Solving MDPs with Unknown Rewards Using Non-Dominated Vector-Valued Functions

Pegah Alizadeh

LIPN, UMR CNRS 7030
Institut Galilée, Université Paris 13
pegah.alizadeh@lipn.univ-paris13.fr

Résumé : This paper addresses Vector-Valued MDPs (VVMDP) to solve *Markov Decision Processes* with unknown rewards. Our method to find optimal strategies is based on reducing VVMDPs computation to the determination of two separate polytopes. The first polytope is the set of admissible vector-valued functions and the second polytope is the set of admissible reward weight vectors. Unknown weight vectors are discovered with respect to a user owned 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 VVMDPs based on classifying advantages, that allows us to approximate the set of non-dominated policies regardless of user priorities. Since any user's optimal policy comes from this set, we propose an algorithm for discovering the approximated optimal policy according to user priorities in parallel with narrowing interactively the weight polytope.

Mots-clés : 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 is modeled 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 (Weng & Zanuttini, 2013; Alizadeh *et al.*, 2016) and the minimax regret approach (Regan & Boutilier, 2009; Xu & Mannor, 2009). 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 (Sutton & Barto, 1998) 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.* (2016) 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.

Xu & Mannor (2009) have shown the Minimax regret methods are computationally NP-hard, Regan and Boutilier explore the set of *nondominated policies* $\bar{\mathcal{V}}$ (Regan & Boutilier, 2011, 2010). They have developed two algorithms: one approximates $\bar{\mathcal{V}}$ offline and utilizes this approximation to compute MaxRegret (Regan & Boutilier, 2010). In the other approach (Regan & Boutilier, 2011), 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 (Regan & Boutilier, 2010) 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 (Alizadeh *et al.*, 2016). (b) Search the optimal vector-valued functions satisfying user priorities using pairwise comparisons (Jamieson & Nowak, 2011) and reward elicitation methods. And finally (c) report some experimental results on MDPs with unknown rewards indicating how our approach calculates the optimal policy with a good precision after asking a small number of queries.

2 Vector-Valued MDPs

An infinite horizon MDP is formally defined as a tuple $(S, A, p, r, \gamma, \beta)$ where S and A are respectively finite sets of states and 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 state s is a function $V^\pi : S \longrightarrow \mathbb{R}$ which is solution of the *Bellman Equation*:

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

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

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

Solving exactly a finite horizon MDP means finding an optimal policy π^* which is a policy maximizing the expectation: $\pi^* = \text{argmax}_\pi\ \beta \cdot v^\pi$.

To compute the optimal policy and its expected value, there is an auxiliary Q function that is defined as:

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^\pi(s') \tag{2}$$

It is shown below how to iterate on V and Q functions using a t index in order to approximate the optimal policy:

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

This schema leaves open in which states the policy is improved before computing next $V^{\pi_t}(s)$ and $Q^{\pi_t}(s, a)$, and what is the stopping criterion. Different choices yield different algorithms, among which the Value Iteration method (VI) tests if the improvement of V^π is below a given threshold (Puterman, 1994). According to Equation 3, 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 in known as *Advantage* (Baird, 1993). By taking initial distribution on the state into account, we have:

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

so Equation 3 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. For instance preferences stating which (state, action) pairs are good or bad 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 (Weng (2011); Weng & Zanuttini (2013)).

An IRMDP is an MDP(S, A, p,r$, \gamma, \beta$) with unknown or exactly known reward $r(s, a)$ for each state s and action a. If a reward value is known, it is required 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 $r(s, a) \in E \cup \mathbb{R}$. This means that the reward is inside $\mathbb{R}$, if its value is known ; otherwise it has a symbolic variable inside E that will be approximated in the future calculations. Thus, this paper handles the same structure as Weng & Zanuttini (2013)'s one on IRMDP with a slight difference: E members are not ordered contrary to the rewards in their structure.

To match our IRMDP to a vector form, namely *Vector-Valued MDP (VVMDP)* (Weng, 2011), we define a vector 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:

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

Example 2.1

Suppose a cab driver in a city containing 3 zones $\{a, b, c\}$. In each area, the driver can stay in the zone or move to another zone. But he has three choices for moving between the zones: Driving in highways, Excess of speed limit or exploring unfamiliar areas. Thus, the "navigation like Cabbie" problem (Ziebart et al., *2008; Bhattacharya & Das, 2002) is defined as an MDP given in Figure 1, with three states $\{$zone a, zone b, zone $c\}$ and four actions $\{$higways, excess speed, unfamiliar areas, move$\}$.*

Instead of having numerical rewards, it is known that every selected action in each state has only one of two qualities: "adventurous" indicated by λ_1 and "cautious" indicated by λ_2. Unknown rewards are transformed to vector rewards such that their first element is the adventurous value, while their second one shows the cautious value. For instance, the second element of $\bar{r}$(zone c, excess limit) is zero, because excess speed limit is not a cautious decision. If $\bar{\lambda} = (\lambda_1, \lambda_2)$ vector is known numerically, we will have r(zone c, excess limit) $= 1\lambda_1 + 0\lambda_2$.

By leaving aside the $\bar{\lambda}$ vector, we have an MDP$(S, A, p, \bar{r}, \gamma, \beta)$ with vector-valued reward function representing the imprecise rewards. 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

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

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

Now, suppose that $\bar{V}^\pi$ is the discounted value function computed from Equation 5. Taking into account the initial distribution, the expected vector value 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 4 and 5, a vector 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 an MDP with unknown rewards boils down to explore the interaction between two separate d-dimensional admissible

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

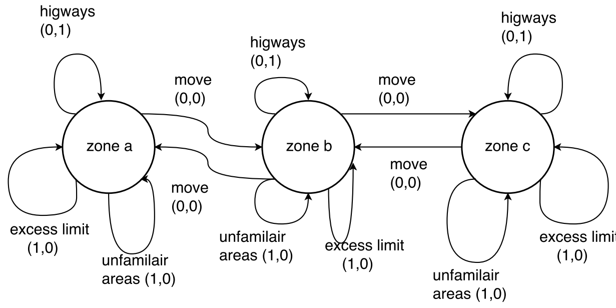


FIGURE 1: An example of small vector-valued MDP

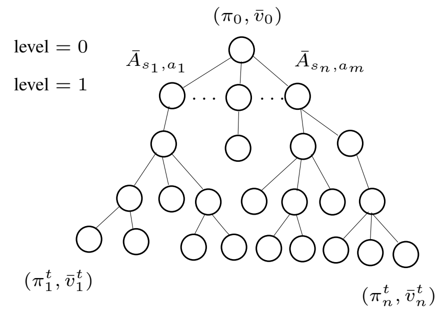


FIGURE 2: Structure tree of $\bar{\mathcal{V}}$ exploration.

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 vector 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 Reductions of Vector-Valued Functions

Our aim is now to discover an optimal scalar-valued function of the form: $\bar{\lambda}^*.\bar{v}^*$ with $\bar{v}^* \in \bar{\mathcal{V}}$ the optimal vector-valued function, and $\bar{\lambda}^* \in \Lambda$ the weight vector satisfying user preferences. In a context where many users will have different preferences, it would make sense to compute $\bar{\mathcal{V}}$ once before searching interactively for each user's preferred weight vector and value function.

Since the $\bar{\mathcal{V}}$ polytope is not calculable, we approximate a subset of non-dominated vectors $\bar{\mathcal{V}}_\epsilon$ with ϵ precision. It is built with the help of classification methods on advantages adapted to VVMDPs from classical Value Iteration (Sutton & Barto, 1998). 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 $\bar{\lambda} \in \Lambda$.

After approximately knowing the set of non-dominated $\bar{v}$ vectors, 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 deciding "which of $\bar{\lambda}^*.\bar{v}^{\pi_i}$ or $\bar{\lambda}^*.\bar{v}^{\pi_j}$ is the highest ?". If the hyperplane $\bar{\lambda}^*.(\bar{v}^{\pi_i} - \bar{v}^{\pi_j}) = 0$ of the weight vectors space does not meet the Λ polytope, interaction with the user is not required anymore. 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. Each interactive comparison integrates a new cut on polytope Λ and eliminates part of the polytope. Through weight vector elicitation, the optimal $\bar{\lambda}^*$ value is approximated according to user preferences.

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

In detail, we cascade three vector comparison methods to shrink the solution space and get more information on polytope Λ. The first two provide an answer when the vectors can be compared relying on knowledge of Λ. When this fails, the third one introduces a new cut on Λ polytope and refines our knowledge about user preferences (Weng & Zanuttini, 2013). Human answers to queries often allow more pruning from the dominance relations.

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 (Weng & Zanuttini, 2013):

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

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. We aim at finding the optimal solution with as few interactions with the user as possible.

Non-dominated vector definition and related explanations are given in Regan & Boutilier (2010), we repeat this definition here:

Definition 3.1
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: $\exists\, \bar{\lambda} \in \Lambda \;\; s.t. \;\; \forall\, \bar{u} \in \bar{\mathcal{V}} \;\; \bar{\lambda} \cdot \bar{v} \geq \bar{\lambda} \cdot \bar{u}$.

It is obvious that $\bar{v}^*$ is a non-dominated vector-valued function, *i.e.* $\bar{v}^* \in \mathrm{ND}(\bar{\mathcal{V}})$. This means that we should find approximation of non-dominated vector-valued function in order to select the optimal policy for any specific user.

Let $\pi^{\hat{s}\uparrow\hat{a}}$ denotes the policy which differs from π, only in state $\hat{s}$, by choosing the action $\hat{a}$ instead of $\pi(\hat{s})$:

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

Analyzing 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 $\mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}]$ is calculated:

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

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

$$\bar{A}_{\hat{s},\hat{a}} = \beta(\hat{s})\{\bar{Q}^{\pi}(\hat{s},\hat{a}) - \bar{V}^{\pi}(\hat{s})\} \tag{6}$$

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 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:

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

meaning that the vector advantages are additive. In particular, we can say:

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

In order to use less comparisons and collect more non-dominated policies, we try to explore policies with the great vector values 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(Alizadeh *et al.*, 2016).

For each $\bar{v}^{\pi}$ vector, there are $|S||A|$ new vectors which are $\{\bar{v}^{\pi}+A_{s_1,a_1}, \bar{v}^{\pi}+A_{s_1,a_2}, \cdots, \bar{v}^{\pi}+A_{s_{|S|},a_{|A|}}\}$. so the main problem is that the number of generated vectors in each iteration increases exponentially. In order to reduce exponential growth in $\bar{v}$ generation while keeping non-dominated vectors, our new algorithm suggests grouping several advantages in each iteration. In fact, in place of adding each $A_{s,a}$ to the $\bar{v}^{\pi}$ vector, we group several advantages and make a new equivalent vector from them. We implement that strategy with the help of a clustering methods on advantages. We use a hierarchical clustering on advantages with a cosine similarity metric [2]. After classifying advantages, suppose C is the set of resulting clusters. If a cluster

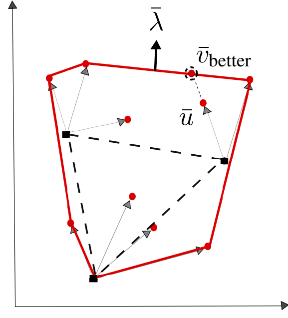


FIGURE 3: $\bar{\mathcal{V}}_{t+1}$ vectors selection after tree extension of $\bar{\mathcal{V}}_t$

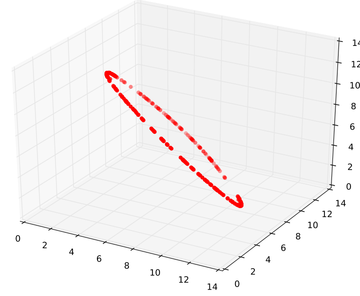


FIGURE 4: Generated non-dominated vectors for an MDP with 128 states, 5 actions, $d = 3$ and $\epsilon = 0.01$ (Algorithm 2)

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].

Since the sum of advantages is appended to $\bar{v}^{\pi'}$, cosine similarity metric favors advantages modifying $\bar{v}^{\pi'}$ in the same direction and their sum is a good approximation of this direction. Classifying advantages in $|C|$ number of clusters, enables us to generate less $\bar{v}$ vectors and more useful (policy, vector-valued function) pairs.

Figure 2 illustrates a tree with a root including policy π_0 and its vector-valued function $\bar{v}_0$. Taking $\bar{v}_0$ and computing $\bar{Q}$-function produce $|S||A|$ advantages. Without clustering on advantages, the root will have $|S||A|$ children nodes and they increases exponentially in the lower levels of tree. Clustering advantages allows us to have less nodes and more effective nodes. If we cluster the generated advantages of each node, it can be assigned to a pair of $(\pi_i^t, \bar{v}_i^t)$ such that π_i^t and $\bar{v}_i^t$ are computed according to the upper explanations. Such that we have less improved policies (or equally nodes) in each iteration (in each level of tree) with greater vector values.

4 How to approximate $\bar{\mathcal{V}}$ with ϵ precision

Since, extracting all members of $\bar{\mathcal{V}}$ is expensive, this section introduces an algorithm to generate an approximation of non-dominated vector-valued functions set $\bar{\mathcal{V}}_\epsilon$. 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. $||\bar{v}' - \bar{v}|| \leq \epsilon$.

The $math\bar{c}alV_\epsilon$ set is a convex set of all non-dominated vector-valued functions including all possible optimal policies for different users. Then every vector value $\bar{v} \in math\bar{c}alV_\epsilon$ is the value of at least one policy π from the set of all $|A|^{|S|}$ policies for a given MDP. With slight abuse of notations we indicate each pair $(\pi, \bar{v})$ as a node n indicating a policy and its related vector value function. We use a graph to visualize our approach in the easier way(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 clusters $\mathcal{A}$, producing a set of clusters $C = \{c_1, .., c_k\}$. It returns the set of new nodes which expand the tree under n and is defined from clusters:

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

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

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

Algorithm 2 Propagation: Generate approximation of non-dominated vectors $\bar{\mathcal{V}}_\epsilon$ using Advantages for a given VVMDP

Inputs: VVMDP($S, A, p, \bar{r}, \gamma, \beta$), $\{(\pi_i^0, \bar{v}_i^0)\}_{0\leq i\leq d}$, $\mathcal{K}$, ϵ
Outputs: set $\bar{\mathcal{V}}_\epsilon$

1: $N \longleftarrow \{(\pi_0^0, \bar{v}_0^0), .., (\pi_d^0, \bar{v}_d^0)\}$
2: $\bar{\mathcal{V}}^{\text{old}} \longleftarrow$ **ConvexHull**(**getVectors**(N))
3: **do**
4: $\quad N \longleftarrow \{\}$
5: $\quad \mathcal{C} \longleftarrow \{\}$
6: $\quad$ **for** $n \in \bar{\mathcal{V}}^{\text{old}}$ **do**
7: $\quad\quad$ add **expand**(n) to $\mathcal{C}$
8: $\quad$ **for** $n \in \mathcal{C}$ **do**
9: $\quad\quad$ **if CheckImprove**$(n, \bar{\mathcal{V}}^{\text{old}}, \mathcal{K}, \epsilon)$ **then**
10: $\quad\quad\quad$ add n to N
11: $\quad \bar{\mathcal{V}}^{\text{new}} \longleftarrow$ **ConvexHull**$(\bar{\mathcal{V}}^{\text{old}} \cup$ **getVectors**$(N))$
12: **while** $\bar{\mathcal{V}}^{\text{new}} \neq \bar{\mathcal{V}}^{\text{old}}$
13: **return** $\bar{\mathcal{V}}^{\text{new}}$

Notwithstanding 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 search tree and rolls up more non-dominated $\bar{v}$s of $\bar{\mathcal{V}}$.

Suppose that, in the 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 round points are the expanded nodes after implementing *Expand* function on any points from t-th iteration. 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.

Two remarks help seeing that this strategy does not prune optimal points. First for a polytope P of $\bar{v}$ vectors and any $\bar{\lambda}$ vector, the maximums of $\bar{\lambda} \cdot \bar{v}$ 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 explore an optimization of non-dominated $\bar{v}$s as subset of $\bar{\mathcal{V}}$ members which is marked as $\bar{\mathcal{V}}_\epsilon$. This algorithm uses four main functions: **ConvexHull**, **CheckImprove**, **expand** and **getVectors**.

ConvexHull gets a set of d-dimensional vectors, completes it with the **0** vector and generates its convex hull. This function returns vertices of the convex hull except the **0** vector. Function **expand** has just been described (Algorithm 1). **getVectors** simply gets a set of nodes and returns back the set of 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:

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

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 inputs a VVMDP, a set of d initial nodes, a stopping threshold ϵ and set of linear constraints of polytope Λ. The initial nodes are generated by choosing the d unit vectors of $\mathbb{R}^d$ [4] this yields d scalar MDP's where the VI method (Sutton & Barto, 1998) can discover the optimal policy and related vector-valued function.

In each iteration, the algorithm generates all children of any given $\bar{\mathcal{V}}^{\text{old}}$ member and makes a $\bar{\mathcal{V}}^{\text{new}}$ set of them using the *CheckImprove* function. The final solution is a set of non-dominated $\bar{v}$ vectors which are each optimal for one or several $\bar{\lambda}$ vectors inside the Λ polytope. Recall that the final error between our prediction of optimal policy — which will be a vector inside set $\bar{\mathcal{V}}_\epsilon$ — and the exact optimal policy is not ϵ. Because the utilized precision ϵ is a criterion for stopping non-dominated vectors generation algorithm.

Figure 4 is an example of MDP with $d = 3$ and 128 states. It demonstrates, how Algorithm 2 generates many vectors for an MDP with average precision $\epsilon = 0.01$. This shows that this algorithm is useful for IRMDPs with average weight dimension d. Because, for larger MDP with higher dimension d there are too many non-dominated vector-valued functions and exploring all these points for Algorithm 2 consumes too much time and memory.

5 Searching Optimal V^* by Interaction with User

After discovering an approximated set of all possible optimal $\bar{v}$ vectors for a known VVMDP offline and regardless of user preferences, we intend to find the optimal policy in the $\bar{\mathcal{V}}_\epsilon$ set with respect to a given user and her preferences. 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 otherwise, we approximate the maximum of $\bar{\lambda}^* \cdot \bar{v}^*$ for $\bar{v}^* \in \bar{\mathcal{V}}_\epsilon$ w.r.t the given user.

At the beginning, the Λ polytope is 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 explore the optimal $\bar{v}^*$ inside $\bar{\mathcal{V}}_\epsilon$. Since KDominance and Pareto comparisons are partial preferences, if two vectors are not comparable by any of them, the final solution is delegating the vectors comparison to the tutor. In Algorithm 3, $\succeq_D$ indicates the comparison regarding paretodominance or Kdominance, and $\succeq_K$ verifies only the Kdominance comparison.

Algorithm 3 is proposed as an approach for finding the optimal $\bar{v}^*$ from approximated set of non-dominated vector-valued functions $\bar{\mathcal{V}}_\epsilon$ according to user preferences. The main characteristic of this algorithm is that it selects the best pair to test to obtain a maximal information on Λ (line 27), in order to minimize the number of queries. In this algorithm, every time that we mention label of a pair $(\bar{v}_i, \bar{v}_j)$ it means: if vectors are comparable then we have:

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

Inspired by Jamieson & Nowak (2011)'s work to rank objects using pairwise comparisons, the algorithm first generates the set P of vectors in $\bar{\mathcal{V}}_\epsilon$ set, *i.e.*, $P = \{(v_i, v_j) \in \bar{\mathcal{V}}^2 | i < j\}$. Let T be the set of ambiguous pairs (To be Determined pairs) which are comparable neither by Pareto nor by Kdominance comparisons. D is the set of compared pairs (Determined 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. Function *Query* receives a pair of vectors and set of constraints $\mathcal{K}$ on Λ polytope. This function proposes the query like " $\bar{v}_i \leq \bar{v}_j$" to the user and appends the new cut to set $\mathcal{K}$ regarding the user preferences while assigning a label to this pair. Function *markDefined* is written as follows and contains

4. The selected $\bar{\lambda}$s inside Λ polytope are $e_1, .., e_d$ such that $e_i = (0, .., 0, 1, 0, ..., 0)$ is a d-dimensional vector with all element equal 0 except the i-th element

Algorithm 3 Search: 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 as a unit cube
Outputs: $\bar{v}^*$

```
1:  T = P
2:  D = U = ∅
3:  K = {0 ≤ x_i ≤ 1 s.t. 0 ≤ i ≤ d]}
4:  for (v_i, v_j) in T do
5:      if v_i ≽_D v_j then
6:          markDefined(v_i, v_j)
7:      else if v_i ≼_D v_j then
8:          markDefined(v_j, v_i)
9:  while T ≠ ∅ do
10:     for (v_i, v_j) in T do
11:         c_{i,j} = c_{j,i} = 0
12:         repeat
13:             choose random λ̄ ∈ Λ(K)
14:             for (v_i, v_j) in T do
15:                 if λ̄ · v̄_i > λ̄ · v̄_j then
16:                     c_{i,j} = c_{i,j} + 1
17:                 else if λ̄ · v̄_j > λ̄ · v̄_i then
18:                     c_{j,i} = 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 ≽_K v_j then
23:                 markDefined(v_i, v_j)
24:         else if c_{i,j} = 0 then
25:             if v_i ≼_K v_j then
26:                 markDefined(v_j, v_i)
27:     (v_i, v_j) = Argmin_{(v_i,v_j)∈T}(|c_{i,j} − 500|)
28:     (K, ans) = Query((v̄_i, v̄_j), K)
29:     if ans = i then
30:         markDefined(v_i, v_j)
31:     else
32:         markDefined(v_j, v_i)
33: return FindBest(D)
```

two parts: $remove(a, l)$ function that removes all pairs containing a from the list l (T set or D set) and puts them into the set of useless pairs U. Another function is $move((v_i, v_j), D)$ function which adds (v_i, v_j) pair into set of useless pairs U. This function updates the U set.

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

The first part of the algorithm (Lines 4 to 8) determines all comparable pairs and removes them from set T, i.e. T set includes all non-comparable pairs and will be investigated by the second part of algorithm.

The second part of the algorithm (Line 9) chooses the best pair from set of ambiguous pairs T 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. Our basic idea is computing the expected value of each pair $(\bar{v}_i, \bar{v}_j)$ inside T set i.e. $\mathbb{E}_{\bar{\lambda}\in\Lambda}[(\bar{\lambda}\cdot\bar{v}_i \geq \bar{\lambda}\cdot\bar{v}_j)]$, and choosing the one with the value close to $\frac{1}{2}$ which is the cut that divides Λ polytope into two almost equal sets. Using a Monte-Carlo method, the algorithm selects 1000 points randomly inside polytope Λ, and chooses the cut that divides the random points into two almost equal sets (Gilbert *et al.*, 2015). This new cut is chosen by the algorithm to query the user. Function *Query* poses the new query $\bar{v}_i \succeq \bar{v}_j$ to

the user and updates the set of constraints $\mathcal{K}$ regarding the user response. Thus polytope Λ shrinks after selecting this new cut.

The algorithm stops when the set of To be Determined pairs (set T) is empty, so all pairs are either determined (in set D) or useless (in set U). 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. At the, this algorithm gives the best vector- valued function with the highest rank.

6 Empirical Evaluation

Our experimental results include two parts: First part analyzes Propagation Algorithm (algorithm 2), while the second part focuses on Propagation-Search Value Iteration (PSVI) algorithm (algorithm 2 and algorithm 3)and compares it to IVI algorithm, an existing interactive value iteration method for exploring the optimal policy regarding agent preferences Weng & Zanuttini (2013)). All experiments have been implemented with Python version 2.7 and CPLEX has been used as a solver for linear programming problems.

6.1 Random MDP

A random MDP is defined by several parameters including its number of states n, its number of actions m and the dimension of its weight space d. All rewards are bounded between 0 and 1. The transition function has several properties: from any state s transitions are restricted to reach only $[\log_2(n)]$ states. For each pair (s, a) reachable states are chosen based on uniform distribution over the set of states, and transition probabilities are formed based on Gaussian distribution. The initial state distribution β is uniform and we choose the discount factor $\gamma = 0.95$. Recall that the weight polytope Λ is initialized as a unite d-dimensional hyper-cube.

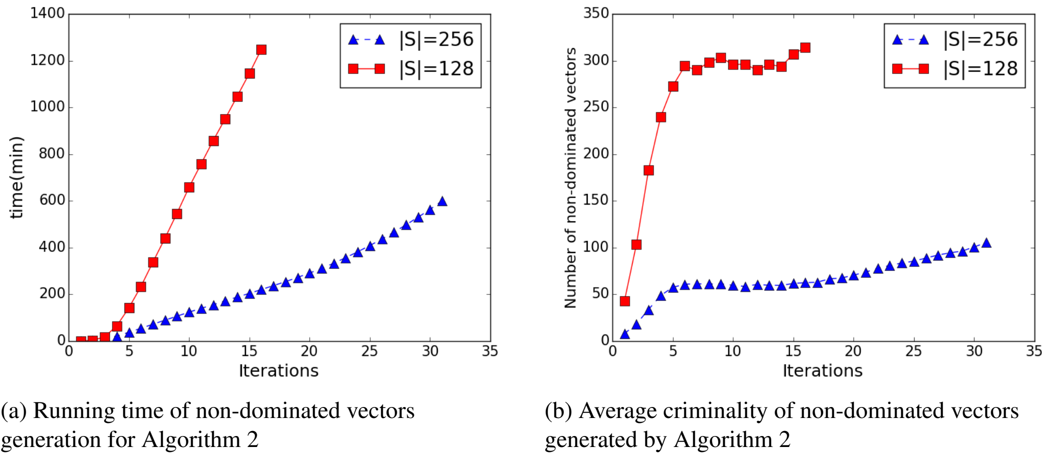
TABLE 1: $|\mathcal{V}_\epsilon|$ as a function of precision ϵ. Results are averaged on 10 random MDPs with $|A| = 5$.

$\lvert S\rvert = 128$	$\epsilon = 0.5$	$\epsilon = 0.2$	$\epsilon = 0.1$	$\epsilon = 0.05$
$d = 3$	3.0	18.89	105.4	212.59
$d = 4$	4.0	4.0	29.8	timeout
$d = 5$	5.0	5.0	5.0	timeout
$\lvert S\rvert = 256$	$\epsilon = 0.5$	$\epsilon = 0.2$	$\epsilon = 0.1$	$\epsilon = 0.05$
$d = 3$	3.0	6.59	98.2	209.2
$d = 4$	4.0	4.0	4.0	timeout
$d = 5$	5.0	5.0	5.0	5.0

Table 1 indicates how the number of non-dominated vector-valued functions changes with respect to the accuracy ϵ (refer to Algorithm 2). This table demonstrates —as we have expected— that $|\mathcal{V}_\epsilon|$ increases by raising precision (lowering ϵ). In this table for the cells with the same $|\mathcal{V}_\epsilon|$ size and d dimension, the algorithm stops after only one iteration. This indicates that the algorithm stops quickly for higher d dimensions and lower ϵ precision. On the other hand, increasing precision produces too many non-dominated vector-valued functions. Since propagation algorithm is computationally expensive in terms of time and memory, we have noted complex cases with « timeout » and have investigated one of these cases in Figures 5(a) and 5(a).

The two graphs in Figure 5 compare two random MDPs with the same actions ($|A| = 5$), the same d dimension (equal 4) and different number of states 128 and 256. They observe non-dominated vector generation with $\epsilon = 0.05$ as an example of timeout cases in our algorithm. Results have been averaged on 10 different MDPs. Diagram $5(b)$ indicates how the number of non-dominated vectors generated changes at each iteration in Algorithm 2. Also, diagram $5(a)$ shows the time elapsed at each iteration (in minutes) when generating non-dominated vectors. Graph $5(a)$ indicates that the propagation algorithm is computationally expensive and time consuming, but Figure $5(b)$ illustrates how $|\mathcal{V}|_\epsilon$ converges to a constant number after a small number of iterations.

Our approximation method of optimal policy exploration involves two parts: the technique for propagating optimal policies $\bar{\mathcal{V}}_\epsilon$ at an ϵ precision with respect to unknown rewards and the algorithm for extracting the best optimal policy from $\bar{\mathcal{V}}_\epsilon$ according to each agent. The former provides an approximation for the set



(a) Running time of non-dominated vectors generation for Algorithm 2

(b) Average criminality of non-dominated vectors generated by Algorithm 2

FIGURE 5: These graphs illustrate how algorithm 2 behaves on MDPs with 5 actions and two different number of states 128 and 256. Also $d = 4$ and ϵ precision is 0.05

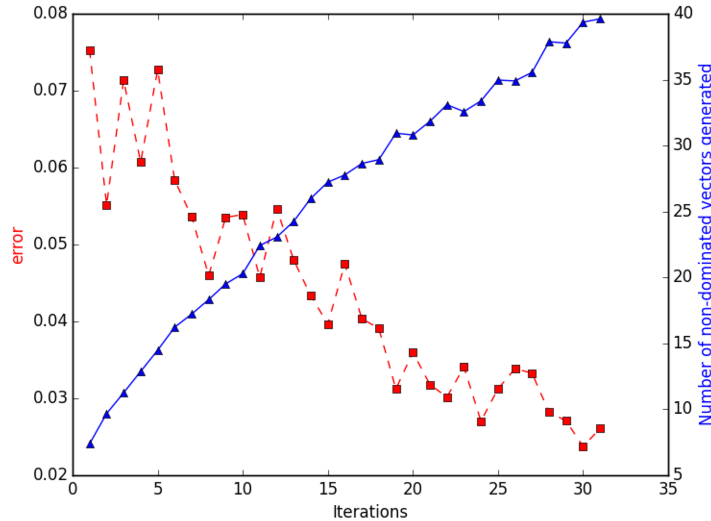


FIGURE 6: Error and number of non-dominated vector-valued functions v.s. iterations in Algorithm 2

of non-dominated vector-valued functions and the latter supplies the ability to discover the optimal policy compatible with user preferences during query elicitation of unknown rewards.

To examine the search of optimal policy (Algorithm 3) on results of the propagating algorithm, we try to explore the optimal policies of 20 different users. Each user is displayed as a $\bar{\lambda}$ inside polytope Λ and results have been averaged on random selection of agents. That means the error of our approach $\bar{v}^{\pi^*}$ is an average on error of all 20 selected users. Recall that the error for each user $\bar{\lambda}$ is computed as: $||\bar{\lambda}^T.\bar{v}^{\pi^*_{\text{exact}}} - \bar{\lambda}^T.\bar{v}^{\pi^*}||_\infty$ (It is assumed that the exact optimal policy of MDP is $\bar{v}^{\pi^*_{\text{exact}}}$).

Figure 6 indicates how error and numbers of generated non-dominated vectors change after each iteration on non-dominated vectors generation. The propagation algorithm generates a finite number of non-dominated vectors at each iteration. This figure illustrates how the number of generated non-dominated vectors increases across iterations. To calculate the error after each iteration, the search algorithm (Algorithm 3) gets the set of generated non-dominated vectors and finds the optimal policy satisfying user preferences. The demonstrated errors have been averaged for 20 users as explained in the previous paragraph. The graph shows how the error reduces after several iterations of the algorithm. This result has been tested on an MDP with 128 states 5, actions $d = 3$ and $\epsilon = 0.05$ in the propagation algorithm. The results have been averaged

on 10 random MDPs.

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

Methods	parameters	$d = 2$	$d = 3$	$d = 4$
PSVI	$\|\mathcal{V}_\epsilon\|$	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
	exploration 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 3: 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 the Search algorithm have been averaged on 50 random $\bar{\lambda} \in \Lambda$ (times are in seconds).

Methods	parameters	$d = 2$	$d = 3$	$d = 4$
PSVI	$\|\bar{\mathcal{V}}_\epsilon\|$	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
	exploration 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

Finally, Tables 2 and 3 compare two algorithms based on several measures, on MDPs with 128 states, 5 actions and d dimensions 2, 3 and 4. The ϵ accuracy is 0.2 and 0.1 respectively for the two tables. $|\bar{\mathcal{V}}|$ is the number of generated vector-valued functions for each dimension using our propagation algorithm (Algorithm 2) while the propagation time is the time of accomplishing this process in seconds. To examine the search optimal policy method (Algorithm 3) on results of the first algorithm, we try to explore the optimal policies of 50 different users. In these tables, we have compared our method with interactive value iteration in two measures: number of queries asked to the user, and calculation time.

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 3, both 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 by about half the number of queries by generating all possible optimal policies before starting any interaction with the user. Recall that the results for $d = 4$ and $\epsilon = 0.2$ are not reliable here, because the selected precision is too small for generating new vector-valued functions using advantages.

7 Conclusions and Future Works

We have introduced an approach for exploring the optimal policy for an MDP in which rewards come from a set of weights. Weights are unknown numerically and are normalized between 0 and 1. 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 optimal policies allows the optimal policy to be discovered while asking a considerably smaller number of questions to the agent. Our experimental results demonstrate the value of our approach.

In future studies, we will study 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 IRL problems, and comparing it to standard algorithms in this domain.

Acknowledgements

The authors gratefully acknowledge the reviewers for their helpful suggestions and comments.

Références

ALIZADEH P., CHEVALEYRE Y. & LEVY F. (2016). Advantage Based Value Iteration for Markov Decision Processes with Unknown Rewards. *International Joint Conference on Neural Networks (IJCNN)*.

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

BHATTACHARYA A. & DAS S. K. (2002). Lezi-update: An Information-theoretic Framework for Personal Mobility Tracking in PCS Networks. *Wirel. Netw.*, **8**(2/3), 121–135.

GILBERT H., SPANJAARD O., VIAPPIANI P. & WENG P. (2015). Reducing the Number of Queries in Interactive Value Iteration. In *4th International Conference on Algorithmic Decision Theory (ADT 2015)*, p. 139–152: Springer.

JAMIESON K. G. & NOWAK R. D. (2011). Active Ranking using Pairwise Comparisons. *CoRR*, **abs/1109.3701**.

PUTERMAN M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming*. Wiley.

REGAN K. & BOUTILIER C. (2009). Regret-based reward elicitation for markov decision processes. *UAI-09 The 25th Conference on Uncertainty in Artificial Intelligence*.

REGAN K. & BOUTILIER C. (2010). Robust policy computation in reward-uncertain mdps using nondominated policies. *Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*.

REGAN K. & BOUTILIER C. (2011). Robust online optimization of reward-uncertain mdps. *Twenty-Second Joint Conference on Artificial Intelligence (IJCAI 2011)*.

SUTTON R. & BARTO A. (1998). *Reinforcement Learning: An introduction*. Cambridge University Press.

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

WENG P. & ZANUTTINI B. (2013). Interactive Value Iteration for Markov Decision Processes with Unknown Rewards.

XU H. & MANNOR S. (2009). Parametric regret in uncertain markov decision processes. *48th IEEE Conference on Decision and Control*, p. 3606–3613.

ZIEBART B. D., MAAS A. L., DEY A. K. & BAGNELL J. A. (2008). Navigate Like a Cabbie: Probabilistic Reasoning from Observed Context-aware Behavior. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, UbiComp '08, p. 322–331: ACM.