STAIRS 2016
D. Pearce and H.S. Pinto (Eds.)

doi:10.3233/978-1-61499-682-8-15

Solving MDPs with Unknown Rewards Using Nondominated Vector-Valued Functions

Pegah ALIZADEH, Yann CHEVALEYRE and François LÉVY
LIPN, UMR CNRS 7030 - Institut Galilée, Université Paris 13
firstname.lastname@lipn.univ-paris13.fr

Abstract. This paper addresses vectorial form of *Markov Decision Processes* (MDPs) to solve MDPs with unknown rewards. Our method to find optimal strategies is based on reducing the computation to the determination of two separate polytopes. The first one is the set of admissible vector-valued functions and the second is the set of admissible weight vectors. Unknown weight vectors are discovered according to an agent with a set of preferences. Contrary to most existing algorithms for reward-uncertain MDPs, our approach does not require interactions with user during optimal policies generation. Instead, we use a variant of approximate value iteration on vectorial value MDPs based on classifying advantages, that allows us to approximate the set of non-dominated policies regardless of user preferences. Since any agent's optimal policy comes from this set, we propose an algorithm for discovering in this set an approximated optimal policy according to user priorities while narrowing interactively the weight polytope.

Keywords. Reward-Uncertain MDPs, Policy Iteration, non Dominated Vector-Valued Functions, Advantages, Reward Elicitation

1. Introduction

Markov decision process (MDP) is a model for solving sequential decision problems. In this model an agent interacts with an unknown environment and aims at choosing the best policy, *i.e.* the one maximizing collected rewards (so called its value). Once the problem modelized as a MDP with precise numerical parameters, classical *Reinforcement Learning (RL)* can prospect the optimal policy. But feature engineering and parameter definitions meets many difficulties.

Two main recent works deal with Incertain Reward MDPs (IRMDPs): the iterative methods [11, 1] and the minimax regret approach [6, 12]. Iterative methods provide a vectorial representation of rewards, the*Vector-Valued MDPs (VVMDP)*, such that determining unknown rewards reduces to determining the weight of each dimension. Standard Value iteration [9] is adapted to search the best value, and interaction is used when value vectors are not comparable using existing constraints on weights. Alizadeh *et al.* [1] use clustering on Advantages in this framework to take advantage of direction in the policies space. Maximum Regret is a measure of the quality of a given policy in presence of uncertainty, so Minimax Regret defines the best policy according to this criterion. When

the quality obtained is not good enough, proposed methods cut the space of admissible rewards with the help of an interactively obtained new bound for some reward, and then recompute Minimax Regret again.

Since Xu and Mannor [12] have shown the Minimax regret methods are computationally NP-hard, Regan and Boutilier explore the set of *nondominated policies* $\bar{\mathcal{V}}$ [8, 7]. They have developed two algorithms: one approximates $\bar{\mathcal{V}}$ offline and utilizes this approximation to compute MaxRegret [7]. In the other approach [8], they adjust the quality of approximated optimal policy using online generation of non-dominated policies.

This paper relies on value iteration method with clustering advantages to generate policies, but adopts the use of non dominated policies of [7] to speed up the computation of the one fitting the user preferences. As the VVMDP is independent of the user preferences, different users may use the same VVMDP and it makes sense to compute non dominated policies once only for different users. So two main algorithms are presented: the Propagation Algorithm (Algorithm 2) and the Search Algorithm (Algorithm 3). The former first receives a VVMDP, a set of constraints for unknown rewards and the precision ϵ for generating non dominated vectors. It explores the set of non-dominated vectors according to the given precision, taking advantage of Advantages clustering. The output $\mathcal{V}_\epsilon$ of Algorithm Propagation and the same set of constraints on reward weights are sent as inputs to the Search Algorithm. This algorithm finds the best optimal policy inside $\mathcal{V}_\epsilon$, interactively querying the user preferences to augment constraints on reward weights. The order of comparisons is dynamically chosen to reduce the number of queries.

Thus this paper has three main contributions: (a) For a given VVMDP, how to approximate the set of non-dominated vector-valued functions using clustering advantages [1]. (b) Search the optimal vector-valued functions satisfying user priorities using pairwise comparisons [3] and reward elicitation methods. (c) And finally we report some experimental results on MDPs with unknown rewards which indicate how our approach calculates the optimal policy with a good precision after asking few questions.

2. Vector-Valued MDPs

A finite horizon MDP is formally defined as a tuple $(S, A, p, r, \gamma, \beta)$ where S and A are respectively sets of finite states and finite actions, $p(s'|s,a)$ encodes the probability of moving 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. 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 solution of the equation: $\forall s,\ V^\pi(s) = r(s,\pi(s)) + \gamma \sum_{s' \in S} p(s'|s,\pi(s)) V^\pi(s')$. Taking into account the initial distribution β, each policy has an expected value function equal $v^\pi = \mathbb{E}_{s\sim\beta}[V^\pi(s)] = \sum_{s\in S} \beta(s) V^\pi(s)$ The optimal policy π^* is the policy with the greatest expectation : $\pi^* = \text{argmax}_\pi\ \beta \cdot v^\pi$. To compute the optimal policy and its expected value, the auxiliary Q function is defined as: $Q^\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^\pi(s')$. It allows to iterate the approximation:

$$\begin{aligned} \pi(s) &= \text{argmax}_a\ Q^\pi(s,a) \\ V^\pi(s) &= Q^\pi(s,\pi(s)) \end{aligned} \tag{1}$$

This schema leaves open in which states the policy is improved before computing next $V^\pi(s)$ and $Q^\pi(s,a)$, and what is the stop test. Different choices yield different algorithms, among which Value Iteration method (VI), which tests if the improvement of V^π is under a given threshold [4]. According to Equation 1, every $Q^\pi(s,a)$ has an improvement over $V^\pi(s)$ which is: $d(s,a) = Q^\pi(s,a) - V^\pi(s)$. This difference weighted by the initial distribution on the states is known as *Advantage* [2]

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

so Equation 1 can be modified as: $\pi(s) = \text{argmax}_a\ A(s,a)$

When designing real cases as MDPs, specifying the reward function is generally a hard problem. Preferences describe which (state, action) pairs are good or bad and should be interpreted into numerical costs. Note that even acquiring all these preferences is time consuming. Therefore, we use a *MDP with imprecise reward values (IRMDP)* and transform it into the vectorial formulation ([10, 11]). This paper handles the same structure on IRMDP with a slight difference: unknown rewards are not ordered contrary to the rewards in [11].

An IRMDP is a MDP(S, A, p,r, γ, β) with unknown or partially known rewards r , where unknown rewards need to be elicited. For that, let $E = \{\lambda_1, .., \lambda_{d-1}\}$ be a set of variables such 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 $r(s,a) \in E \cup \mathbb{R}$.

To match our IRMDP to a vectorial form, namely *Vector-Valued MDP (VVMDP)* [10], we define a vectorial reward function $\bar{r} : S \times A \longrightarrow \mathbb{R}^d$ 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 $r(s,a) = \sum_{i=1}^{d} \lambda_i.\bar{r}(s,a)[i]$, so any reward $r(s,a)$ is a dot product between two d-dimensional vectors:

$$r(s,a) = \bar{\lambda} \cdot \bar{r}(s,a). \tag{2}$$

Example 2.1 *Suppose the MDP in figure 3, with two states* {*hometown, beach*} *and six actions* {*swimming, reading book, going to exhibition, biking, wait, move*}. *Instead of having numerical rewards, it is known that every selected action in each state has only one of two qualities: "sportive" indicated by* λ_1 *and "artistic" indicated by* λ_2. *Unknown rewards are transformed to vectorial rewards such that their first element is the sportive value, while their second one shows the artistic value. For instance, the second element of* $\bar{r}$(*hometown, biking*) *is zero, because biking is not an artistic activity. If* $\bar{\lambda} = (\lambda_1, \lambda_2)$ *vector is known numerically, we will have* r(*hometown, biking*) $= 1\lambda_1 + 0\lambda_2$.

By leaving aside the $\bar{\lambda}$ vector, we have a MDP($S, A, p, \bar{r}, \gamma, \beta$) with vector-valued reward function representing the imprecise rewards. Basic techniques of MDP's can be applied componentwise. The discounted value function of policy π is a function $\bar{V}^\pi$:

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

$S \longrightarrow \mathbb{R}^d$ which provides in each state the discounted sum of reward vectors, and can be computed as

$$\bar{V}^{\pi}(s) = \bar{r}(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) \bar{V}^{\pi}(s') \tag{3}$$

In order to find the maximum vector in equations 1, 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 the Section 3.

Now, suppose that $\bar{V}^{\pi}$ is the discounted value function computed from equation 3. Taking into account the initial distribution, the vectorial expected value function is:

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

Hypothesizing a numerical weight value for the $\bar{\lambda}$ vector, and based on equations 2 and 3, a vectorial expected value function could be used to compute its corresponding scalar expected value function:

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

thus providing a comparison between any two policies. 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 reward 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: $\forall i = 1, \cdots, d \quad 0 \leq \lambda_i \leq 1$

3. Vector-Valued Functions Reduction

Our aim is now discovering the optimal scalar-valued function of the form: $\bar{\lambda}^* . \bar{v}^*$ with $\bar{v}^* \in \bar{\mathcal{V}}$ the optimal vectorial value function, and $\bar{\lambda}^* \in \Lambda$ the weight vector satisfying user preferences. In a context where different users have different preferences, it would make sense to compute $\bar{\mathcal{V}}$ set once and then searching interactively for each user, her preferred weight vector $\bar{\lambda}^*$ and the value function related to her optimal policy $\bar{v}^*$.

Since calculating whole $\bar{\mathcal{V}}$ polytope is not practically possible, we use an approximation $\bar{\mathcal{V}}_{\epsilon}$ with ϵ precision. It is built with the help of classification methods on advantages adapted to VVMDPs from classical Value Iteration [9]. The set $\bar{\mathcal{V}}_{\epsilon}$ is independent of user preferences, i.e. each $\bar{v} \in \bar{\mathcal{V}}_{\epsilon}$ approaches some optimal vector-valued functions regarding any particular $\bar{\lambda} \in \Lambda$.

The set of all non-dominated $\bar{v}$ vectors being approximately known, the next point is finding the optimal $\bar{v}^*$ according to user preferences. To compare between $\bar{v}$ vectors, our approach allows interaction with the user. Comparing the vectors $\bar{v}^{\pi_i}$, and $\bar{v}^{\pi_j}$ means

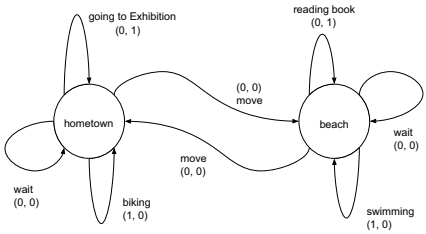

Figure 1. An example of small vector-valued MDP

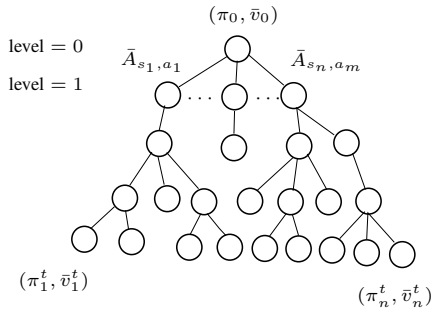

Figure 2. Structure tree of $\bar{\mathcal{V}}$ exploration.

deciding "which of $\bar{\lambda}^*.\bar{v}^{\pi_i}$ or $\bar{\lambda}^*.\bar{v}^{\pi_j}$ is the highest?". If the hyperplan $\bar{\lambda}^*.(\bar{v}^{\pi_i} - \bar{v}^{\pi_j}) = 0$ of the weight vectors space does not meet the Λ polytope, interaction is not needed. Else it defines a cut, and the user answer decides which part is kept. Through this process, the optimal $\bar{\lambda}^*$ value is approximated according to user preferences.

Example 3.1 *Referring to example 2.1, the Λ polytope is a unit square with axes representing sportive and artistic weights. $\lambda_1 = \lambda_2$ cuts the Λ polytope in two parts, and elicitation of user preferences between equal numbers of artistic and sportive privileges (the query "$\lambda_1 = \lambda_2$?") prunes one half of the polytope.*

In detail, we cascade three vector comparison methods. The first two provide an answer when the vectors can be compared relying on knowledge of Λ. If they fail, the third one introduces a new cut on Λ polytope and refines our knowledge about user preferences [11].

1. *Pareto dominance* is the cartesian product of orders and has the lowest computational cost: $(\bar{v}^{\pi_i} \succcurlyeq_D \bar{v}^{\pi_j}) \Leftrightarrow \forall i \;\; \bar{v}^{\pi_i}[i] \geq \bar{v}^{\pi_j}[i]$
2. *KDominance*: $\bar{v}^{\pi_i} \succcurlyeq_K \bar{v}^{\pi_j}$ holds when all the $\bar{\lambda}$ of Λ satisfy $\bar{\lambda} \cdot \bar{v}^{\pi_i} \geq \bar{\lambda} \cdot \bar{v}^{\pi_j}$, which is true if the following linear program has a non-negative solution [11]: $\min_{\bar{\lambda} \in \Lambda} \bar{\lambda} \cdot (\bar{v}^{\pi_i} - \bar{v}^{\pi_j})$
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 the less desired option, because it devolves answering to the agent. Our aim is finding the optimal solution with as few as possible interactions with the user .

Non-dominated vector definition and related explanations are given in [7]. A VVMDP with feasible set of weights Λ being given, $\bar{v} \in \bar{\mathcal{V}}$ is non-dominated (in $\bar{\mathcal{V}}$ w.r.t Λ) if and only if: $\exists \; \bar{\lambda} \in \Lambda \quad s.t. \quad \forall \; \bar{u} \in \bar{\mathcal{V}} \quad \bar{\lambda} \cdot \bar{v} \geq \bar{\lambda} \cdot \bar{u}$. It is obvious that $\bar{v}^*$ is a non-dominated vector-valued function, *i.e.* $\bar{v}^* \in \mathrm{ND}(\bar{\mathcal{V}})$. It means, we should find approximation of non-dominated vector-valued functions in order to select the optimal policy for any specific user.

Let $\pi^{\hat{s}\uparrow\hat{a}}$ notes the policy which differs from π, only in state $\hat{s}$, it chooses the action $\hat{a}$ instead of $\pi(\hat{s})$: $\pi^{\hat{s}\uparrow\hat{a}}(s) = \begin{cases} \pi(s) & \text{if } s \neq \hat{s} \\ \hat{a} & \text{if } s = \hat{s} \end{cases}$

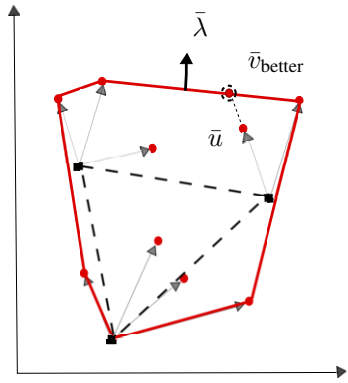

Figure 3. $\bar{\mathcal{V}}_{t+1}$ vectors selection after tree expansion of $\bar{\mathcal{V}}_t$

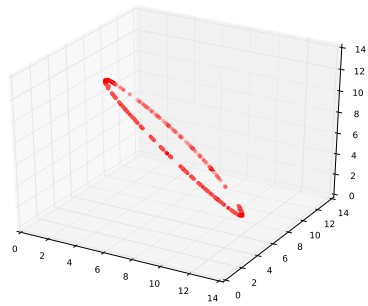

Figure 4. Generated non-dominated vectors from algorithm 2 for an MDP with 128 states, 5 actions and $d = 3$ (with $\epsilon = 0.01$)

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

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

We can explore new policies differing from π in more than one state. Let $\pi'' = \pi^{\hat{s}\uparrow\hat{a}1,\cdots,\hat{s}\uparrow\hat{a}k}$ be the policy which differs from π in the state-action pairs $\{(\hat{s}_1,\hat{a}_1),\cdots,(\hat{s}_k,\hat{a}_k)\}$. As each $\hat{a}_i$ is chosen such that $\bar{Q}^{\pi}(\hat{s}_i,\hat{a}_i) \geq \bar{V}^{\pi}(\hat{s}_i)$, we have:

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

In order to use less comparisons and collect more 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 $\bar{\lambda}\cdot\sum_{s\in S,a\in A}\bar{A}_{s,a}$. 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 [1].

In order to reduce exponential growth in $\bar{v}$s generation while keeping non-dominated vectors, our new algorithm suggests clustering advantages set $\{\bar{v}^{\pi}+\bar{A}_{s_1,a_1},\bar{v}^{\pi}+\bar{A}_{s_1,a_2},\cdots,\bar{v}^{\pi}+\bar{A}_{s_{|S|},a_{|A|}}\}$ in each iteration for the given vector $\bar{v}^{\pi}$ using cosine similarity metric [2] (Details are given in [1]).

After classifying advantages, suppose C is the set of resulting clusters. If a cluster c includes k advantages $\bar{A}_{s_{i_1},a_{i_1}},..,\bar{A}_{s_{i_k},a_{i_k}}$, the new policy π' is different from π in $(s_{i_1},a_{i_1}),..,(s_{i_k},a_{i_k})$ pairs and $\bar{v}^{\pi'} = \bar{v}^{\pi} + \sum_{j=1}^{k}\bar{A}_{s_{i_j},a_{i_j}}$ [3]

Classifying advantages in $|C|$ number of clusters, enables us to generate less $\bar{v}$ vectors and more useful (policy, vectored-value) pairs. Regarding Figure 2, the tree will have less and more effective nodes. The advantage of classification is that we have less improved policies with greater vector values.

[2] For each vector A, B cosine metric is $d_{\text{cosine}}(A,B) = 1 - \frac{A.B}{||A||||B||}$

[3] If there are several advantages with the same state $\hat{s}$, we choose one of them randomly

4. Approximate $\bar{\mathcal{V}}$ set with ϵ precision

Since extracting all members of $\bar{\mathcal{V}}$ is expensive, this section introduces an algorithm to generate an approximation of this set, namely $\bar{\mathcal{V}}_\epsilon$. This approximation at ϵ precision is a subset of $\bar{\mathcal{V}}$ such that $\forall \bar{v} \in \bar{\mathcal{V}}\ \exists \bar{v}' \in \bar{\mathcal{V}}_\epsilon\ s.t\ ||\bar{v}' - \bar{v}|| \leq \epsilon$.

Every vectorial value $\bar{v}$ is the value of at least one policy π. Any $\bar{v}$ of $\bar{\mathcal{V}}$ can be generated from a given user preference vector, using any initial pair $(\pi_0, \bar{v}_0)$, advantage clustering and value iteration method on VVMDPs . With slight abuse of notations we indicate each pair (π, $\bar{v}$) as a node n and utilize a graph to visualize our approach (cf. Figure 2). The approximation algorithm stores nodes and is initialized with a random policy $\pi_0 \in \Pi$ and a d-dimensional $\bar{v}_0$ vector equal to $\mathbf{0}$.

Essential to the algorithm is the function **expand**, described in Algorithm 1. It takes a node $n = (\pi, \bar{v})$ and returns back a set of new pairs (policy, vector-valued function). It computes first the full set $\mathcal{A}$ of advantages. This set has $|S||A|$ nodes of the form $(\pi_{s,a}, \bar{v}_{s,a})$: $\pi_{s,a}$ only differs from π in $\pi_{s,a}(s) = a$ and $\bar{v}_{s,a} = \bar{v} + \bar{A}_{s,a}$. Then the algorithm calls the **Cluster-Advantages** function which classifies $\mathcal{A}$, producing a set of clusters $C = \{c_1, .., c_k\}$. It returns the set of new nodes which expand the tree under node n and is defined from clusters:

$$N = \left\{ (\pi_j, \bar{v}_j) \;\middle|\; \begin{array}{l} \bar{v}_j = \bar{v} + \sum_{\bar{A}_{s,a} \in c_j} \bar{A}_{s,a} \\ \pi_j(s) = \begin{cases} a & \text{if } \bar{A}_{s,a} \in c_j \\ \pi(s) & \text{otherwise} \end{cases} \end{array} \right\}$$

Algorithm 1 expand : Expand Children for given node n

Inputs: node $n = (\pi, \bar{V})$ and VVMDP$(S, A, p, \bar{r}, \gamma, \beta)$
Outputs: N is set of n's children

1: $\mathcal{A} \longleftarrow \{\}$
2: **for each** s, a **do**
3: $\quad \mathcal{A} \longleftarrow$ Add $A_{s,a}$ to $\mathcal{A}$
4: $N \longleftarrow$ **Cluster-Advantages**$(\mathcal{A}, n)$
5: **return** N

Now with standing classification, adding all the nodes generated by **expand** remains exponential with respect to time and space, and all new nodes are not important either. Therefore, we look for an approach that avoids expanding the whole tree search and rolls up more non-dominated $\bar{v}$s of $\bar{\mathcal{V}}$. Suppose in t-th step of node expansion, we have n generated nodes in $N_t = (\pi_1^t, \bar{v}_1^t), \cdots, (\pi_n^t, \bar{v}_n^t)$ and name $\bar{\mathcal{V}}_t$ the second projection of N_t: $\bar{\mathcal{V}}_t = \{\bar{v}_1^t, .., \bar{v}_n^t\}$. The **expand** function on N_t will produce a new set of nodes. In fact, due to the convexity of $\bar{\mathcal{V}}$, vertices belonging to the convex hull of $\bar{\mathcal{V}}_t$ are enough to approximate $\bar{\mathcal{V}}$. This means that to build N_{t+1} we can compute the union of N_t and its expanded children, get their projections on their value coordinate, get the convex hull of this projection and prune nodes with $\bar{v}$ inside the convex hull.

For instance in Figure 3, square points result from t-th iteration and form the dashed convex hull. Red points are the expanded nodes. The polygon with straight lines represents the convex hull at $t + 1$-th iteration and $\bar{\mathcal{V}}_{t+1}$ involves the vertices of this polygon.

Algorithm 2 Propagation: Approximates $\bar{\mathcal{V}}_\epsilon$ set of non-dominated vectors for a given VVMDP

Inputs: VVMDP$(S, A, p, \bar{r}, \gamma, \beta)$, $\mathcal{K}$, ϵ
Outputs: $\bar{\mathcal{V}}_\epsilon$ set

$N \longleftarrow \{(\pi_0^0, \bar{v}_0^0), .., (\pi_d^0, \bar{v}_d^0)\}$
$\bar{\mathcal{V}}^{\text{new}} \longleftarrow$ **ConvexHull**(**getVectors**(N))
do
  $\bar{\mathcal{V}}^{\text{old}} \longleftarrow \bar{\mathcal{V}}^{\text{new}}$
  $N \longleftarrow \{\}$
  $\mathcal{C} \longleftarrow \{\}$
  for $n \in \bar{\mathcal{V}}^{\text{old}}$ **do**
    add **expand**(n) to $\mathcal{C}$
  for $n \in \mathcal{C}$ **do**
    if CheckImprove$(n, \bar{\mathcal{V}}^{\text{old}}, \mathcal{K}, \epsilon)$ **then**
      add n to N
  $\bar{\mathcal{V}}^{\text{new}} \longleftarrow$ **ConvexHull**$(\bar{\mathcal{V}}^{\text{old}} \cup$ **getVectors**$(N))$
while $\bar{\mathcal{V}}^{\text{new}} \neq \bar{\mathcal{V}}^{\text{old}}$
return $\bar{\mathcal{V}}^{\text{new}}$

Two remarks help seeing that this strategy does not prune optimal points. First the extrema of $\bar{\lambda} \cdot \bar{v}$ for $\bar{v}$ in a polytope P are on the vertices of P's convex hull. Second, the interior set of the $\bar{\mathcal{V}}_t$ polytope is increasing with t, so an interior point of $\bar{\mathcal{V}}_t$ cannot be a vertex of $\bar{\mathcal{V}}_{t+1}$

Using previous ideas and observations, we propose Algorithm 2 to generate an approximation of $\bar{\mathcal{V}}$. This algorithm uses four main functions: **ConvexHull**, **CheckImprove**, **expand** and **getVectors**. **ConvexHull** gets a set of d-dimensional vectors, completes it with the $\mathbf{0}$ vector and generates its convex hull. This function returns vertices of the convex hull except the $\mathbf{0}$ vector. Function **expand** has just been described (Algorithm 1). **getVectors** simply gets a set of nodes and returns back the set of their second elements (their value vectors).

Finally **CheckImprove** checks if a candidate node should be added to the current convex hull or not. It receives as arguments a candidate vector $\bar{v}$, an old set of selected vectors $\bar{\mathcal{V}}^{\text{old}}$ and the precision ϵ. Its return value is defined below:

$$\textit{CheckImprove}(\bar{v}, \bar{\mathcal{V}}^{\text{old}}, \mathcal{K}, \epsilon) = \quad \textbf{false if} \begin{cases} \bar{v} \in \textit{ConvexHull}(\bar{\mathcal{V}}^{\text{old}}) \\ \text{or } \exists\ \bar{u} \in \bar{\mathcal{V}}^{\text{old}}\ s.t\ ||\bar{v} - \bar{u}|| \leq \epsilon\,, \\ \text{or } \textbf{KDominates}(\bar{u}, \bar{v}, \mathcal{K}) \leq \epsilon \end{cases} \textbf{true o.w.}$$

If the candidate vector-valued function is inside the old convex hull or there is a vector in the old set that is ϵ-close to the candidate (in terms of euclidean or kdominate distance), this last will not be added.

Algorithm 2 receives a VVMDP, a stopping threshold ϵ and set of linear constraints of polytope Λ. The initial set of nodes N is generated by choosing the d unit vectors of $\mathbb{R}^d$[4]. This yields d scalar MDP's where classical value iteration method [9] can discover the optimal policy and related vectored-value function $\bar{v}_k^0$.

[4]It means $\bar{v}_0^k = (0, \cdots, 0, 1, 0, \cdots, 0)$ where 1 is the k-th element of the vector

Algorithm 3 Search Algorithm: Find Optimal $\bar{v}^*$ in $\bar{\mathcal{V}}_\epsilon$

Inputs: $\bar{\mathcal{V}}_\epsilon$ an approximation of non-dominated $\bar{v}$ vectors and $\mathcal{K}$ a constraints set defining Λ polytope
Outputs: $\bar{v}^*$

1: $T = P$
2: $D = U = \emptyset$
3: $\mathcal{K} = \{0 \leq x_i \leq 1 \; s.t. \; i \in [0, d]\}$
4: **for** (v_i, v_j) in T **do**
5: | **if** $v_i \succcurlyeq_D v_j$ **then**
6: | | **markDefined**(v_i, v_j)
7: | **else if** $v_i \preccurlyeq_D v_j$ **then**
8: | | **markDefined**(v_j, v_i)
9: **while** $T \neq \emptyset$ **do**
10: | **for** (v_i, v_j) in T **do**
11: | | $c_{i,j} = c_{j,i} = 0$
12: | | **repeat**
13: | | | choose random $\bar{\lambda} \in \Lambda(\mathcal{K})$
14: | | | **for** (v_i, v_j) in T **do**
15: | | | | **if** $\bar{\lambda} \cdot \bar{v}_i > \bar{\lambda} \cdot \bar{v}_j$ **then**
16: | | | | | $c_{i,j} += 1$
17: | | | | **else if** $\bar{\lambda} \cdot \bar{v}_j > \bar{\lambda} \cdot \bar{v}_i$ **then**
18: | | | | | $c_{j,i} += 1$
19: | | **until** 1000 times
20: | **for** (v_i, v_j) in T **do**
21: | | **if** $c_{j,i} = 0$ **then**
22: | | | **if** $v_i \succcurlyeq_K v_j$ **then**
23: | | | | **markDefined**(v_i, v_j)
24: | | **else if** $c_{i,j} = 0$ **then**
25: | | | **if** $v_i \preccurlyeq_K v_j$ **then**
26: | | | | **markDefined**(v_j, v_i)
27:
28: | $(v_i, v_j) = \text{Argmin}_{(v_i,v_j)\in T}(|c_{i,j} - 500|)$
29: | $(\mathcal{K}, ans) =$ **Query**$((\bar{v}_i, \bar{v}_j), \mathcal{K})$
30: | **if** $ans = i$ **then**
31: | | **markDefined**(v_i, v_j)
32: | **else**
33: | | **markDefined**(v_j, v_i)
34: **return FindBest**(D)

In each iteration, the algorithm generates all the children of any given $\bar{\mathcal{V}}^{\text{old}}$ member and makes a $\bar{\mathcal{V}}^{\text{new}}$ using *expand* and *CheckImprove* functions. The final solution will be an approximation of non dominated $\bar{v}$ vectors, and they assign to optimal policies for one or several $\bar{\lambda}$ vectors inside the Λ polytope. Figure 4 is an example of MDP with $d = 3$ and 128 states. It demonstrates how algorithm 2 generates many vectors for the MDP with average precision $\epsilon = 0.01$. This shows that this algorithm is useful for IRMDPs with average weight dimension d.

5. Searching Optimal V^* by Interaction with User

After discovering an approximated set of all possible optimal $\bar{v}$ vectors regardless of user preferences in Algorithm 2, we intend to find the optimal policy in the $\bar{\mathcal{V}}_\epsilon$ set with respect to the user preferences (The main goal of Algorithm 3). In this section we propose an approach to find the optimal $\bar{v}^* \in \bar{\mathcal{V}}_\epsilon$ and an approximation of vector $\bar{\lambda}^* \in \Lambda$ embedding user priorities. In fact by asking queries to the user (when the algorithm cannot decide), we approximate the maximum of $\bar{\lambda}^* \cdot \bar{v}^*$ for $\bar{v}^* \in \bar{\mathcal{V}}_\epsilon$ and the given user.

Note that the Λ polytope is initially a unit cube of dimension d and $\mathcal{K}$ is its set of constraints. As detailed in section 3, there are three types of comparisons to search the optimal $\bar{v}^*$ inside $\bar{\mathcal{V}}_\epsilon$. KDominance and Pareto comparisons are partial preferences, and when two vectors are not comparable by any of them, the final solution is delegating the comparison to the tutor. In Algorithm 3, $\succeq_D$ is the comparison regarding paretodominance or Kdominance comparisons, and $\succeq_K$ indicates just the Kdominance comparison.

The main characteristic of algorithm 3 is that it selects the best pair to test to obtain a maximal informative cut on Λ (line 13), in order to minimize the number of queries. In this algorithm, the label of a pair $(\bar{v}_i, \bar{v}_j)$ notes if vectors are comparable:

$$\text{label} = \begin{cases} 1 & \text{if } \bar{v}_i \succeq \bar{v}_j \\ -1 & \text{if } \bar{v}_i \prec \bar{v}_j \end{cases}$$

Following Jamieson and Nowak work [3] to rank objects using pairwise comparisons, the algorithm first generates the set P of vectors on $\bar{\mathcal{V}}$ set *i.e* $P = \{(v_i, v_j) \in \bar{\mathcal{V}}^2 | i < j\}$. T is the set of ambiguous pairs, which are comparable neither by Pareto nor by Kdominate comparisons. D is the set of compared pairs and U are the useless ones.

Functions **Query**, **markDefined** and **FindBest** are used within the algorithm and are explained in the rest of this section. The **Query** function receives a pair of vectors and set of constraints on the Λ polytope. This function proposes the query " $\bar{v_i}$ or $\bar{v_j}$" to the user and appends the new cut to $\mathcal{K}$ in accordance with user preferences while assigning a label to this pair. The **markDefined** function is as following:

$$\textbf{markDefined}(v_i, v_j) = \begin{cases} \text{move}((v_i, v_j), D) \\ \text{remove}(v_j, T) \end{cases}$$

where the function **remove**(a, l) removes all pairs containing a from the list l (T or D) and puts them into the set of useless pairs U.

The second part of the algorithm (Lines 9 to 28) chooses the best ambiguous pair from T set and proposes it as a query to the user. In fact, It looks for the cut that gets rid of more redundant points in Λ polytope. It uses a Monte-Carlo method, selecting 1000 points randomly inside Λ polytope, and choosing the cut that divides the random points into two almost equal sets.

The algorithm stops when the set of To be Determined pairs (T set) is empty, so all pairs are either determined (in D set) or useless (in U set). Knowing D, the **FindBest** algorithm computes all maximal points in linear time. It first initializes all c_i labels to 0, and then inside a loop it assigns $c_j = 1$ for all pairs $(v_i, v_j) \in D$. At the end, maximal points v_i's are those where c_i is equal 0. Finally, this algorithm gives the best vector-valued function with the highest rank at the end.

6. Empirical Evaluation

We have tested our Propagation-Search Value Iteration (PSVI) algorithm on small MDPs and compared it to IVI algorithm, an existing interactive value iteration method for exploring the optimal policy regarding agent preferences [11]. We have done the experiments for both methods on small, randomly generated IRMDPs. For each MDP with n states, m actions and weight vectors of dimension d, the transition functions restrict to reach $\lceil \log_2(n) \rceil$ states from each state. Initial state distribution β is uniform, the discount factor $\gamma = 0.95$ and each $r(s, a)$ is normalized between 0 and 1 (For more details see Random MDP [5]). Results in this section have been averaged on 5 random MDPs and 50 various users with different priorities.

Tables 1 and 2 compare two algorithms based on several measures on MDPs with 128 states, 5 actions and various d dimensions including 2, 3 and 4. The ϵ precisions are defined equal 0.2 and 0.1 for two presented tables of this section. $|\bar{\mathcal{V}}|$ is the number of generated non dominated vector-valued functions for each dimension using our Propagation algorithm (Algorithm 2) while the propagation time indicated in the tables is the time of accomplishing this process in seconds.

To evaluate the Search Algorithm (Algorithm 3) on obtained result of the Propagation algorithm, we try to explore the optimal policies related to 50 different users. Each user is displayed as a random $\bar{\lambda}$ inside Λ polytope and results have been averaged on

Table 1. Average Results on 5 iterations of mdp with $|S| = 128$, $|A| = 5$ and $d = 2, 3, 4$. The Propagation algorithm accuracy is $\epsilon = 0.2$. The results for Search algorithm are averaged on 50 random $\bar{\lambda} \in \Lambda$ (times are seconds).

Methods	parameters	$d = 2$	$d = 3$	$d = 4$
psvi	$\lvert\bar{\mathcal{V}}_\epsilon\rvert$	8.4	43.3	4.0
	Queries	**3.27**	**12.05**	**5.08**
	error	0.00613	0.338	0.54823
	propagation time	33.6377	170.1871	3.036455
	search time	1.20773	36.4435	6.022591
ivi	Queries	17.38	41.16	69.18
	error	**0.0058**	**0.319**	**0.5234**
	ivi time	9.8225060	5.57657	5.30287
psvi	total time	94.0242	10501.7171	304.166
ivi	total time	491.1253	278.8285	265.1435

Table 2. Average Results on 5 iterations of mdp with $|S| = 128$, $|A| = 5$ and $d = 2, 3, 4$. The Propagation algorithm accuracy is $\epsilon = 0.1$. The results for Search algorithm of optimal policy are averaged on 50 random $\bar{\lambda} \in \Lambda$ (times are seconds).

Methods	parameters	$d = 2$	$d = 3$	$d = 4$
psvi	$\lvert\bar{\mathcal{V}}_\epsilon\rvert$	7.79	154.4	32.2
	Queries	**2.52**	**15.99**	**15.7**
	error	0.0035	0.14914	0.519
	propagation time	57.530	3110.348	893.4045
	search time	0.8555	229.134	95.90481
ivi	Queries	17.8	42.15	67.79
	error	**0.0033**	**0.142**	**0.493**
	ivi time	10.0345	6.99638	5.309833
psvi	total time	100.305	14567.048	4795.2405
ivi	total time	501.725	349.819	265.49165

random selection of agents (here results have been averaged on 50). In Tables 1 and 2, the search time indicates the average time of policy search for 50 various users, and the error parameter for psvi method is defined as the difference between value of exact optimal policy $\bar{v}^{\pi^*_{\text{exact}}}$ and the optimal result of our approach $\bar{v}^{\pi^*}$ for a specific user $\bar{\lambda}$ as: $||\bar{\lambda}^T.\bar{v}^{\pi^*_{\text{exact}}} - \bar{\lambda}^T.\bar{v}^{\pi^*}||_\infty$.

These preliminary results indicate that though our algorithms take more time to produce all optimal policies list, it proposes considerably less questions to the user in comparison with IVI algorithm in order to find the optimal policy with the same accuracy. For instance regarding table 2, two algorithms find the optimal policy with an error around 0.1 after asking 16 and 42 queries respectively for psvi and ivi algorithms. The most striking advantage of our method is that it reduces around half number of queries by generating all possible optimal policies before starting any interaction with user.

In these results total time has a formulation for each approach regarding 50 tested users. Total time for IVI is simply 50×ivi time, and for the other method we have: 50×exploration time + propagation time. That means, after finding an approximated set of optimal policies for all type of users, our methods do not need recalculating non dominated policies anymore. On the other hand, the ivi algorithm should restart the algorithm from the beginning every time and it can discover only one optimal policy for one special

user. Psvi algorithm has a higher calculation time than ivi algorithm, because it takes a considerable amount of time on producing optimal policies (In Propagation Algorithm).

7. Conclusions and Future Works

We have introduced an approach for exploring the optimal policy for a MDP with uncertain rewards. We have presented a method for computing an approximate set of non-dominated policies and related vector-valued function in this case. We also showed that offline approximation of the set of non-dominated policies allows the optimal policy to be discovered while asking a considerably smaller number of questions to the agent. In future studies, we will study on possible solutions to reduce complexity of non-dominated vectors generation by removing more useless non-dominated vectors in any iteration. Another idea is implementing the other potential application of our method on Inverse Reinforcement Learning problems, and comparing it to standard algorithms in this domain

References

[1] Alizadeh, P., Chevaleyre, Y., and Levy, F. (2016). Advantage based value iteration for Markov decision processes with unknown rewards. *International Joint Conference on Neural Networks (IJCNN)*.

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

[3] Jamieson, K. G. and Nowak, R. D. (2011). Active ranking using pairwise comparisons. *CoRR*.

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

[5] Regan, K. and Boutilier, C. (2008). Regret-based reward elicitation for Markov decision processes. *NIPS-08 workshop on Model Uncertainty and Risk in Reinforcement Learning*, 1.

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

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

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

[9] Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An introduction*, volume 116. Cambridge University Press.

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

[11] Weng, P. and Zanuttini, B. (2013). Interactive value iteration for Markov decision processes with unknown rewards. *IJCAI*.

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