- Hansen E, Zilberstein S (2001) LAO*: A heuristic search algorithm that finds solutions with loops. *Artif Intell* 129:35–62
- Hansen E, Zilberstein S (1999) Solving Markov decision problems using heuristic search. *AAAI Technical Report:42–47*
- Trevizan FW, Cozman FG, de Barros LN (2007) Planning under risk and knightian uncertainty. Hyderabad, India, pp 2023–2028. *Inproceedings of International Joint Conferences on Artificial Intelligence*
- McMahan HB, Likhachev M, Gordon GJ (2005) Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees:569–576. *Inproceedings of the 22nd international conference on Machine Learning (ICML '05)* New York NY
- Satia JK, Lave Jr RE (1970) Markovian decision processes with uncertain transition probabilities. *Oper Res* 21:728–740
- Delgado KV, Sanner S, de Barros LN (2011) Efficient solutions to factored MDPs with imprecise transition probabilities. *Artif Intell* 175(9-10):1498–1527
- Delgado KV, de Barros LN, Cozman FG, Sanner S (2011) Using mathematical programming to solve factored Markov decision processes with imprecise probabilities. *Int J Approx Reason (IJAR)* 52(7):1000–1017
- Delgado KV, de Barros LN, Dias DB, Sanner S (2015) Real-time dynamic programming for Markov Decision Processes with imprecise probabilities. *Artif Intell* 230:192–223
- Hauskrecht M (1997) Dynamic decision making in stochastic partially observable medical domains. *Lecture Notes in Artificial Intelligence* 1211:296–299. *Ischemic heart disease example*, 6th Conference on Artificial Intelligence in Medicine, Springer
- Puterman ML, *Processes MarkovDecision* (1994) *Discrete Stochastic Dynamic Programming*, 1st ed, New York, NY, USA: John Wiley & Sons Inc
- Buffet O, Aberdeen D (2005) Robust planning with (l)RTDP, in *Proc. of the 19th. Int Joint Conf:1214–1219*. on *Artificial Intelligence (IJCAI05)*
- Givan R, Leach S, Dean T (2000) Bounded-parameter Markov Decision Processes. *Artif Intell* 122:71–109
- Pal R, Datta A, Dougherty ER (2008) Robust intervention in probabilistic boolean networks. *IEEE Trans Signal Process* 56(3):1280–1294
- Sanner S, Goetschalckx R, Driessens K, Shani G (2009) Bayesian realtime dynamic programming, in *21st International Joint Conference on Artificial Intelligence (IJCAI-09)*. Kaufmann Publishers Inc., San Francisco, CA, pp 1784–1789
- Cui S, Sun J, Yin M, Lu S (2006) Solving uncertain Markov decision problems: an Interval-Based method, second international conference. *ICNC:948–957*
- Patek SD, Bertsekas DP (1999) Stochastic shortest path games. *SIAM J Control Optim* 37:804–824
- Witwicki SJ, Melo FS, Capitan J, Spaan MTJ (2013) A flexible approach to modeling unpredictable events in MDPs. *ICAPS:260–268*. *proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*



Daniel A. M. Moreira is a M.Sc student in Computer Science at the University of São Paulo, Brazil. His research focuses on algorithms for probabilistic planning in artificial intelligence. During his studies he receives a scholarship from CAPES (Coordination for the Improvement of Higher Education Personnel).



Karina Valdivia Delgado is an assistant professor at School of Arts, Sciences and Humanities of University of São Paulo, Brazil. She received her M. Sc. and Ph.D. degrees in Computer Science in 2005 and 2010, respectively, from University of São Paulo, Brazil. She received the Silver IJAR Young Researcher Award for demonstrating excellence in research on imprecise probabilities in 2009 granted by the International Journal of Approximate

Reasoning. Her research interests include sequential decision making, probabilistic planning in artificial intelligence, application of AI methods to education, inventory routing and vehicle routing.



Leliane Nunes de Barros is an associated professor at the Computer Science Department of University of São Paulo, Brazil. She received her Ph.D. degree from University of São Paulo, Brazil, in the area of Knowledge Engineering for Planning. Her research interests include automated planning, probabilistic planning, planning as model-checking and model-based diagnosis. She is a member of the Special Committee on Artificial Intelligence of the Brazilian Computer Society.