Advantage Based Value Iteration for Markov Decision Processes with Unknown Rewards

Pegah Alizadeh, Yann Chevaleyre, François Lévy
Institut Galilée - Université Paris 13
99, avenue Jean-Baptiste Clément
93430 Villetaneuse, France
Email: firstname.lastname@lipn.univ-paris13.fr

***Abstract*—This paper addresses approximating the optimal policy in *Markov Decision Process* with unknown rewards. The MDP is transformed into a *Vector-Valued MDP (VVMDP)*. We introduce a new interactive algorithm ABVI, whose principle is using value iteration on VVMDPs and querying the user when necessary. This algorithm uses classification method to reduce the number of proposed queries. We integrate value iteration with querying the user to select appropriate backups. In this paper, our goal is to accelerate the value iteration algorithm and to reduce the number of queries.**

1. Introduction

Markov decision process (MDP) is a model for solving sequential decision problems where a tutor interacts with environment and adapts his policy by taking numerical reward signals into account. If MDP parameters are given numerically, most *reinforcement learning (RL)* agent can autonomously learn any task.

Finding the shortest route between two points A and B on the GPS which monitors some statistics about traffic jams is an example for MDPs with uncertain reward values. In this case, the MDP specification remains as a bottleneck because when a user interacts with the environment, it receives some quantitative reactions that should be interpreted into numerical rewards (punishments). Besides the user may have some preferences in his mind, for instance: he may like to have a bakery in his route, or he appreciates to walk inside botanic parks on the way. Therefore, the optimal shortest route (*optimal policy*) should be updated following any user's preference expressed in a given Imprecise Reward MDP.

A considerable number of researchers have studied the direct computation of robust optimal policies for *Imprecise reward MDPs (IRMDPs)* [3], [5], [8], [13]. In some works the problem has been extended to a structured representation of rewards [6], [11], [12] for simplifying calculations. [6] relies on linear approximation for reward fuctions, while [11], [12] utilize a vectorial representation namely *Vector-Valued MDP (VVMDP)*. In this paper, we extend MDP with unknown rewards to a VVMDP with a *Weng's vector structure* on rewards [11]. The VVMDP is generated by transforming each unknown reward to a dot product between an unknown reward vector and a weights vector. The determination of unknown weight vectors (noted $\bar{\lambda}$), is thus separated from the accumulation of (unknown) rewards along histories, which is known from the history. This helps us to calculate the optimal value function $\bar{V}^*(s)$ for each state s, and should get the final value function by: $V^*(s) = \bar{\lambda} \cdot \bar{V}^*(s)$ if the exact value of $\bar{\lambda} \in \Lambda$ was given.

Since $\bar{\lambda}$ value is not certain, we present a modified value iteration method in interaction with the user to eliminate unwanted $\bar{\lambda}$s inside Λ and approximate optimal $\bar{V}^*$ vector. For that, we handle two separate polytopes: vectorial value functions polytope and weights polytope. Assume $\bar{\mathcal{V}}$ is the polytope for admissible optimal $\bar{V}$s and is computable. [12] implements a value iteration method on VVMDP while refining the knowledge about possible value of weights by querying a tutor whenever it is required. In general, the classical value iteration method consists of many comparisons among vectors: each iteration compares close vectors with each other.

The main contribution of this paper is considering *advantage* vectors in any iteration i.e. the individual improvement brought up by each change of action in a given state. We cluster advantages according to their directions and produce new vectors representative of each cluster. This basic idea allows us to compare several vectors relying on the same user query, thus reducing the volume of interaction.

Our final contribution is reporting some experiments that indicate how the presented method is effective after asking less questions to the tutor. However the overall number of proposed queries in our approach is higher than the similar previous method [12], while a majority of queries are requested after obtaining a reasonably precise solution.

2. Background

This section concerns the required components of VVMDPs and discusses some mathematical and notational subtleties.

2.1. Markov Decision Process

A MDP is formally defined by a tuple $(S, A, p, r, \gamma, \beta)$ where S is a set of states, A is a set of actions, $p(s'|s,a)$ encodes the probability of going to state s' when being in state s and choosing action a, $r : S \times A \longrightarrow \mathbb{R}$ is a reward function, $\gamma \in [0, 1[$ is a discount factor and β is a distribution on initial states. We concentrate on infinite horizon MDPs with finite number of states and actions.

A stationary policy $\pi : S \longrightarrow A$ prescribes to take the action $\pi(s)$ when in state s. For each policy π, the expected discounted sum of rewards for policy π in s is a function $V^\pi : S \longrightarrow \mathbb{R}$ which is defined by

$$\forall s,\ V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^\pi(s') \quad (1)$$

As $r(s, \pi(s))$ is bounded and $\gamma < 1$, $V^\pi(s)$ is the *discounted value function* of π in s, and is a solution of the fixed-point equation (1).

Taking into account the initial distribution β, each policy has an expected value function equal

$$\mathbb{E}_{s\sim\beta}[V^\pi(s)] = \sum_{s \in S} \beta(s) V^\pi(s) = \beta \cdot V^\pi$$

The optimal policy π^* is the one with the greatest expected value among all possible policies of this MDP:

$$\pi^* = \text{argmax}_\pi\ \beta \cdot V^\pi \quad (2)$$

To compute the optimal policy and its expected value, an approximation is iterated with the help of the auxiliary Q^π function:

$$\begin{array}{rcl} Q^{\pi_t}(s,a) & = & r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_t}(s') \\ \pi_{t+1}(s) & = & \text{argmax}_a\ Q^{\pi_t}(s,a) \\ V^{\pi_{t+1}}(s) & = & Q^{\pi_t}(s, \pi_{t+1}(s)) \end{array} \quad (3)$$

All states are updated during each iteration until satisfying the stopping criteria. Value iteration is the algorithm which applies this iterative approach until the value improvement $||\beta \cdot V^{\pi_t} - \beta \cdot V^{\pi_{t+1}}||$ is under a given threshold.

Consider now, the difference between $Q^\pi(s,a)$ and $V^\pi(s)$:

$$d(s,a) = Q^\pi(s,a) - V^\pi(s)$$

This variation weighted by the initial distribution on the state is known as *Advantage* [2]

$$A(s,a) = \beta(s) d(s,a) = \beta(s)\{Q^\pi(s,a) - V^\pi(s)\}$$

so Equation 3 can be modified:

$$\pi_t(s) = \text{argmax}_a\ A_t(s,a)$$

Since this paper main idea uses advantages, we will investigate advantages and their properties in the following sections.

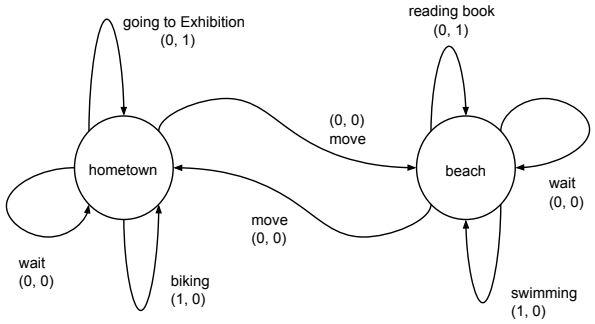


Figure 1: An example of small vector-valued MDP with 2 states and 6 actions.

2.2. Vector-Valued MDPs

When designing real cases as MDPs, specifying the reward function is generally a hard problem. For instance preferences stating which (state, action) pairs are good or bad should be interpreted into numerical costs. Note that even knowing all these preferences is time consuming. In order to tackle this complexity, we use a *MDP with imprecise reward values (IRMDP)*. IRMDP extends *Ordered Reward MDP (ORMDP)* and its transformation into VVMDP ([11], [12]). This paper handles nearly the same structure on IRMDP, at the difference that unknown rewards are not ordered as they are in [12].

An IRMDP is a MDP(S, A, p,r, γ, β) with unknown or exactly known rewards r, where unknown rewards need to be elicited. For that, let $E = \{\lambda_1, .., \lambda_{d-1}\}$ be a set of variables. It is initially only known that $\forall\ i,\ 0 \leq \lambda_i \leq 1$. E is the set of possible unknown reward values for the given MDP and we have $r(s,a) \in E \cup \mathbb{R}$.

We will match our IRMDP to a vectorial form, namely *Vector-Valued MDP (VVMDP)* [11]. For that, a vectorial reward selector function $\bar{r} : S \times A \longrightarrow \mathbb{R}^d$ is defined as[1] :

$$\bar{r}(s,a) = \begin{array}{l} (1)_j \text{ if } r(s,a) \text{ has the unknown value } \lambda_j\ (j < d) \\ x.(1)_d \text{ if } r(s,a) \text{ is known to be exactly } x. \end{array}$$

Let also $\bar{\lambda}$ be the vector $(\lambda_1, .., \lambda_{d-1}, 1)$. For all s and a, we have $r(s,a) = \sum_{i=1}^{d} \lambda_i . \bar{r}(s,a)[i]$, and hence any reward $r(s,a)$ is a dot product between two d-dimensional vectors:

$$r(s,a) = \bar{\lambda} \cdot \bar{r}(s,a) \quad (4)$$

Example 2.1. Suppose Figure 1 as a MDP with two states {hometown, beach} and six actions {swimming, reading book, going to exhibition, biking, wait, move}. Numerical value of rewards are not given in this model, instead it is known that every selected action in each state has only two effects: "sportive" indicated by λ_1 and "artistic" indicated by λ_2. Unknown rewards are transformed to vectorial rewards such that the first element represents sportive coefficient of activity, while

1. $(1)_j$ notes the d-dimensional vector $(0, .., 0, 1, 0, .., 0)$ having a single 1 in the j-th element.

the second one shows the artistic value. For instance, in $\bar{r}$(hometown, biking) vector the second element is zero, because biking is not an artistic activity. If $\bar{\lambda} = (\lambda_1, \lambda_2)$ vector is known numerically, we will have $\bar{r}$(hometown, biking) $= 1.0\lambda_1 + 0.0\lambda_2$.

By omitting the $\bar{\lambda}$ vector, we have a MDP$(S, A, p, \bar{r}, \gamma, \beta)$ with vector-valued reward function. Basic techniques of MDP's can be applied componentwise. The discounted value function of policy π is a function $\bar{V}^{\pi} : S \longrightarrow \mathbb{R}^d$ which provides in each state the discounted sum of reward vectors, and can be computed as

$$\bar{V}^{\pi}(s) = \bar{r}(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) \bar{V}^{\pi}(s') \qquad (5)$$

In order to find the maximum vector in equation 3, the iteration step need to compare value vectors with each other. To do this comparison, we use three possible methods which will be explained in Section 3.

Suppose that for each $s \in S$, $\bar{V}^{\pi}(s)$ is the discounted value function computed from equation 5. Taking into account the initial distribution, the vectorial expected value function is:

$$\bar{V}^{\pi} = \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}(s)] = \sum_{s \in S} \beta(s) \bar{V}^{\pi}(s)$$

Assume a numerical value for the $\bar{\lambda}$ vector. Considering equations 4 and 5, the scalar expected value function v^{π} can be computed as:

$$v^{\pi} = \sum_{i=1}^{d} \lambda_i . \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}](s)[i] = \bar{\lambda} \cdot \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}](s)$$

This provides a comparison between any two policies. Following this relation, restricting the values of $\bar{\lambda}$ prunes some discounted value functions as non maximal, and conversely, knowing which of two policies is prefered prunes values of $\bar{\lambda}$ which conflict with this preference.

So, finding an optimal policy for a MDP with unknown rewards boils down to explore the interaction between two separate d-dimensional admissible polytopes. The first one is a set of admissible vector-valued functions for the transformed VVMDP and the second one is a set of all possible weight vectors for the same VVMDP. In the following, the set of all possible weight vectors $\bar{\lambda}$ for the VVMDP is noted as Λ, the set of all possible policies as Π, and the set of their vectorial value functions as $\bar{\mathcal{V}}$ ($\bar{\mathcal{V}} = \{\bar{V}^{\pi} : \pi \in \Pi\}$). Without loss of generality, we assume that the Λ polytope is a d-dimensional unit cube:

$$\forall i = 1, \cdots, d \quad 0 \leq \lambda_i \leq 1$$

And since generating all $\bar{\mathcal{V}}$ members is expensive, we just calculate the required vectors inside the $\bar{\mathcal{V}}$ polytopes.

3. Vector-Valued Functions

To clarify the problem, suppose $\bar{\mathcal{V}}$ and Λ are given. Our aim is exploring the optimal value function as: $\bar{\lambda}^* \cdot \bar{V}^*$ such that $\bar{V}^* \in \bar{\mathcal{V}}$ is the optimal vectorial value function for the VVMDP matching our IRMDP, and $\bar{\lambda}^* \in \Lambda$ is the vector weight satisfying user preferences. To tackle this problem we must handle two types of questions: which $\bar{V}^*$ vectors inside $\bar{\mathcal{V}}$ should be selected, and how to explore $\bar{\lambda}^*$ according to user preferences.

The principal idea is producing $\bar{V}^*$ inside $\bar{\mathcal{V}}$ using value iteration on VVMDP, and exploring $\bar{\lambda}^* \in \bar{\mathcal{V}}$ by elicitation methods on the Λ polytope . Value iteration algorithm [12] on VVMDP needs to compare vectors. Every comparison which elicits a preference integrates a new cut on Λ polytope and eliminates part of the polytope according to user preferences.

Example 3.1. Return to Example 2.1, suppose we need to compare two vector-valued functions $\bar{V}^{\pi_i} = 0.5\lambda_1 + 0.5\lambda_2$ and $\bar{V}^{\pi_j} = 0.3\lambda_1 + 0.7\lambda_2$. The comparison between $0.5\lambda_1 + 0.5\lambda_2$ and $0.3\lambda_1 + 0.7\lambda_2$ reduces to the question $\lambda_1 > \lambda_2$. As a result a query of the form "Is $\lambda_1 > \lambda_2$?" is proposed to the user. It provides a cut on Λ polytope which splits it in two parts. Elicitation of user preferences among artistic or sportive privileges prunes one half of the polytope. For instance, if user prefers sportive activities, the upper part of the square should be removed.

Each iteration of the value iteration method must compare two vectors of $\bar{\mathcal{V}}$, say $\bar{V}^{\pi_i}$, and $\bar{V}^{\pi_j}$. From the user point of view, the comparison amounts to deciding "which of $\bar{\lambda}^* \cdot \bar{V}^{\pi_i}$ or $\bar{\lambda}^* \cdot \bar{V}^{\pi_j}$ is the highest?". In other words, the previous question is equivalent to deciding the sign of $\bar{\lambda}^* \cdot (\bar{V}^{\pi_i} - \bar{V}^{\pi_j})$ quantity. But $\bar{\lambda}^*$ is unknown, so we are going to approximate its numerical value inside the Λ polytope and define the equation sign in the same time.

In detail, we use three types of vector comparison methods in our algorithm, which are tested in the given order until any of them gives an answer. The first two methods provide an answer when the vectors can be compared relying on the restrictions of Λ as is. When they fail, the third comparison introduces a new cut on Λ polytope and refines our knowledge about user preferences [12].

1) *Pareto dominance* is a partial preference relation which is the least expensive method. It is defined as:

   Definition 3.1. For two given vectors $\bar{A} = (a_1, \cdots, a_d)$ and $\bar{B} = (b_1, \cdots, b_d)$ in $\mathbb{R}^d$ we have:

   $$\bar{A} \succcurlyeq_D \bar{B} \Leftrightarrow \forall i \;\; a_i \geq b_i$$

2) *KDominance* comparison succeeds when all the $\bar{\lambda}$ of Λ verify $\bar{\lambda} \cdot \bar{V}^{\pi_i} \geq \bar{\lambda} \cdot \bar{V}^{\pi_j}$ or when they all verify $\bar{\lambda} \cdot \bar{V}^{\pi_j} \geq \bar{\lambda} \cdot \bar{V}^{\pi_i}$. It is the second least expensive test. It can be formulated as a linear program [12]:

   $$\min_{\bar{\lambda} \in \Lambda} \bar{\lambda} \cdot (\bar{V}^{\pi_i} - \bar{V}^{\pi_j})$$

$\bar{V}^{\pi_i}$ is preferred to $\bar{V}^{\pi_j}$, if the above LP has a non-negative solution.

3) Ask the query to the user: "Is $\bar{\lambda} \cdot \bar{V}^{\pi_i} \succ \bar{\lambda} \cdot \bar{V}^{\pi_j}$?"

The final solution is the most expensive and less desired option, because our aim is finding the optimal solution with less interactions and interruptions with the user. In fact this last option devolves answering to the user.

For many vector comparisons, Pareto dominance cannot decide. As we are interested in reducing the number of queries proposed to the user, we aim at collecting more Kdominance comparable $\bar{V}$ pairs. Since our algorithm is based on value iteration, we desire to generate Kdominance comparable $\bar{V}$ vectors in every iteration.

Let $\pi^{\hat{s}\uparrow\hat{a}}$ means: the policy which is the same as π, except it chooses the action $\hat{a}$ in state $\hat{s}$ instead of choosing the action $\pi(\hat{s})$:

$$\pi^{\hat{s}\uparrow\hat{a}}(s) = \begin{cases} \pi(s) & \text{if } s \neq \hat{s} \\ \hat{a} & \text{if } s = \hat{s} \end{cases}$$

According to standard policy improvement in [7], the following observation applies:

Observation 3.1. If $\bar{V}^{\pi}$ is not optimal for the value $\bar{\lambda} \in \Lambda$, there exist a state $\hat{s}$ and an action $\hat{a}$ s.t.: $\bar{\lambda} \cdot \bar{V}^{\pi^{\hat{s}\uparrow\hat{a}}} \geq \bar{\lambda} \cdot \bar{V}^{\pi}$.

Since π and $\pi^{\hat{s}\uparrow\hat{a}}$ policies only differ in the $\hat{s}$ state, the following inequality results from observation 3.1:

$$\bar{\lambda} \cdot \bar{Q}^{\pi}(\hat{s}, \pi^{\hat{s}\uparrow\hat{a}}(\hat{s})) = \bar{\lambda} \cdot \bar{V}^{\pi^{\hat{s}\uparrow\hat{a}}}(\hat{s}) \geq \bar{\lambda} \cdot \bar{V}^{\pi}(\hat{s}) \quad (6)$$

In the light of 6, we analyze the advantage of $\pi^{\hat{s}\uparrow\hat{a}}$ over π, *i.e.* the difference of vectors $\mathbb{E}_{s\sim\beta}[\bar{V}^{\pi^{\hat{s}\uparrow\hat{a}}}]$ and $\mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}]$. First,

$$\begin{aligned} \bar{A}_{\hat{s},\hat{a}} &= \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi^{\hat{s}\uparrow\hat{a}}}] - \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}] \\ &= \sum_s \beta(s)\bar{V}^{\pi^{\hat{s}\uparrow\hat{a}}}(s) - \sum_s \beta(s)\bar{V}^{\pi}(s) \end{aligned}$$

Since the only difference between π and $\pi^{\hat{s}\uparrow\hat{a}}$ is state $\hat{s}$, we have:

$$\bar{A}_{\hat{s},\hat{a}} = \beta(\hat{s})\bar{V}^{\pi^{\hat{s}\uparrow\hat{a}}}(\hat{s}) - \beta(\hat{s})\bar{V}^{\pi}(\hat{s})$$

Following Equation 6, we can write:

$$\bar{A}_{\hat{s},\hat{a}} = \beta(\hat{s})\{\bar{Q}^{\pi}(\hat{s}, \pi^{\hat{s}\uparrow\hat{a}}(\hat{s})) - \bar{V}^{\pi}(\hat{s})\}$$

and finally advantage of $(\hat{s}, \hat{a})$ pair is as below [2], [4]

$$\bar{A}_{\hat{s},\hat{a}} = \beta(\hat{s})\{\bar{Q}^{\pi}(\hat{s}, \hat{a}) - \bar{V}^{\pi}(\hat{s})\} \quad (7)$$

We can explore new policies with more than one advantage vector differences from π. Let $\pi'' = \pi^{\hat{s}_1\uparrow\hat{a}_1,\cdots,\hat{s}_k\uparrow\hat{a}_k}$ be the policy which only differs from π in the state-action pairs $\{(\hat{s}_1, \hat{a}_1), \cdots, (\hat{s}_k, \hat{a}_k)\}$. Assuming that each $\hat{a}_i$ is chosen such that $\bar{Q}^{\pi}(\hat{s}_i, \hat{a}_i) \geq \bar{V}^{\pi}(\hat{s}_i)$ and as value iteration modifies the actions in all the states before iterating on the value function, we have:

$$\bar{\lambda} \cdot \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi''}] \geq \bar{\lambda} \cdot \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi^{\hat{s}\uparrow\hat{a}1}}] \geq \bar{\lambda} \cdot \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}]$$

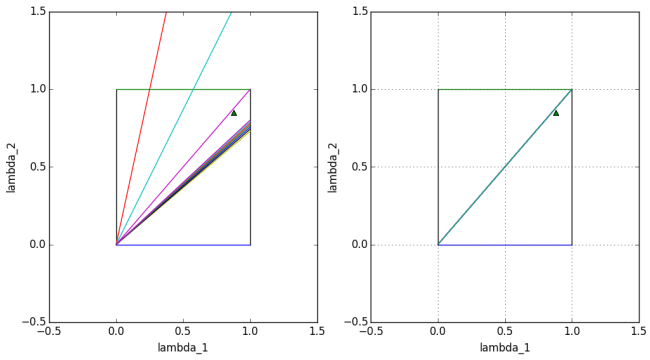


Figure 2: two types of cuts proposed by: using clustering methods(left figure) and without clustering(right figure) in 2 dimensional weight space.

The change in $\mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}]$ can be analyzed more precisely:

$$\begin{aligned} &\mathbb{E}_{s\sim\beta}[\bar{V}^{\pi''}] - \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}] \\ &\quad = \mathbb{E}_{s\sim\beta}[\bar{Q}^{\pi}(s, \pi''(s))] - \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}] \\ &\quad = \sum_{i=1}^{k} \beta(\hat{s}_i)\{\bar{Q}^{\pi}(\hat{s}_i, \pi''(\hat{s}_i)) - \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}](\hat{s}_i)\} \\ &\quad = \sum_{i=1}^{k} \bar{A}_{\hat{s}_i,\pi''(\hat{s}_i)} \end{aligned}$$

meaning that several vectorial advantages are additive. In particular, we can say:

$$\text{if } \bar{\lambda}\cdot\mathbb{E}_{s\sim\beta}[\bar{V}^{\pi''}] - \bar{\lambda}\cdot\mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}] \geq 0 \text{ then } \bar{\lambda}\cdot\sum_{i=1}^{k} \bar{A}_{\hat{s}_i,\pi''(\hat{s}_i)} \geq 0$$

In order to use less comparisons and collect more pointwise non-dominated policies, we try to explore a set of well distributed policies in $\bar{\mathcal{V}}$. Therefore, for an arbitrarily selected policy π, we are interested in comparing it with alternative policies with large values of $\bar{\lambda} \cdot \sum \bar{A}_{s,a}$. Based on this objective, we concentrate on the advantages set $\mathcal{A} = \{\bar{A}_{s,a} | s \in S, a \in A\}$ and their characterizations.

4. Advantage-Based Value Iteration

This section proposes a new algorithm named *Advantage-Based Value Iteration (ABVI)* to find an approximation of optimal policy for a given IRMDP (see Algorithm 2). ABVI is inspired by *interactive value iteration (IVI)* [12], which is copied in this paper as Algorithm 1. The following notations are used: $\mathcal{K}$ denotes a set of constraints defining the polytope Λ; the *getBest* function compares $\bar{V}_{\text{best}}$ and $\bar{Q}(s, a)$ according to the three methods listed in section 3 and modifies $\mathcal{K}$ if the method involves a new constraint on Λ. *getBest*, is exactly the same function as in algorithm 1 [12].

In Algorithm 1, each state s generates $|A|$ different increments of the form $\frac{\bar{A}_{s,a}}{\beta(s)} = \bar{Q}(s, a) - \bar{V}_t(s)$ in every iteration. Thus, for each state s, $\bar{V}_t(s)$ is compared with

Algorithm 1 Interactive Value Iteration

Inputs: MDP($S, A, p, \bar{r}, \gamma, \beta$), Λ, ϵ
Outputs: optimal $\bar{V}^*(s)$ for each s

$t \longleftarrow 0$
$\forall s\ \bar{V}_0(s) \longleftarrow (0, .., 0)$: zero vector of d dimension
$\mathcal{K} \longleftarrow$ set of constraints on Λ
repeat
  $t \longleftarrow t + 1$
  for each s **do**
    $V_{\text{best}} \longleftarrow (0, .., 0)$
    for each a **do**
      $\bar{Q}(s, a) \leftarrow \bar{r}(s, a) + \gamma \sum_{s'} p(s'|s, a)\bar{V}_{t-1}(s')$
      $(V_{\text{best}}, \mathcal{K}) \longleftarrow$ **getBest**$(V_{\text{best}}, \bar{Q}(s, a), \mathcal{K})$
    end for
    $\bar{V}_t(s) \leftarrow V_{\text{best}}$
  end for
until $||\mathbb{E}_{s\sim\beta}[\bar{V}_t] - \mathbb{E}_{s\sim\beta}[\bar{V}_{t-1}]|| < \epsilon$
Return $\bar{V}_t$

the set of new vectors $\{\bar{V}_t(s) + \frac{\bar{A}_{s,a_1}}{\beta(s)}, \ldots, \bar{V}_t(s) + \frac{\bar{A}_{s,a_m}}{\beta(s)}\}$ (where $A = \{a_1, .., a_m\}$ is the set of actions for the given VVMDP).

Example 4.1. Referring to Example 2.1, suppose each state has the initial value $(0, 0)$: $\bar{V}$(hometown) $= \bar{V}$(beach) $= (0, 0)$. Using interactive value iteration, in every state the vectorial value should be compared with each updated vector in each iteration. For example in state "home town", $(0, 0)$ vector is compared to 4 new vectors $\{(0, 0) + (0, 1), (0, 0) + (0, 0), (0, 0) + (1, 0), (0, 0) + (0, 0)\}$ which are related respectively to "going to exhibition","move", "biking" and "wait". It will be the same in state "beach" regarding reward vectors $\{(0, 0) + (0, 1), (0, 0) + (0, 0), (0, 0) + (1, 0), (0, 0) + (0, 0)\}$.

In order to accelerate value iteration and reduce its complexity, we do clustering on the advantages set $\mathcal{A}$ at each iteration. Aggregation of advantages based on their classification is our basic idea to accelerate the value iteration method. ABVI also takes initial state distributions into account to assign a single advantage vector to all MDP states. Instead of comparing $\bar{V}_t(s)$ with $\bar{V}_t(s) + \frac{\bar{A}_{s,a}}{\beta(s)}$ for each s, a, we try to produce a greater vector to add to global value vector for each iteration.

Example 4.2. Continuing Example 4.1, if the initial distribution on states is β(hometown, beach) $= (0.6, 0.4)$, initial vector values will be β(hometown)$\bar{V}$(hometown) + β(beach)$\bar{V}$(beach) $= (0, 0)$. Thus, this time $(0, 0)$ should be compared to $\{0.6(0, 1),\ 0.6(0, 0),\ 0.6(1, 0),\ 0.6(0, 0),\ 0.4(0, 1),\ 0.4(0, 0),\ 0.4(1, 0),\ 0.4(0, 0)\}$ as a set of all possible advantages for all (state, action) pairs of our MDP example. If $0.6(1, 0)$ and $0.4(0, 1)$ are new selected advantages, the next vector-valued function will be $(0, 0) + 0.6(1, 0) + 0.4(0, 1) = (0.6, 0.4)$ which represents policy π as π(hometwon) = biking and π(beach) = reading book.

Algorithm 2 Value Iteration Algorithm with Advantages

Inputs: MDP($S, A, p, \bar{r}, \gamma, \beta$), Λ, ϵ
Outputs: optimal policy π_{best}

$t \longleftarrow 0$
$\pi_{\text{best}} \longleftarrow$ choose random policy
$\forall s\ \bar{V}_0(s) \longleftarrow (0, .., 0)$: zero vector of d dimension
$\mathcal{K} \longleftarrow$ set of constraints on Λ
repeat
  $A_{\text{adv}} \longleftarrow \emptyset$
  for each s, a **do**
    $\bar{Q}_t(s, a) \longleftarrow \bar{r}(s, a) + \gamma \sum_{s'} p(s'|s, a)\bar{V}_t(s')$
    $A_{s,a} \longleftarrow \beta(s)\{\bar{Q}_t(s, a) - \bar{V}_t(s)\}$
    $A_{\text{adv}} \longleftarrow$ Add $A_{s,a}$ to A_{adv}
  end for
  $S_\Pi \longleftarrow$ **Cluster-Advantages**$(A_{\text{adv}}, \bar{V}_t, \pi_{\text{best}})$
  for $(\pi_c,$ **value**$(\pi_c)) \in S_\Pi$ **do**
    $(\pi_{\text{best}}, \mathcal{K}) \leftarrow$ **getBest**(**value**(π_c), **value**$(\pi_{\text{best}}), \mathcal{K})$
  end for
  for each s **do**
    $\bar{V}_{t+1}(s) = \bar{r}(s, \pi_{\text{best}}(s)) + \gamma \sum_{s'}\ p(s'|s, \pi_{\text{best}}(s))\bar{V}_t(s')$
  end for
  $t \longleftarrow t + 1$
until $||\bar{V}_{t-1} -$ **value**$(\pi_{\text{best}})|| \leq \epsilon$

Regarding to Section 3, each comparison between vectors $\bar{V}_i$ and $\bar{V}_j$ requiring a user query results in a cut on Λ polytope. The cuts generated by clustering on advantages give more information on weight polytopes because they generate pairs of vectors more different from each other than the classical value iteration. For instance, Figure 2 illustrates how clustering proposes various range of cuts on polytope in comparison with value iteration approach.

Algorithm 2 implements these ideas. It receives a VVMDP and a d-dimensional unit cube Λ as inputs. In this algorithm we start with a random policy $\pi_{\text{best}} \in \Pi$ and as IVI algorithm we suppose $\forall s \in S,\ \bar{V}_0(s) = \mathbf{0}$.

In each iteration, the algorithm generates $|S||A|$ advantages and keeps them in the A_{adv} set. Then the *Cluster-Advantages* function gets the advantages in A_{adv} and classifies them. The classification criterium is discussed at the end of this section. In the following, C denotes the set of clusters generated from A_{adv}.

For each cluster $c \in C$, the algorithm constructs the equivalent vector-valued function and its related policy. **CH-vertices**(c) denoting the set of vertices of the convex hull built on the cluster c, the new policy π_c is defined as:

$$\pi_c(s) = \begin{cases} a & \text{such that } A_{s,a} \in \textbf{CH-vertices}(c) \\ \pi_{\text{best}}(s) & \text{if no such } a \text{ exist.} \end{cases}$$

If there are several $A_{s,a} \in$ **CH-vertices**(c) for the same s, the algorithm randomly selects one of them, say $A_{s,a'}$, and assigns action a' to $\pi_c(s)$. To summarize, each cluster

c produces a (policy, vector-valued function) pair which is:

$$(\pi_c, \overbrace{\sum_s \beta(s)\bar{V}_t(s) + \sum_{A \in \textbf{CH-vertices}(c)} A}^{\textbf{value}(\pi_c)})$$

After having stored all new pairs $(\pi_c, \textbf{value}(\pi_c))$ in the set S_Π, the algorithm is ready to compare the newly explored policies with the best policy from the previous iteration. **value**(π) is the vectorial value function of dimension d for policy π.

Assume *getBest* receives two vectors $\textbf{value}(\pi_i) = \bar{V}_i$ and $\textbf{value}(\pi_j) = \bar{V}_j$. If Pareto dominance and Kdominance could not decide for vectors comparison, a query of form "is $\bar{V}_i$ preferred to $\bar{V}_j$?" will be proposed to the tutor. User's reply introduces a new cut on the Λ polytope as $\bar{\lambda} \cdot (\bar{V}_i - \bar{V}_j) \geq 0$ or $\bar{\lambda} \cdot (\bar{V}_i - \bar{V}_j) \leq 0$ and upgrades our knowledge about user's preferences by pruning part of Λ polytope. Note also that colinear advantages induce the same cut.

$\bar{V}_t$ matrix is updated according to the best policy π_{best} at the end of each iteration (lines 16 and 17). This iteration continues until vector-valued functions converge.

To do classification on advantages, we use hierarchical clustering with *CosineSimilarity* norm. Selecting a convenient norm is important, because each norm yields different clustering results on our data set. The CosineSimilarity norm is based on colinearity, so we get an equivalent vector for each cluster by summing its members. Allocating a threshold for clustering will define the final number of classified vectors. Suppose a threshold of δ, meaning that clusters satisfy $\forall c \in C,\ \forall A_i, A_j \in c\ d_{\text{cosine-similarity}}(A_i, A_j) \leq \delta$. This threshold defines a maximum angle between any two vectors of each cluster. For instance in Figure 3, 2-dimensional vectors have been classified. Each different group of points represents a cluster, while each vector is the equivalent of the corresponding cluster: the equivalent vector is the sum of vectors belonging to the cluster. The number of clusters is determined by the selected threshold δ. Larger δ gives fewer clusters and hence worse approximations of optimal policies at the end.

Example 4.3. Continuing Example 4.1, classifying these vectors will reduce comparisons among vectors. As an example, if we classify vectors in two sets as $\{(0, 0.6), (0, 0.4)\}$ and $\{(0.6, 0), (0.4, 0)\}$ by ignoring zero vectors, the initial vector $(0, 0)$ should be compared to each sum of set members i.e. $(0, 1.0) + (0, 0)$ and $(1.0, 0) + (0, 0)$. Thus, there are 2 comparisons among vectors for all states, while classical value iteration should check 8 comparisons for all states.

In Algorithm 2, there are $|S_\Pi|$ comparisons in each iteration (see line 13. $|S_\Pi|$ is the number of clusters on advantages). In comparison, Algorithm 1 uses $|S||A|$ comparisons. It means that the presented algorithm has less Kdominance comparisons and finally has to solve a smaller number of Linear Programming problems in any iteration.

As an example, Figure 3 shows the generated advantages for a MDP with 30 sates, 5 actions and $d = 2$, in one iteration of the value iteration algorithm. Each single point represents a vector that is assigned to $A_{s,a}$. Clustering on advantages returns 13 clusters. This allows us to compare $\bar{V}_t$ with 13 vectors instead of comparing it with 150 vectors for the IVI algorithm.

5. Related Work

At the best of our knowledge, existing algorithms for MDPs with uncertain rewards address three principal approaches. One has been presented in *Inverse Reinforcement Learning*, which infers rewards by observing an expert user behavior [6]. It means IRL algorithms builds reward functions based on given optimal policies. An IRL approach closer to our contribution finds an optimal policy through some reward functions such that its feature expectations match the expert user feature expectations [1]. Feature expectations are analog to vector-valued functions of our method while the goal is to find reward weights.

Another approach is using an iterative algorithm to find an optimal policy such as value iteration or policy iteration [12]. To implement this idea they assume that available rewards for any (state, action) pair are ordered [11] and transform IRMDPs to VVMPs. They also discuss how to generate informative queries to refine algorithm knowledge on unknown rewards. Our work uses the same structure on rewards, except it does not suppose an order on available rewards and implements value iteration method on VVMDP in interaction with expert user.

Another approach is using optimization methods based on *minimax regret* method [8], [9], [10]. The authors show how to find optimal policy by optimizing minimax regret while set of all candidate value functions are computable. They generate queries using the same criteria. Method drawbacks are NP-Complexty and not being applicable on large MDPs. Among these paper series, the only approach which addresses some experimental results on large MDPs is Regan and Boutilier work [10]. They improved quality of approximation by exploring the set of nondominated policies. The importance of their presented method is that the set of non-dominated policies is improved using online adding/pruning algorithm that adjusts during reward elicitation. Since this algorithm is online, if this exploration algorithm continues for longer time, the algorithm will find more non-dominated policies. As a consequence, the calculated optimal policy from the minimax regret method will be more accurate on more non-dominated policies. As there are too many non-dominated policies for each MDP even with few states, they have used a binary structure on states for doing experiments. For instance, state $s =< x_1, x_2, .., x_7 >$ is composed of 7 binary variables which yield a MDP with $|S| = 128$. That indicates that even this new forms of non-dominated policy search demands huge source of memory and calculation time such that it is not applicable on large MDPs without any structure.

ABVI algorithm includes less linear programming problems and is less complex in comparison to the presented minimax regret approaches. This algorithm converges faster

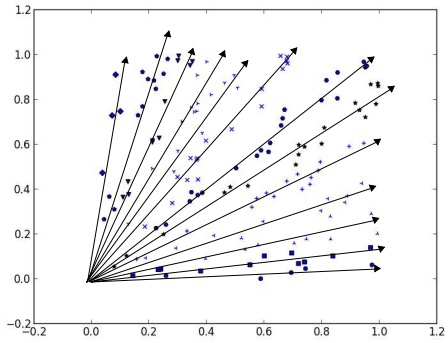
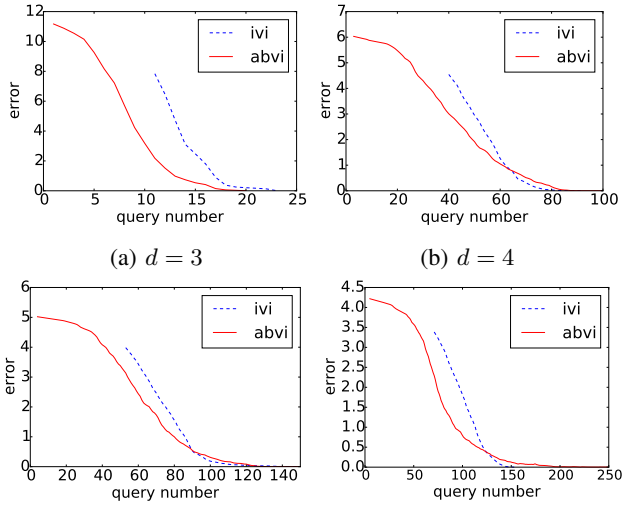


Figure 3: Clustering on advantages for a given VVMDP with 2-dimensional $\bar{\lambda}$s. Each different group points constitutes a cluster and each vector is equivalent to the sum of $\bar{\lambda}$s in the corresponding cluster.

to the optimal policy for not too high precision, and is applicable on larger MDPs [12]. Some results on classical MDPs have been presented in Section 6.

6. Experimental Results

We have examined our results on random MDPs when ϵ error in Algortithm 2 is equal 10^{-4}. All experiments have been implemented with Python version 2.7. We have used "CPLEX" solver to solve linear programming problems and *scipy.cluster* package to do hierarchical clustering on advantages. Results have been averaged on 10 iterations.



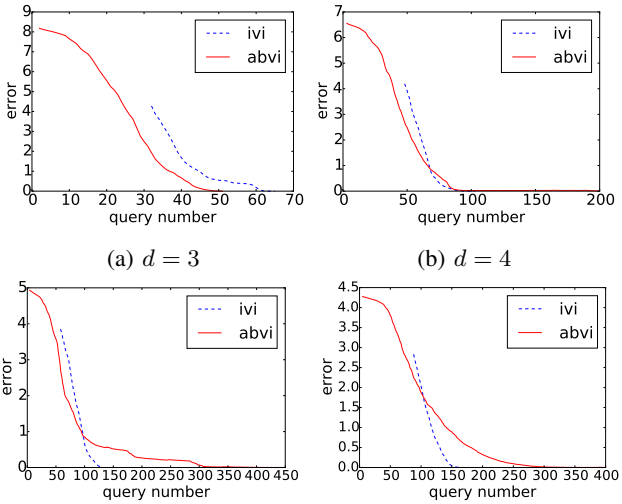
(a) $d = 3$ (b) $d = 4$

(c) $d = 5$ (d) $d = 6$

Figure 4: These graphs illustrate errors vs the number of queries answered by user where $|S| = 128$ and $d = 3, 4, 5, 6$

6.1. Random MDPs

A random MDP is defined by several parameters including its number of states n, its number of actions m and its



(a) $d = 3$ (b) $d = 4$

(c) $d = 5$ (d) $d = 6$

Figure 5: These graphs illustrate errors vs the number of queries answered by user where $|S| = 256$ and $d = 3, 4, 5, 6$

weight space dimension d. All rewards are bounded between 0 and 1. Transition function has several properties: from any state s restrict transitions to reach $\lceil \log_2(n) \rceil$ states. For each pair of (s, a) draw reachable states based on uniform distribution over the set of states. For drawn states, transition probabilities are formed based on Gaussian distribution. The initial state distribution β is uniform and we choose discount factor $\gamma = 0.95$. Remind that weight polytope Λ is initialized as a unite d-dimensional hyper-cube.

If $\bar{\lambda}$ vector is known numerically regarding to the special user priorities, the exact vector-valued function is obtainable. The exact vector is assigned as $\bar{V}_{\text{exact}}$. Thus, demonstrated error in Figures 6 and 5 is defined as the difference between the exact vector and the vector obtained at the end of each iteration for the methods IVI and ABVI. For instance, if $\textbf{value}(\pi^*) = \bar{V}_{\text{exact}}$ is the exact answer, and $\textbf{value}(\pi') = \bar{V}_t$ is the obtained result of t-th iteration for a given VVMDP:

$$\text{error} = ||\bar{V}_t - \bar{V}_{\text{exact}}||_\infty$$

Our experiments have been done on a random MDP with 5 actions and two various number of states 128 and 256. In Figure 6, the number of states is $n = 128$ and the error is compared to the number of asked queries for $d = 3, 4, 5, 6$. The figure displays that errors in ABVI algorithm are always less than IVI algorithm for small d-dimensions. For example when $d = 3$, after proposing 15 queries to user, ABVI error falls down to 0.53, while IVI algorithm still has an error around 2.5. For higher d dimensions such as 4, 5 and 6, error in ABVI reduces quicker than in IVI , except to gain the optimal solution with error 10^{-4}, ABVI asks more comparison questions to the user. The reason is that, clustering on advantages has less importance at the end of value iteration. For higher d dimensions, if we look for an

number of states	10	20	30	40	50
IVI	35.8	56.1	55.2	68.6	77.9
ABVI	58	85.5	85	81	104.9

TABLE 1: number of states vs number of queries for IVI and ABVI

d dimension	2	4	6	8	10	
IVI	16.1	40.3	74.4	68.6	113.9	
ABVI	13.9	55.7	108.9	81	193.7	

TABLE 2: d dimension vs. number of queries for IVI and ABVI

optimal policy with less precision, ABVI will converge to the optimal solution by asking less number of queries. Similarly, Figure 5 compares number of required queries with error changing for both approaches where $|S| = 256$ and $d = 3, 4, 5, 6$. It has a similar behavior as the previous figure in error changing vs. number of queries. Considering this figure if we look for an answer with less accuracy, the process will finish with less queries using ABVI method. For example for $d = 4$ and 5, our method has the same precision with less queries than IVI until an error around 1.0. These results indicate that ABVI method is less reliable than IVI method for MDPs with more states. We believe it can be improved by adjusting the clustering parameters dynamically.

In the rest of our experimental results, we have counted the number of user queries with respect to $|S|$ and to d running until reaching a final error less than 10^{-4}. In this part every execution is averaged on 10 times repetition. Table 1 demonstrates how the number of proposed questions changes in regards of state changes when $d = 5$ and $m = 5$. Table 2 shows the number of queries when d-dimension changes (for $m = 5$ and $n = 25$). In both tables ABVI proposes more queries to tutor than IVI approach. Based on two previous presented figures, majority of these queries are asked for reducing error from a pretty good precision to the goal precision equal 10^{-4}.

7. Conclusions and Future Works

We have introduced an approach for exploring the optimal policy for a MDP in which rewards come from a set of weights. Weights are unknown numerically and are normalized between 0 and 1. We calculate optimal vector-valued functions using the Value Iteration method by interacting with user in required cases, inspired by vectorial form of value iteration method. Our approach computes advantages in any iteration and tries to reduce the number of vector comparisons by classifying and grouping advantages. We have indicated how classification technique accelerates our iteration method and converges faster to the optimal vector-valued solution.

Our next goal is finding a way to generate a set of optimal vector-valued functions totally independent of weights knowledge. After, we will utilize weight polytope to extract the best optimal value-vectored among from approximated set according to user preferences. That will be the only part where algorithm interacts with user.

References

[1] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st international conference on machine learning*, 2004.

[2] L. C. Baird. Advantage updating. *Technical report. WL-TR-93-1146, Wright-Patterson Air Force Base*, 1993.

[3] E. Delage and S. Mannor. Percentile optimization in uncertain markov decision processes with application to efficient exploration. *In Proceedings of the 24th International Conference on Machine Learning, ICML 07*, pages 225–232, 2007. New York, NY, USA.

[4] S. Kakade. *On the sample complexity of reinforcement learning*. PhD thesis, Gatsby computational Neuroscience Unit. University College London, London, UK, 2003.

[5] H. Macmahan, G. Gordon, and A. Blum. Planning in the presence of cost functions controlled by an adversary. *ICML-03*, pages 536–543, 2003.

[6] A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann, 2000.

[7] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. Wiley, 1994.

[8] K. Regan and C. Boutilier. Regret-based reward elicitation for markov decision processes. *UAI-09 The 25th Conference on Uncertainty in Artificial Intelligence*, 2009.

[9] K. Regan and C. Boutilier. Robust policy computation in reward-uncertain mdps using nondominated policies. *Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, 2010.

[10] K. Regan and C. Boutilier. Robust online optimization of reward-uncertain mdps. *Twenty-Second Joint Conference on Artificial Intelligence (IJCAI 2011)*, 2011.

[11] P. Weng. Markov decision processes with ordinal rewards: Reference point-based preferences. *International Conference on Automated Planning and Scheduling*, 21:282–289, 2011.

[12] P. Weng and B. Zanuttini. Interactive value iteration for markov decision processes with unknown rewards. *IJCAI*, 2013.

[13] H. Xu and S. Mannor. Parametric regret in uncertain markov decision processes. *48th IEEE Conference on Decision and Control*, pages 3606–3613, 2009.

The author has requested enhancement of the downloaded file. All in-text references underlined in blue are linked to publications on ResearchGate.