Université Paris 13

École Doctorale Galilée

THÈSE

présentée par

Pegah ALIZADEH

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Spécialité: INFORMATIQUE

Elicitation and Planning in Markov Decision Processes with Unknown Rewards

soutenue publiquement le 09 Décembre 2016 devant le jury :

Nicolas MAUDET	Rapporteur
Bruno ZANUTTINI	Rapporteur
Jérôme LANG	Examinateur
Henry SOLDANO	Examinateur
Paolo VIAPPIANI	Examinateur
Yann CHEVALEYRE	Directeur de thése

RÉSUMÉ

Les processus décisionnels de Markov (MDPs) modélisent des problèmes de décisions séquentielles dans lesquels un utilisateur interagit avec l'environnement et adapte son comportement en prenant en compte les signaux de récompense numérique reçus. La solution d'un MDP se ramène à formuler le comportement de l'utilisateur dans l'environnement à l'aide d'une fonction de politique qui spécifie quelle action choisir dans chaque situation. Dans de nombreux problèmes de décision du monde réel, les utilisateurs ont des préférences différentes, donc, les gains de leurs actions sur les états sont différents et devraient être re-décodés pour chaque utilisateur. Dans cette thèse, nous nous intéressons à la résolution des MDPs pour les utilisateurs ayant des préférences différentes.

Nous utilisons un modèle nommé MDP à Valeur vectorielle (VMDP) avec des récompenses vectorielles. Nous proposons un algorithme de recherche-propagation qui permet d'attribuer une fonction de valeur vectorielle à chaque politique et de caractériser chaque utilisateur par un vecteur de préférences sur l'ensemble des fonctions de valeur, où le vecteur de préférence satisfait les priorités de l'utilisateur. Etant donné que le vecteur de préférences d'utilisateur n'est pas connu, nous présentons plusieurs méthodes pour résoudre des MDP tout en approximant le vecteur de préférence de l'utilisateur.

Nous introduisons deux algorithmes qui réduisent le nombre de requêtes nécessaires pour trouver la politique optimale d'un utilisateur: 1) Un algorithme de recherche-propagation, où nous propageons un ensemble de politiques optimales possibles pour le MDP donné sans connaître les préférences de l'utilisateur. 2) Un algorithme interactif d'itération de la valeur (IVI) sur les MDPs, nommé algorithme d'itération de la valeur basé sur les avantages (ABVI) qui utilise le clustering et le regroupement des avantages. Nous montrons également comment l'algorithme ABVI fonctionne correctement pour deux types d'utilisateurs différents: confiant et incertain.

Nous travaillons finalement sur une méthode d'approximation par critére de regret minimax comme méthode pour trouver la politique optimale tenant compte des informations limitées sur les préférences de l'utilisateur. Dans ce système, tous les objectifs possibles sont simplement bornés entre deux limites supérieure et inférieure tandis que le système

ne connaît pas les préférences de l'utilisateur parmi ceux-ci. Nous proposons une méthode heuristique d'approximation par critère de regret minimax pour résoudre des MDPs avec des récompenses inconnues. Cette méthode est plus rapide et moins complexe que les méthodes existantes dans la littérature.

ABSTRACT

Markov decision processes (MDPs) are models for solving sequential decision problems where a user interacts with the environment and adapts her policy by taking numerical reward signals into account. The solution of an MDP reduces to formulate the user behavior in the environment with a policy function that specifies which action to choose in each situation. In many real world decision problems, the users have various preferences, and therefore, the gain of actions on states are different and should be re-decoded for each user. In this dissertation, we are interested in solving MDPs for users with different preferences.

We use a model named Vector-valued MDP (VMDP) with vector rewards. We propose a propagation-search algorithm that allows to assign a vector-value function to each policy and identify each user with a preference vector on the existing set of preferences where the preference vector satisfies the user priorities. Since the user preference vector is not known we present several methods for solving VMDPs while approximating the user's preference vector.

We introduce two algorithms that reduce the number of queries needed to find the optimal policy of a user: 1) A propagation-search algorithm, where we propagate a set of possible optimal policies for the given MDP without knowing the user's preferences. 2) An interactive value iteration algorithm (IVI) on VMDPs, namely Advantage-based Value Iteration (ABVI) algorithm that uses clustering and regrouping advantages. We also demonstrate how ABVI algorithm works properly for two different types of users: confident and uncertain.

We finally work on a minimax regret approximation method as a method for finding the optimal policy w.r.t the limited information about user's preferences. All possible objectives in the system are just bounded between two higher and lower bounds while the system is not aware of user's preferences among them. We propose an heuristic minimax regret approximation method for solving MDPs with unknown rewards that is faster and less complex than the existing methods in the literature.

Contents

List of Figures

List of Tables

Abbreviations

ABVI	**A**dvantage **B**ased **V**alue **I**teration
CS	**C**urrent **S**olution
HLG	**H**alve **L**argest **G**ap
IRL	**I**nverse **R**einforcement **L**earning
IRMDP	**I**mprecise **R**eward **M**arkov **D**ecision **P**rocess
IVI	**I**nteractive **V**alue **I**teration
LP	**L**inear **P**rogramming
MDP	**M**arkov **D**ecision **P**rocess
MMR	**M**ini**M**ax **R**egret
MOMDP	**M**ulti **O**bjective **M**arkov **D**ecision **P**rocess
ND	**N**on-**D**ominated
PAPI	**P**reference-based **A**pproximate **P**olicy **I**teration
PBDP	**P**reference-**B**ased **D**ecision **P**rocess
PBPI	**P**reference-**b**ased **P**olicy **I**teration
PBRL	**P**reference-**b**ased **R**einforcement **L**earning
PSVI	**P**ropagation-**S**earch **V**alue **I**teration
RL	**R**einforcement **L**earning
SRPM	**S**elected **R**andom **P**oints **M**ethod
SRSA	**S**tate **A**ction **R**eward **S**tate **A**ction
VMDP	**V**ector-valued **M**arkov **D**ecision **P**rocess

Notations

Vectors	$\bar{}$	$\bar{x} = (x_1, \ldots, x_d) = [x_1, \ldots, x_d]^T$ (dimension d)
P-norm	$\|\|\ \|\|_p$	$\|\|\bar{x}\|\|_p = \sqrt[p]{\sum_{i=1}^{d} x_i^p}$
	$\|\|\ \|\|_1$	$\|\|\bar{x}\|\|_1 = \sum_{i=1}^{d} \|x_i\|$
	$\|\|\ \|\|_\infty$	$\|\|\bar{x}\|\|_\infty = max_i \|x_i\|$
CH-vertices(c)	$CH - vertices(c)$	the set of vertices of the convex hull built on the cluster c

Chapter 1

Introduction

Suppose Bob is a taxi driver, he drives in a city with several area zones. He can start his work from any area zone in the city and he picks up many clients during a day while moving among area zones. Suppose his goal is to increase his income by maximizing his total rout length during the day. Assume that he uses a computer which has also access to some statistics about traffic like, for example, for each road, it knows what is the probability of being slowed down. With this new information, the environment may now be modeled as a stochastic Markov Decision Process (MDP) and Bob's optimal strategy can be explored using MDP.

On the other hand, suppose that a generic taxi driver follows several goals at the end of the day, such as maximizing the average speed, minimizing the fuel consumption, minimizing the latency and so on. With this new information, the long term effectiveness of each decision in each area zone have several aspects regarding to the defined objectives in the environment. For instance by being in a junction T, an action like turning to the right can be evaluated as a good action w.r.t minimizing the fuel consumption while it is a bad action w.r.t maximizing the average speed. Then, instead of defining one single value for every state and decision, we assign several values for effect of one decision in one situation. By supposing d objectives in the environment, effects of decisions can be defined as d dimensional vectors.

Moreover, assume each driver has different types of preferences among objectives. For instance, Bob may prefer to minimize the fuel consumption more than maximizing the average speed or his colleague may prefer to maximize the average speed and minimize

the latency while he does not care about fuel consumption. Thus, the best strategy for Bob, is not the best for his colleague. Recall that, in the taxi driver's mind, preferences on objectives are ordinal rather than cardinal. It means that they can not evaluate routes with a single number, but for the two given routes A and B, they can naturally tell which one they prefer. Thus to find the most satisfying rout plan during the day for each given driver, the computer will have to ask each driver many queries in order to elicit his preferences. These queries might be of the form "do you prefer route A to route B ?". Based on these queries, the computer will build a compact model of driver's preferences, and it will then compute the driver's preferred routes.

Markov decision process (MDP) is a model for solving sequential decision problems where a user like Bob interacts with environment and adapts his policy by taking numerical reward signals into account. Typically, these models include the *states* of environment, the possible *actions* that agent can perform in each state, and the efficacy of each action in each state w.r.t performing a task in the environment. The effect of states on the environment should be specified as numerical feedback, namely *rewards*. The solution of decision theoretic model as MDP is to formulate the user behavior in the environment with a function, namely *policy* that specifies which action to choose in each state. If MDP parameters are given numerically, most reinforcement learning (RL) agent can autonomously learn any task. In this dissertation, we focus on MDPs with infinite size, i.e. MDPs without any final states such as the taxi driver example who starts in a start state and continues driving for a whole day.

In most of the research on planning in decision problems, the effects of actions in each state are codified in scalar rewards. Therefore, the solution is the policy with the maximum expected sum of rewards. There are many real world decision problems like taxi driver's problem that users have various preferences, and therefore, effect of actions on states are different and should be re-decoded for each user. To solve these kinds of problems, it is required to model MDP while reward values are unknown. More specifically, we assume MDPs with not-given rewards; the only information on rewards is that they are bounded in some intervals. A typical method for finding the optimal policy is using the optimization solution with respect to bounded polytope of all unknown rewards. Therefore our first approach is a robust approach; i.e. what can we say about optimal solutions, if we have a few information about rewards. In this part, we try to answer

these questions: *Is it possible to reduce complexity of optimization calculation?*, *Can we find a faster and more precise method rather than existed approaches in the literature?*.

There are many tasks that are more naturally described in terms of multiple, possibly conflicting objectives, e.g., a traffic control system should minimize both travel time and fuel costs for a car [Roijers et al., 2013]. For these cases, it is more convenient to define rewards as vectors. It means, effect of each action in any state should be defined as vector such that each element of vector represents the action effect w.r.t one of objective in the system. If rewards are vectors, the value of each policy will be vectors too. This form of MDP with vector rewards is known as Vector-valued MDP (VMDP). Technically, if the preferences of the user among objectives are not known, finding a policy maximizing objectives is not neither possible nor satisfying for different users with different preferences. One idea is to assign a weight vector to each user w.r.t her preferences among the objectives. For instance, for traffic control system, if the selected driver prefers to only minimize her fuel consumption, the vector weight on two objectives including the travel time and the fuel costs is $(0, 1)$. In fact, rewards are linear combination of objective rewards. Another question regarding this part is that *is it possible to approximate each user weight vector on objectives?*, *except optimization solutions, is there another method to find the optimal policy with respect to the unknown bounded reward weight vector?*

Any optimization method on MDPs with unknown rewards depends on our information about rewards. If there isn't enough information, the optimal solution will not be precise enough. To gain more information on rewards, it is required to communicate with users. A user's response to only one question on preferences between objectives can give us lots of information about the reward values. Therefore, another part of this thesis concentrates on the communication with user by generating various types of queries. This part can be implemented on two different views: MDPs with unknown rewards and VMDPs with unknown weight vectors. Thus, for two different models we are looking for answers to these questions: *Which type of queries should be proposed to the user in each model?*, *how to generate queries heuristically or precisely*, *when each query should be proposed to the user inside different policy calculation methods?* and *is it possible to reduce number of queries as small as possible?*

This thesis is divided into four main chapters. In chapter 2, we present a general background on Markov Decision Processes with unknown rewards. We introduce various structures of MDPs that have been used in the literature. In chapter 3, we review the work done so far in the field of sequential decision making under uncertainty. We first present some approaches that learn from observing an expert behavior in performing a task that should be learned by the system. Second, we introduce some methods in the literature that learn the optimal solution of MDPs by comparing preferences of user on various trajectories. And finally, we illustrate some settings to learn the optimal solution using communicating with users and preference elicitation during the optimal solution computation process.

Chapter 4 presents our two main contributions; In the first approach we present how to propagate all possible optimal policies without knowing user preferences. It means, for each user with different list of preferences, her optimal solution should be contained inside the set of explored optimal policies. Since this set includes all required information for finding optimal policies, we do not require MDPs anymore. Our second contribution of this chapter is communicating with user and learning her best satisfied policy from the small explored set of optimal policies. It means, the system carries the huge part of calculations on its part and consequently asks a few number of questions to the user. Another contribution is to reduce the prepossessing computation complexity. For this reason we modify interactive value iteration algorithm by clustering advantages. This method is supposed to converge faster to the optimal policy after asking fewer number of queries.

Our final contribution of this thesis in Chapter 5 is to propose a faster and less complicated approximation method for solving MDPs with unknown rewards using minimax regret method.

Chapter 2

Preliminaries

Foreword

Markov Decision Processes (MDPs) [Puterman, 2005] have proven to be successful in many systems that are controlled by sequential decisions, as game playing [Fearnley, 2010], robotics [Girard, 2014] and finance [Bäuerle and Rieder, 2011] among other models. They propose suitable frameworks for decision-making and optimally acting in stochastic environments. They generally assume a complete and correct world model, with stochastic state transitions and reward functions. This chapter first establishes the basic terminology and essential concepts related to MDPs (section 2.1), and then reviews different types of MDPs that have been proposed to model the environment in case of uncertainty in this modeling and some related definitions (section 2.2).

2.1 Markov Decision Processes

Formally, a discrete-time *Markov Decision Process (MDP)* is defined by a tuple of 6 quantities, $(S, A, p, r, \gamma, \beta)$. These characteristics are :

- *States.* The finite set of all states is denoted by $S = \{s_1, s_2, \cdots, s_n\}$. The States are completely observable. The number of states is denoted by n or $|S|$.

- *Actions.* The agent has the opportunity to interact with the environment by executing an action from the finite set of actions denoted by $A = \{a_1, a_2, \cdots a_m\}$. The number of actions is denoted by m or $|A|$.

- *State Transition Probability Distribution.* $p(s'|s, a)$ encodes the probability of going to state s' when the agent is in state s and chooses action a. Since p is a conditional probability distribution, one has $\forall(s, a),\ \sum_{s' \in S} p(s'|s, a) = 1$ (In literature, the state transition probability function has been represented with transition table T too).

- *Reward (cost) Function.* $r : S \times A \longrightarrow \mathbb{R}$, quantifies the utility of performing action a in state s. There exist other formulation for reward functions e.g. $r : S \times A \times S \longrightarrow \mathbb{R}$ or $S \longrightarrow \mathbb{R}$.

- *Discount Factor.* $\gamma \in [0, 1)$ indicates how less important are future rewards compared to the immediate ones.

- *Initial States Distribution.* $\beta(s) : S \longrightarrow [0, 1]$ indicates that the probability the agent starts her[1] interactions with the environment in state s is $\beta(s)$. Since β is a probability distribution on S, $\sum_{s \in S} \beta(s) = 1$.

We consider infinite horizon processes where the future rewards are exponentially discounted with γ. We illustrate the MDP model with a navigation problem for a cab driver [Bhattacharya and Das, 2002, Ziebart et al., 2008] as below.

Example 2.1. *We consider a cab driver in a city that is divided into* 8 *location areas (LAs)* $\{A, B, C, D, E, F, G, H\}$ *(See Figure 2.1). The accessibility among LAs is illustrated in Figure 2.2. In each area, the driver can decide moving to another area in four directions North, East, South and West or staying in the same area. The observation of the history of residents movements [Bhattacharya and Das, 2002] from* 9 *a.m. to* 9 *p.m. allows us to define for each zone, the probability of being visited by residents. The results indicate that zones* E, F, G *and* H *are never visited by residents and hence they can be excluded from our analysis. In Figure 2.3 is presented the MDP associated to the possible movements of the taxi. We suppose that the action "stay" fails in* 10% *of the cases and does not stay in the same zone. All together, the four "move" actions result in a change of zone with a probability of* 90% *according to Figure 2.3 and* 10% *in*

[1] we will refer to the agent as 'she'

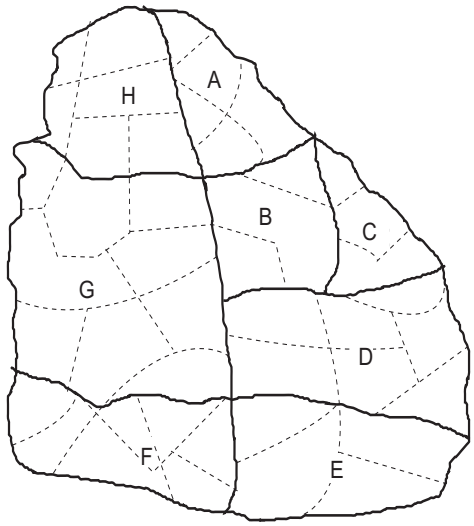

FIGURE 2.1: City divisions into 8 location areas

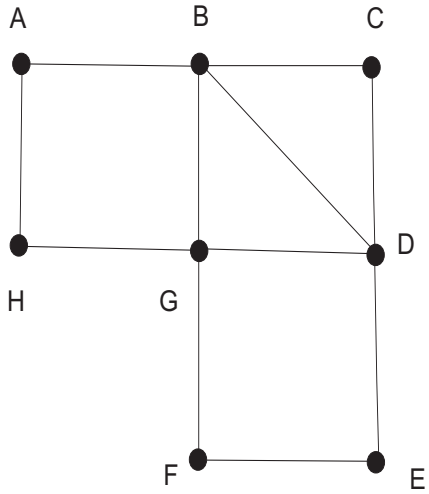

FIGURE 2.2: Mobility graph on location areas

other possible zones. The transition probabilities among the zones are given in Table 2.1. Therefore, the behavior of the cab driver can be modeled as an MDP:

- $S = \{A, B, C, D\}$
- $A = \{North, East, South, West, Stay\}$. *Note that the effect of actions in states are not deterministic.*
- T *is illustrated in Table 2.1. As an example the probability of moving to state* B *from state* C *after choosing action move West is* 0.9 *i.e.* $p(B|C, West) = 0.9$. *In the table, the acronyms St, N, E, S, W represent respectively Stay, North, East, South and West in the table.*
- r *gives a negative feedback to each action that is not compliant with the the goal of the driver, the start zone and destination zone of the cab driver. For example if the driver goal is to always drive in zone* A *and staying in the same zone regardless of*

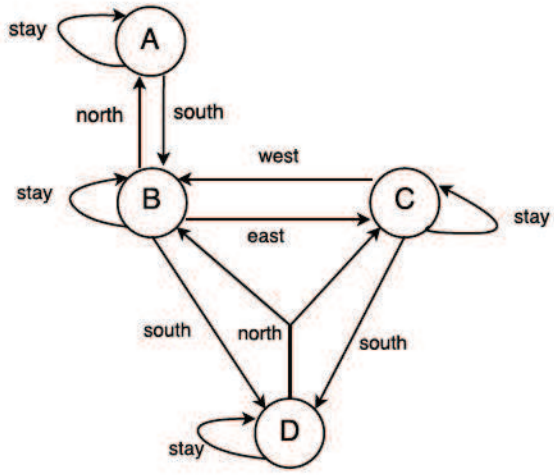

FIGURE 2.3: Movement on four states $\{A, B, C, D\}$.

$p(s'\|s,a)$		A					B					C					D				
s'	s	St	N	E	S	W	St	N	E	S	W	St	N	E	S	W	St	N	E	S	W
a		0.9	-	-	-	-	0.033	0.9	0.05	0.05	-	-	-	-	-	-	-	-	-	-	-
b		0.1	-	-	1	-	0.9	-	-	-	-	0.05	-	-	0.1	0.9	0.05	0.5	-	-	-
c		-	-	-	-	-	0.033	0.05	0.9	0.05	-	0.9	-	-	-	-	0.05	0.5	-	-	-
d		-	-	-	-	-	0.033	0.05	0.05	0.9	-	0.05	-	-	0.9	0.1	0.9	-	-	-	-

TABLE 2.1: probability distribution function T for cab driver.

her start point, the feedback of going to zones B, C and D are -1 while going to zone A has reward 1.

- γ *has a value between* 0 *and* 1. *We select it equal* 0.9 *in this example.*
- β *should be defined according to the probability of starting the process in a given state. For instance in our case the taxi driver drives all the time (the* horizon *is infinite). Thus, she can initiate the process in any state, i.e.* $\forall s \in \{A, B, C, D\},\ \beta(s) =$ 0.25.

After modeling the problem, the problem should be solved. An MDP can be executed for a given number of steps (finite horizon) or endlessly (infinite horizon). To execute an MDP, we need an elementary policy π. An elementary policy for an MDP selects actions according to the state:

- π can be *deterministic*: $\pi : S \longrightarrow A$ is a function from the set of states S into the set of actions A.

- *stochastic*: $\pi : S \longrightarrow \mathcal{P}(A)$ is a function from the set of states S into the set of probability distributions over actions. For any s, a , this function can be demonstrated as $\pi(s, a)$ too.

There are potentially two sources of randomness on the actions selection[2] which combine freely in MDPs: the non-determinism of *effect* of actions is coded in the State Transition table T (the image of an action is a probability distribution over states), while the non-determinism of *choice* of actions is coded in the stochastic elementary policies. Any one of these sources has the full expressive power of both combined.

Definition 2.1. A *policy* π at a finite horizon h is a sequence of h elementary policies, $\pi = (\pi_1, \pi_2, \cdots \pi_h)$ where each π_i is used once in turn.

When all the π_i are identical, the policy is said *stationary*. This is the only case considered when the horizon is unbounded. Strictly speaking, at the infinite horizon a policy is an infinite sequence of identical elementary policies ; in practice, it is noted as single elementary policy.

The set of deterministic policies at horizon h for a MDP (resp. stochastic policies) is indicated Π_h^D (resp. Π_h^S). The set of deterministic policies (resp. stochastic policies) with infinite horizon is indicated Π_∞^D (resp. Π_∞^S). Recall that the polices studied in this thesis have an infinite horizon, hence are *stationary*. They are also deterministic, while actions have stochastic effects. The set of all stationary policies for an MDP is noted Π and has cardinality $|A|^{|S|}$.

In order to compare policies and find the policy which is best compliant to the MDP goal, it is required to assign a real value to each policy. Techniques of policy evaluation and comparison are presented in the following sections.

2.1.1 Primal Formulation

For each state s, the utility of a stationary policy π is the expected discounted sum of rewards starting in state s. The evaluation function of each policy is called its *Value function* and it is defined by $V^\pi : S \longrightarrow \mathbb{R}$:

[2] Because there is a third potential source of randomness in MDPs, namely, stochastic rewards.

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r(s_t, a_t) | s_0 = s\right] \tag{2.1}$$

It is known that V^{π} satisfies the following recursive equation:

$$\forall\ s, V^{\pi}(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^{\pi}(s') \tag{2.2}$$

As $r(s, \pi(s))$ is bounded and $\gamma < 1$, $V^{\pi}(s)$ is a solution of the fixed-point equation 2.2. Equation 2.2 is called *Bellman Equation* [Bellman, 1957].

Value functions on states induce a partial ordering over policies, i.e., π is better than or equal to π' if and only if its value is greater for all states:

$$\pi \succeq \pi' \Leftrightarrow \forall\ s,\ V^{\pi}(s) \geq V^{\pi'}(s)$$

For an MDP, there is always at least one optimal policy π, s.t., $\forall\ \pi'\ :\ \pi \succeq \pi'$. Referring to Bellman Equation 2.2, we note that the optimal solution depends on any state $s \in S$ and it is optimal regardless of the initial state distribution β. Using β, the value function on states 2.1 can be translated into the state-independent value function:

$$V_{\beta}^{\pi} = \mathbb{E}_{s \sim \beta}[V^{\pi}(s)] = \sum_{s \in S} \beta(s) V^{\pi}(s) = \beta \cdot V^{\pi} \tag{2.3}$$

In the MDP setting, several functions are refereed to vectors too. By assuming an arbitrary order on the state set S, several function notations are referred to vectors in $\mathbb{R}^d$ including β, V^{π}.

The solution of the MDP is an optimal policy π^*, *i.e.* one with the greatest expected value among all possible values of the MDP.

$$\pi^* = \text{argmax}_{\pi \in \Pi} \beta \cdot V^{\pi} \tag{2.4}$$

Example 2.2. *Returning to Example 2.1, assume the taxi driver goal is to increase her incomes at the end of each day. If the north of city is the wealthy part of the area,*

then π' *can be an example of optimal policy:* $\pi'(a) = Stay$, $\pi'(b) = North$, $\pi'(c) = West$, $\pi'(d) = North$ *which conducts the driver to the north (zone a).*

Several algorithms are commonly used to find the optimal policy π^* exactly: *Value Iteration (VI)* [Bellman, 1957] and *Policy Iteration (PI)* [Howard, 1960]. For controlling these algorithms, it is better to consider the value of different actions: the *Q-Value Function* maps each state-action pair into $\mathbb{R}$. $Q^\pi(s,a)$ is the value of each state s after executing action a and then following policy π.

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^\pi(s'), \quad \forall\, s, a \tag{2.5}$$

The value function Q^π is related to value V^π because $V^\pi(s) = Q^\pi(s, \pi(s))$. Many Reinforcement Learning algorithms select Q-value functions instead of V value functions, because they allow to calculate a greedy policy associated to the value function directly. As an example, the greedy policy π' regarding the already calculated value function Q^π is any policy satisfying:

$$\pi'(s) \in \text{argmax}_{a \in A} Q^\pi(s,a) \tag{2.6}$$

The *Policy Improvement Theorem* [Sutton and Barto, 1998] assures that π' improves on π, unless it has the same values as π. In this case, they are both optimal. For each $s \in S$:

$$V^{\pi'}(s) \geq Q^\pi(s, \pi'(s)) = Q^\pi(s, \text{argmax}_a Q^\pi(s,a)) = \text{max}_a Q^\pi(s,a) \geq Q^\pi(s, \pi(s)) = V^\pi(s) \tag{2.7}$$

Definition 2.2. [Baird, 1993, Kakade, 2003] For an infinite horizon MDP, the *advantage* of a policy π in state s and action a is defined as

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{2.8}$$

The advantage $A^\pi(s,a)$ is the amount by which $V^\pi(s)$, the value function at state s increases or decreases if action a is taken in state s instead of following $\pi(s)$. The idea is the following: if π does not have any large advantage, then there is not much space to improve π. When searching for a policy π that competes against the optimal policy π^*, an algorithm need to minimize the advantages of π.

By taking initial distributions on starting states β into account, the advantage in pair (s, a) is modified as:

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

The advantages of π indicate if π can be improved in state s. A classical method for finding the optimal policy is Value Iteration Algorithm [Sutton and Barto, 1998] (given in Algorithm 1). The value iteration algorithm aims at calculating the optimal value function V^{π^*} (or, for simplicity of notation, V^*). The value iteration algorithm applies an iterative approach until the value improvement is under a given threshold ϵ (a user defined parameter). The smaller ϵ, the higher the precision of the predicted policy.

Algorithm 1 Value Iteration

Input: an MDP, ϵ precision
Output: optimal value function V^* according to ϵ precision

1: $t \longleftarrow 0$
2: $V_0 \longleftarrow \mathbf{0}$ vector of dimension $|S|$
3: **repeat**
4: $\quad t \longleftarrow t + 1$
5: $\quad$ **for each** s **do**
6: $\quad\quad V_t(s) \longleftarrow \max_a \{r(s, a) + \gamma \sum_{s'} p(s'|s, a)V_{t-1}(s')\}$
7: **until** $||V_t - V_{t-1}||_{\infty} \leq \epsilon$
8: $\pi_t(s) \longleftarrow \text{argmax}_a \{r(s, a) + \gamma \sum_{s'} p(s'|s, a)V_{t-1}(s')\}$
9: **return** π_t

All states are updated during each iteration until the stopping criteria is satisfied. An upper bound is imposed on the distance between V^* and the final V_t (the one that satisfies in stopping criteria) [Bertsekas and Tsitsiklis, 1996]:

$$||V_t - V^*||_{\infty} \leq \frac{\gamma}{1 - \gamma}\epsilon$$

It has been observed that the policy often becomes an optimal one long before the value function estimates converge to their optimal values. In the value iteration method, to get an optimal policy with ϵ precision can be obtained in a number of iteration that is polynomial in n, m and $\frac{1}{\gamma}$ [Littman et al., 1995].

The policy Improvement theorem (prooved in Equation 2.7) inspires another algorithm to solve MDPs called *policy iteration (PI)* (presented in Algorithm 2). The PI algorithm

starts with an initial arbitrary policy, takes rewards of states as their initial Q-values, i.e. $\forall\ s, a\ \ Q_0(s,a) = r(s, \pi_0(s))$, and computes a policy according to the maximum principle Q-value function. Then, it iteratively performs two steps: the policy improvement step (line 7 in Algorithm 2) which updates the current policy if any improvement is possible, and the policy evaluation step (line 8 in Algorithm 2), which updates the value of each state given the current policy. Moreover, the stopping criteria relies on policy change rather than value change.

Algorithm 2 Policy Iteration

Input: an MDP, ϵ precision
Output: optimal value function V^* according to ϵ precision

1: $t \longleftarrow 0$
2: $\pi_0 \longleftarrow$ arbitrary function defined on $S \rightarrow A$
3: for all $s \in S \quad V^{\pi_0}(s) \longleftarrow Q_0(s, \pi_0(s))$
4: **repeat**
5: $\quad t \longleftarrow t + 1$
6: $\quad$ for all $s, a \quad Q_{\pi_t}(s,a) \longleftarrow r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_{t-1}}(s')$
7: $\quad$ for all $s \in S \quad \pi_t(s) \longleftarrow \text{argmax}_{a \in A} Q_{\pi_t}(s,a)$
8: $\quad$ for all $s \in S \quad V_{\pi_t}(s) \longleftarrow Q_{\pi_t}(s, \pi_t(s))$
9: **until** $\pi_t = \pi_{t-1}$
10: **return** π_t

Besides the Value Iteration and the Policy Iteration algorithms there are more general algorithms to solve MDP problem such as TD(λ), Q-learning [Watkins, 1989] and State Action Reward State Action (SARSA) algorithms. Q-Learning algorithm directly approximates the optimal Q^*-value function without following the policy. This enables an early convergence of algorithm to the optimal policy.

2.1.2 From Primal to Dual Formulation

Referring to Equation 2.2, the optimal policy is a fixed point solution of Bellman equations. This can be calculated as the following linear program [Puterman, 1994]:

$$\begin{aligned} &\text{Minimize} \sum_{s \in S} \beta(s) V(s) \\ &\text{subject to:} \\ &V(s) \geq r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s') \ \ \forall\ s \in S, \forall a \in A \end{aligned} \tag{2.9}$$

The dual of 2.9 is:

$$\begin{aligned}
&\text{Maximize} \sum_{s\in S}\sum_{a\in A} r(s,a)f(s,a) \\
&\text{subject to:} \\
&\sum_{a\in A} f(s,a) = \beta(s) + \gamma \sum_{s'\in S}\sum_{a'\in A} p(s|s',a')f(s',a') \;\; \forall\, s \in S \\
&f(s,a) \geq 0 \;\; \forall s,a
\end{aligned} \tag{2.10}$$

where $f : S \times A \longrightarrow \mathbb{R}$ represents the $|S| \times |A|$ variables of the linear program; the first $|S|$ constraints of 2.10 are the flow constraints[3]. These constraints define the set of admissible values f ,namely $\mathcal{F}$. In the rest of the thesis, we will use the notation $f \in \mathcal{F}$ in linear programs, as a shortcut instead of rewriting the set of flow constraints defining $\mathcal{F}$. Thus, with this notation, the linear program 2.10 simply becomes:

$$\text{Maximize}_{f\in\mathcal{F}} \sum_{s,a} r(s,a)f(s,a) \tag{2.11}$$

Intuitively, $f(s,a)$ can be viewed as the total discounted probability of being in state s and taking action a. In the definition below $f(s,a)$ is defined more precisely.

Definition 2.3. A *State Occupancy Function* $f^\pi : S \longrightarrow \mathbb{R}$ is the total discounted probability of being in a state s w.r.t policy π:

$$f^\pi(s) = \sum_{s_0\in S} \beta(s) \sum_{t=0}^{\infty} \gamma^t p^\pi(s_t = s|s_0)$$

where $p^\pi(s_t = s|s_0)$ is the probability of being in state s by starting in state s_0 and following policy π.

Definition 2.4. An *Occupancy function* $f^\pi : S \times A \longrightarrow \mathbb{R}$ is the total discounted probability of being in a state and taking an action. Each policy π induces a unique occupancy function, $f^\pi(s,a)$ defined by[4]:

$$f^\pi(s,a) = f^\pi(s)\pi(s,a)$$

[3] As it will be explained in the rest of this section, $\forall\, s \;\sum_a f(s,a) = 1$. Thus, the dual LP 2.10 has a bounded objective.

[4] Note that π induces a unique f^π only for fixed β which is defined in a given MDP.

Note that $f^\pi(s,a)$ is not a real probability because $\sum_s \sum_a f^\pi(s,a) = \frac{1}{1-\gamma}$. Each f^π belongs to the set $\mathcal{F}$ of visitation frequencies, which are feasible solutions of the dual linear problem 2.10 [Puterman, 1994]. If f^π is a solution of the dual program 2.10, the policy π related to the occupancy function f^π can be obtained from this formulation:

$$\pi(s,a) = \frac{f^\pi(s,a)}{\sum_{a' \in A} f^\pi(s,a')} \tag{2.12}$$

By abuse of notation, we use f instead of f^π, if we know this occupancy function is related to which policy π. Referring to Definition 2.4 and value function property in equation 2.3, we can see that the value function of policy π can be computed as follows:

$$\beta \cdot V^\pi = \sum_{s \in S} \beta(s) V^\pi(s) = \sum_{s \in S} \sum_{a \in A} r(s,a) f(s,a) = f \cdot r$$

where f and r are two vectors of dimension $|S||A|$. Since the optimal value function V^*_β is $\sup_{\pi \in \Pi} \sum_{s \in S} \beta(s) V^\pi(s)$, it satisfies the previous equation, i.e.:

$$\beta \cdot V^* = \sum_{s \in S} \sum_{a \in A} r(s,a) f^*(s,a) \tag{2.13}$$

where $f^* = \text{argmax}_{f \in \mathcal{F}} \sum_{s \in S} \sum_{a \in A} r(s,a) f(s,a)$[5]. In this dual method, to find the optimal policy [Puterman, 1994], occupancy functions facilitate optimal policy computation because they are independent of reward values and can be calculated from dual linear program 2.11.

2.2 MDPs with Unknown Rewards

The algorithms presented in section 2.1.1 solve the optimal policy problems in MDPs directly by using Bellman equation. These methods essentially require the full observation of states S and actions A, the full knowledge of the reward functions $r(s,a)$ in each state and action, and the transition probability model $p(.|s,a)$. In real cases, specifying the reward function is generally a problem (assuming at least that the transition function can

[5] f^* and V^* represent respectively f^{π^*} and V^{π^*} such that π^* is the founded optimal policy

be computed). To relax the reward knowledge requirements, there are various structures on MDPs that only need to know that bounds on the rewards. Thus, the mentioned approaches rely on the assumption that, from a modeling point of view, dealing with bounded rewards might be more convenient than dealing with exact numerical ones. This section first establishes the general setting of MDPs without knowing the exact specification of reward values, and then from that it introduces a different formulation of MDPs with reward vectors. Next, it illustrates how several problems can be formulated with reward vectors MDPs. Afterward, the classic planning algorithms in MDPs are reviewed with respect to the new formulation.

2.2.1 General Setting

In many cases, obtaining a reward function can be difficult, because it requires human judgments of preferences and trade-offs, this is the case of cognitive assistance technologies [Boutilier et al., 2006a], or expensive computation such as computing a function of resource availability in autonomic computing [Boutilier et al., 2003]. Another case that makes numerical specification of rewards difficult is when there is no unique objective to be optimized and the models potentially depend on multiple objectives. For example, a taxi driver who should minimize travel time and should also minimize the fuel cost. Or, in designing computer systems, one client is interested not only in maximizing performances but also in minimizing power. Therefore, a usual MDP model is not sufficient to express these natural problems.

In Example 2.1, rewards have been specified precisely according to the final goal, namely finding the shortest trajectory that will result in the location area a. In this example, the system has an optimal trajectory solution which is the same for all drivers, regardless of different preferences or various objectives for various drivers. But if the model is required to account for the preferences of specific users on dissimilar objectives, the presented reward function is not adequate.

Example 2.3. *In the taxi driver example, we assume that drivers drive in the city during a day without choosing an area zone as final destination. Instead, they select their trajectories according to various objectives; such as:*

- *minimizing the travel time*

- *reducing the fuel usage*
- *maximizing the average speed* [6]
- *minimizing the number of direction changes* [7]

According to the listed objectives, a satisfying algorithm is the one that suggests different trajectories to different drivers. This means that the algorithms that proposes the same optimal solution for all drivers can not be reliable in this setting.

Example 2.4. *Assume that there are* 1000 *drivers with various preferences on the given objectives list. As an example, we take two drivers as driver* 1 *and driver* 2 *and try to find the optimal policy for each one. Suppose that the driver* 1 *prefers to reduce fuel consumption and to maximize the average speed while the driver* 2 *intends to follow trajectories with a small number of direction changes and she does not care about the rest of objectives. If we know how to define any* 20 *effects of (state, action) pairs for each user, the classical methods for solving MDPs can be implemented to find the optimal trajectory for each driver.*

The issue is that determining rewards for huge MDPs with many number of states actions or solving MDP for enormous number of drivers is problematic. In addition, translation of user preferences like “good", “bad" or “do not care", into precise numerical rewards is difficult. For instance in the given example, it is not clear for us how to specify rewards numerically for each user to support her preferences in the system.

Moreover, there are two types of users, *confident user* and *uncertain user*. The former user replies all comparison questions correctly regarding her final goal, while the latter one responds some queries with few percentages of error. It means for any two comparison query, the certain user answers the question all the time correctly regarding her preferences. On the other hand, the uncertain user sometimes can not give an exact answer to the query regarding her final priorities on the objectives. This is because of several reasons:

- Either the comparison questions are not easy to be answered by users

[6] That is result of less stops and no stuck in heavy traffics

[7] For taxi drivers, it is easier to remember the trajectories with less direction changes, for instance going to the north all the time and once turning to the west is easier to handle than go to north go to west go to north go to east and then go to the north.

- Or the user is not certain about preference of some objectives rather that some other objectives

To solve this problem, the simplest solution is to assume that rewards are unknown and are considered as some bounded variables. As the simplest consideration, we can say all 20 rewards in taxi driver example are unknown and bounded between 0 and 1.

In literature on preference elicitation, there are many studies that do not require full specification of the utility functions — either value functions or Q-value functions in our case — to make optimal or near optimal decisions [Boutilier et al., 2006a, Chajewska et al., 2000]. There are various optimization solutions for narrowing the set of preferred policies by knowing the bounds on rewards. Those bounds allow to confirm dominance of some policies over some other ones. To narrow more closely the set of optimal solutions, preference elicitation methods can be used. It means shrinking some reward bounds and eliciting part of them gives more information and finally gets a better approximation of the optimal policy.

A general formulation of MDP with partial knowledge of reward function takes as input a set of reward functions. Regan and Boutilier [2012] defined an *Imprecise Reward MPD (IRMDP)* by replacing rewards r by a set of feasible reward functions $\mathcal{R}$. Assuming rewards are bounded under some constraints, $\mathcal{R}$ will be a convex polytope defined by these constraints. These bounds can be defined either by a user or a domain expert or they can be concluded from observation of user behaviors such as in inverse reinforcement learning [Ng and Russell, 2000]. In our case, those rewards can be confined between 0 and 1 as a general case.

Definition 2.5. An Imprecise Reward MDP (IRMDP) is an MDP $(S, A, p, \mathcal{R}, \gamma, \beta)$ in which the reward function has been replaced by a bounded polytope $\mathcal{R}$ of so called *admissible* reward functions. This convex polytope can be defined by a linear constraints set $\mathcal{R} = \{r | \mathbf{A}r \leq \mathbf{b}\}$. where $|\mathcal{R}|$ denotes the number of constraints in the polytope.

Since rewards are not given numerically, they should be assigned by variables and they should be approximated using optimization techniques. This means that an IRMDP is an MDP with reward variables and a given polytope of rewards $\mathcal{R}$.

Example 2.5. *In the taxi Driver example, we need* 20 *variables for the unknown and bounded rewards, namely* $r(a, Stay)$, $r(a, North)$, $r(a, Eest)$, $r(a, South)$, $r(a, West)$,

$\cdots$, $r(d,s)$, $r(d,North)$, $r(d,East)$, $r(d,South)$ *and* $r(d,West)$. *This means that the reward functions are embedded in a polytope in* 20 *dimensional space.*

As the MDP works for all users and the unknown rewards are restricted to the polytope $\mathcal{R}$, referring to the dual formulation of MDP (given in Equation 2.11), the MDP solution can be computed optimistically from the following problem w.r.t f and r:

$$f^* = \text{Argmax}_{f\in\mathcal{F},r\in\mathcal{R}} \sum_{s\in S}\sum_{a\in A} r(s,a)f(s,a) \tag{2.14}$$

This program maximizes the value function for the most favorable reward function r. Note that a reward function is a real function on the set of reward variables, so each reward function can be viewed as a vector with coordinates indexed by: $r = (r(s_1,a_1), r(s_1,a_2), \cdots, r(s_1,a_m), \cdots, r(s_n,a_1), r(s_n,a_2), \cdots, r(s_n,a_m))$[8]. If rewards are given precisely, the optimal solution for MDP would be calculable with the following linear program:

$$\begin{aligned}
&\text{Maximize } \sum_{s\in S}\sum_{a\in A} r(s,a)f(s,a)\\
&\text{subject to:}\\
&\sum_{a\in A} f(s,a) = \beta(s) + \sum_{a\in A}\sum_{s'\in S} p(s'|s,a)f(s,a) \;\; \forall s\in S\\
&f(s,a) \geq 0 \;\; \forall s,a
\end{aligned} \tag{2.15}$$

However, the probabilities of rewards are unknown, we are in decision under uncertainty setting problem. Several methods are available for solutions for decision making under uncertainty [Pažek and Rozman, 2009] including optimistic approach(maxmax), pessimistic approach (maxmin), coefficient of optimistic approach (hurwicz criterion) or Savage's and Laplace's criterion.

[8]Hence $r \in \mathbb{R}^{mn}$ and $\mathcal{R} \subset \mathbb{R}^{mn}$. When a function is considered as a vector, the standard $^-$ notation in the name of vectors is not used.

2.2.1.1 Robust Objective Functions

A well known approach of solving MDPs with respect to polytope $\mathcal{R}$ (See 2.14) is the minimax regret method. To explain it we need to introduce the following functions:

Definition 2.6. Given an IRMDP$(S, A, p, \mathcal{R}, \gamma, \beta)$ such that $\mathcal{R}$ is a bounded polytope of unknown reward functions. Let $r \in \mathcal{R}$ be a reward function. The *Regret* of a policy $\pi \in \Pi$ with respect to r is defined as:

$$\text{Regret}(\pi, r) = \max_{\pi' \in \Pi} \{V_r^{\pi'} - V_r^{\pi}\}$$

This regret is the performance gap between the policy π and the optimal policy assuming r function is the actual one. Since the actual reward is unknown, an adversary player can choose any reward function $r \in \mathcal{R}$ and vary the regret of decision maker. Hence the following definitions:

Definition 2.7. The *MaxRegret* of policy π with respect to feasible set of reward $\mathcal{R}$ is:

$$\text{MaxRegret}(\pi, \mathcal{R}) = \max_{r \in \mathcal{R}} \text{Regret}(\pi, r) = \max_{r \in \mathcal{R}} \max_{\pi' \in \Pi} \{V_r^{\pi'} - V_r^{\pi}\}$$

Definition 2.8. The *MiniMax Regret (MMR)* of a feasible set of reward $\mathcal{R}$ is:

$$\min_{\pi \in \Pi} MaxRegret(\pi, \mathcal{R}) = \min_{\pi \in \Pi} \max_{r \in \mathcal{R}} \max_{\pi' \in \Pi} \{V_r^{\pi'} - V_r^{\pi}\}$$

The minimax regret provides worst-case bounds on missed opportunities. Specifically, let π be a policy supporting the minimax regret and let δ be the max regret achieved by π. Then, given any instantiation of r, no policy has expected value greater than δ more than that of π [Regan and Boutilier, 2008].

Any presented solutions on this formulation can be applied on small MDPs with few states and actions. In order to deal with the curse of dimensionality, we will explain another formulation of IRMDP with a smaller dimension of space and consequently with a simpler calculation methods.

2.2.2 Reward Vectors

In the previous section, we have introduced a new formulation of MDPs when reward values are unknown or partially known. In this section, we will introduce an MDP with vector rewards, namely *Vector-valued MDP (VMDP)*. While various types of problems that can use VMDPs can be introduced [Akrour et al., 2012, Roijers et al., 2013, Viappiani and Boutilier, 2010, Weng, 2011], we illustrate how the presented problems transform to VMDPs.

Definition 2.9. [Wakuta, 1995] A Vector-valued MDP (VMDP) is defined by a tuple $(S, A, \bar{r}, p, \gamma, \beta)$ where the vector-valued reward function $\bar{r}$ is defined on $S \times A$ and $\bar{r}(s,a) = (r_1(s,a), \cdots, r_d(s,a)) \in \mathbb{R}^d$ is the vector-valued reward defined by $\bar{r}$ in (s,a).

Recall that a scalarizable VMDP is a special case of IRMDP. Concurrently with VMDP the term *Multi-Objective MDPs (MOMDPs)* is also used when several objectives are considered together and these objectives are not directly comparable [Roijers et al., 2013]. In this case, each $r_i : |S| \times |A| \longrightarrow \mathbb{R}$ is defined according to the i-th objective in the MDP.

Example 2.6. *Referring to the taxi driver example, the vector-valued reward function has dimension $d = 4$, due to the four different objectives given in Example 2.3 (minimizing the travel times, reducing the fuel usage, maximizing the average speed and minimizing the number of direction changes). A driver having a vector of rewards $(1, 0, 0, 0)$ only wants to minimize his travel time. On the other hand, a driver having a vector of rewards $(0, 1, 0.5, 0)$ wants to reduce fuel usage and thinks about maximizing average speed.*

Similarly to MDPs, a value function $\bar{V}^\pi$ in a VMDP specifies the expected cumulative discounted reward vector:

$$\bar{V}^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \bar{r}(s_t, a_t) | s_0 = s \right] \tag{2.16}$$

Basic techniques of MDPs can be applied component-wise to vector-valued functions. For instance, in each state the discounted sum of reward vectors can be computed from the Bellman Equation:

$$\bar{V}^{\pi}(s) = \bar{r}(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, a)\bar{V}^{\pi}(s') \tag{2.17}$$

Finally by taking the initial distribution β into account, we have:

$$\bar{V}^{\pi}_{\beta} = \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}(s)] = \sum_{s \in S} \beta(s)\bar{V}^{\pi}(s) \tag{2.18}$$

where $\bar{V}^{\pi}_{\beta}$ is a vector of length d.

Pointwise domination does not ensure the existence of an optimal policy π^* in VMDP. Indeed, defining the policy with the greatest vector-valued $\bar{V}$ yields:

$$\forall \pi \in \Pi \quad \forall s \in S \quad \bar{V}^{\pi^*}(s) \geq \bar{V}^{\pi}(s) \tag{2.19}$$

But two pointwisely maximal vectors may be pointwisely uncomparable, preventing to decide which one is optimal without additional information. Such information can be provided in the form of a scalarization function which we will discuss in the following.

In the following subsections, we introduce problems that can be transformed to VMDP and we illustrate how this transformation happens. There are several problems modeled in VMDP formulation including inverse reinforcement learning, sensori motor states, preference learning and so on. These techniques are introduced in Section 3.

2.2.3 Multi-Objective MDPs with Linear Scalarization

There are several approaches in the literature for solving multi-objective MDPs (MOMDPs) such as using Pareto-dominance policies [Moffaert and Nowé, 2014] or using scaliraztion function [Roijers et al., 2013]. In this section ,we concentrate on linear scalarization function and we will show how to solve MOMDPs by generating a linear scalarization function.

A general technique to transform a VMDP into an MDP is scalarization, i.e. providing a function from the vector-valued reward space to the real numbers. Linear scalarization is almost universally used: a scalar reward function is computed by $r(s, a) = \bar{\lambda} \cdot \bar{r}(s, a)$ where $\bar{\lambda} = (\lambda_1, \cdots, \lambda_d)$ is a weight vector scalarizing $\bar{r}$s. Using this new reward function

$\bar{\lambda} \cdot \bar{r}$, the VMDP will change to the ordinary Markov decision process MDP($\bar{\lambda}$) depending on $\bar{\lambda}$ weight. For the sake of abbreviation $\bar{\lambda}$ is called *preference vector* in the rest of this manuscript. Now, the total discounted reward in π is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} \bar{\lambda} \cdot \bar{r}(s_t, a_t) | s_0 = s\right] = \bar{\lambda} \cdot \bar{V}^{\pi}(s) \tag{2.20}$$

Finally:

$$V_{\beta}^{\pi} = \bar{\lambda} \cdot \bar{V}_{\beta}^{\pi} \tag{2.21}$$

As can be seen, for a given VMDP changing $\bar{\lambda}$ weight, will change policy value functions and consequently will change the optimal policy regarding to $\bar{\lambda}$ vector.

Providing a weight for objectives in MOMDP is a particular case of scalarization. Another case is when, in the general setting, the states are defined as a set of features and reward functions are given or computed separately for different features, then combined for each set according to its features.

We present two scenarios from a survey of multi-objective MDPs (MOMDP) [Roijers et al., 2013], for modeling MDPs with unknown rewards. Note that, as each objective has its own reward, only the relative weight $\bar{\lambda}$ of rewards is a question in this case:

- *unknown weight scenario* occurs when $\bar{\lambda}$ is unknown only at the moment when planning or learning happen.
- *decision support scenario* occurs when $\bar{\lambda}$ is unknown through the entire decision-making process.

This thesis deals with the case where weights are not known and they will be explored using a decision support scenario. In this case, without specifying the $\bar{\lambda}$ vector, finding a solution for MDP (planning or learning phase) is impossible.

Example 2.7. *Referring to Example 2.3 for the sake of simplicity, assume there are only two objectives in the system : minimizing fuel consumption and minimizing latency (i.e., the time that driver need to reach her destination). For the VMDP, reward vectors*

$\bar{r}(s,a) = (r_1(s,a), r_2(s,a))$ *are given, involving two reward functions* $r_i : S \times A \longrightarrow \mathbb{R}$ $i = 1, 2$.

For vector $\bar{V}^\pi = (\bar{V}^\pi_{latency}, \bar{V}^\pi_{energy})$*, the user assigns a weight vector* $\bar{\lambda} = (\lambda_{latency}, \lambda_{energy})$ *to objectives according to her preference. Thus, we have* $\forall \pi \;\; V^\pi = \bar{\lambda} \cdot \bar{V}^\pi$.

In fact, approximating user preferences and finding the $\bar{\lambda} \in \Lambda$ weight vector satisfying user priorities in VMDP problems is an issue and will be studied in the rest of this thesis.

2.2.3.1 Categorized rewards

Weng [2011] technique for defining a VMDP structure with bounded polytope Λ is based on classifying reward values in several categories. Contrary to MOMDPs, one single objective is initially defined, and rewards for this single objective are not known. A simplifying modeling decision is made: using a qualitative set E of reward values. Hence the reward function is defined as $r : S \times A \longrightarrow E$ and each reward can be seen as a categorical variable (in the sense used by statistics).

Assume there are d qualitative rewards $E = \{\lambda_1, \lambda_2, \cdots, \lambda_d\}$. Then, for each state s and each action a, $r(s,a)$ takes one value from E. For an MDP with $|S||A|$ unknown rewards there are d total unknown values, and in most examples $d < |S||A|$. It means also that, in the model, several rewards on different pairs of (state, action) are known to have the same symbolic value, whatever numeric mapping is used for these symbolic values.

Example 2.8. *Consider a single objective "maximizing average speed". More, E includes three qualitative values:* λ_1 = *"low",* λ_2 = *"normal", and* λ_3 = *"high". While a numeric reward function for this objective may involves* 20 *different values, the qualitative one only involves 3 different values.*

It is initially only known that E is the set of all unknown reward values for the MDP[9]. Without loss of generality, we can confine λ variables in a d dimensional hyper-cube that is indicated as $\Lambda = \{\bar{\lambda} = (\lambda_1, \cdots \lambda_d) | \; \mathbf{0} \leq \mathbf{I}\bar{\lambda} \leq \mathbf{1}\}$. Where $\mathbf{I}$ is a unit matrix of dimension $d \times d$ and $\mathbf{0}$ and $\mathbf{1}$ are respectively 0 and 1 vectors of length d.

[9]Weng added another characteristic on unknown weights [Weng, 2012, Weng and Zanuttini, 2013]. He assumed that E is an ordered set $(E, >)$, so his rewards are in fact ordinal variables, and the order is such that $\lambda_1 >, \lambda_2 >, \cdots, > \lambda_d$.

To transform the initial qualitative MDP into a VMDP, a reward vector $\bar{r} : S \times A \longrightarrow \mathbb{R}^d$ is defined for each pair (s, a) :

$$\text{if } r(s_i, a_j) = \lambda_k \quad \text{then} \quad \bar{r}(s_i, a_j) = e_k \tag{2.22}$$

where e_k denotes the d dimensional vector $(0, \cdots, 0, 1, 0, \cdots, 0)$ having a single 1 in k-th element. Note that the reward $r(s, a)$ is a dot product between two d-dimensional vectors:

$$r(s, a) = \bar{\lambda} \cdot \bar{r}(s, a) \tag{2.23}$$

By considering the $\bar{r}(s, a)$ vectors, we have an MDP with vector-valued reward function.

2.2.4 Non-dominated Policies

An intermediate situation is when weights are not exactly known, but are nevertheless restricted to a feasible set of weights. This allows to reduce the search space of policies to *non-dominated policies*, which are defined bellow [Regan and Boutilier, 2010].

Definition 2.10. An uncertain reward MDP with feasible set of reward weights Λ being given, policy π is non-dominated in Π with respect to Λ if and only if

$$\exists\, \bar{\lambda} \in \Lambda \;\text{ s.t. }\; \forall \pi' \in \Pi \;\; \bar{\lambda} \cdot \bar{V}_\beta^\pi \geq \bar{\lambda} \cdot \bar{V}_\beta^{\pi'}$$

In other words, a non-dominated policy is optimal for some feasible rewards. Let $ND_\Lambda(\Pi)$ denotes the set of nondominated policies w.r.t Λ. Since the optimal policy is the policy that should be dominating the other policy values with respect to one special λ satisfying user priorities, the optimal policy is inside $ND(\Pi)$. In fact computing the $ND(\Pi)$ set can help us to reduce the complexity of searching optimal solution for MDPs with uncertain rewards.

As explained in the previous sections, the set of all possible weight vectors $\bar{\lambda}$ for the VMDP is noted as Λ and the set of all possible policies as Π. In VMDP to each policy π a d-dimensional vector $\bar{V}_\beta^\pi$ is assigned. By knowing all policies, the set of their vector

value functions is showed by $\bar{\mathcal{V}}$ ($\bar{\mathcal{V}} = \{\bar{V}_\beta^\pi : \pi \in \Pi\}$). According to equations 2.20 and 2.21, finding an optimal policy for a VMDP with unknown reward weights Λ can be boils down to explore the interaction between two separate d-dimensional admissible polytopes: $\bar{\mathcal{V}}$ and Λ.

Suppose each vector inside the Λ polytope represents a user and her priorities on objectives. Preferences of a given user can be approximated by asking questions to the user regarding her preferences between two reward-vectors and pruning accordingly part of the Λ polytope. Preference elicitation methods and related approaches will be introduced in Section 3.3.

Finally, in Chapter 4, we will work on the $\bar{\mathcal{V}}$ set and its characteristics. In order to solve the difficulty of $\bar{\mathcal{V}}$ generation, we will require to use the concept, namely *Convex Hull*. We define convex hull here and will give more details and explanations in Chapter 4.

Definition 2.11. *convex set* :A set of vectors S in a d-dimensional vector space is said to be convex if for any two vectors $\bar{a}, \bar{b} \in S$ and any $t \in [0, 1]$, the vector $t\bar{a} + (1-t)\bar{b}$ is also in S. That means the line segments connecting $\bar{a}$ and $\bar{b}$ should be contained in S too[10].

Definition 2.12. The *convex hull* of a set of vectors S in a d-dimensional vector space is the smallest convex set in the same space that contains S.

2.3 Conclusion

In this chapter, we have presented an overview of typical Markov Decision Processes and their required definitions and theories at the beginning. Thus, we have introduced various structures of MDPs under uncertainty. Since MDPs do not own know reward functions, we presented some structures that produce a vector valued MDP with vector reward functions. To make MDPs scalar, we have defined a weight vector on rewards that should be predicted according to the user behaviors and expectations.

In particular, VMDP with unknown bounded polytope Λ seems a suitable framework for solving MDPs with uncertain rewards. Because it predicts the optimal policy using optimization approaches directly w.r.t polytope Λ. To increases accuracy of optimal

[10]Utah university lecture notes. http://www.cs.utah.edu/ suresh/compgeom/convexhulls.pdf

solutions using optimization algorithms, we can query a question to the user in order to get more information on reward weights and shrink the Λ polytope.

Chapter 3

A Survey of Problems and Algorithms in IRMDPs

Foreword

The problems of sequential decision-making under uncertainty have been widely studied in the recent years. In many approaches, these problems have been presented as Markov Decision Processes. In the classical setting, each action selected by the agent in a state has a stochastic outcome in MDP environment and should be assigned as a numerical value known as its reward. The MDP solution is a policy function with the highest sum of rewards and defines which action to chose in each state.

In our case, rewards are not given precisely while they have been confined in a bounded polytope. In fact, MDPs with reward vectors (VMDPs) or MDPs with imprecise rewards (IRMDPs) are suitable models for sequential decision making under uncertainty. On the other hand, if the system is aware of user preferences on uncertainty, it can transform VMDPs or IRMDPs to classical MDPs with known reward values.

Example 3.1. *For a public transport system that aims to minimize latency (i.e., the time that commuters need to reach their destinations) and pollution cost, we have an MDP model with unknown rewards. Because, reward definition considering precisely is not possible considering two objectives latency and pollution cost. Thus, the optimal solution for various users are not the same [Roijers et al., 2013]. For instance, the*

optimal solution for a user with pollution cost minimization tendency is different from a user who prefers to reduce the latency.

In order to deal with this problem, we intend to approximate a reward function concerning a given user with various types of preferences. Defining a reward function means approximating a weight vector $\bar{\lambda}$ or reward function $r \in \mathcal{R}$ respectively for VMDP or IRMDP models. After specifying the reward function, the optimal sequence can be found using classical search algorithms on MDPs. These parameters are unknown because the user can not define her preference weights on objectives as scalar values, instead she can answer preference questions in several forms:

1- comparison among the trajectories,

2- comparing several selected actions in one state or

3- comparison among the states, atomic transitions and so on.

In this chapter we are considering MDPs for which those rewards are not defined or are not available. We take a closer look at different approaches used to solve the decision making problem in uncertain dynamic environments. In section 3.1, robust methods for solving uncertain MDP problems are presented, which attempt to define the optimal policy in presence of uncertainty. Section 3.2 presents another learning method that concentrate on learning rewards by observing the expert performance. After learning the reward function, any typical algorithm on MDPs can return back the optimal policy. Finally Section 3.3 considers solutions where a user or an expert tutor interacts with the system to bring more preference information on demand.

3.1 Choosing a Robust Policy under Uncertainty

The robust solutions for MDPs with imprecisely known rewards or transition probabilities are policies that are as good as possible according to all possible values of unknown parameters. In this section we concentrate on MDPs with unknown rewards. To avoid widening the state of the art, some interesting methods for solving MDPs with unknown transition probabilities are presented in Appendix A.

In this section, we observe some studies so far in literature to solve MDPs with unknown rewards with respect to bounded set of possible rewards without modifying the imprecise reward set. They restrict the admissible set of rewards to a polytope while they approximate the optimal policy by solving a set of linear programming problems. This means, these approaches never attempt to extract more information on rewards by asking a new query to the user during the computation process.

Early methods for solving an IRMDP with polytope of unknown rewards $\mathcal{R}$ are the optimistic and pessimistic approaches [Givan et al., 2000, McMahan et al., 2003]). An example of pessimistic approach is the Maximin criterion [McMahan et al., 2003]. In the Maximin criterion a conservative policy is chosen by optimizing against the worst possible instantiation of reward function r. This is like a game between two players such that one player tries to choose a policy with maximum value, while the adversary selects the worst possible value of rewards inside $\mathcal{R}$ to minimize the first player result. The general formulation is as below:

$$\text{Maximin}(\mathcal{R}) = \max_{\pi \in \Pi} \min_{r \in R} V_r^{\pi} \tag{3.1}$$

where V_r^{π} is the expectation of sum of discounted rewards with respect to reward function $r \in \mathcal{R}$ i.e.

$$V_r^{\pi} = \mathbb{E}\{\sum_{i=0}^{\infty} \gamma^{i-1} r(s_i, a_i)\}$$

This formulation can be written on VMDPs as well:

$$\pi^* = \text{argmax}_{\pi \in \Pi} \min_{\bar{\lambda} \in \Lambda} \bar{\lambda} \cdot \bar{V}^{\pi} \tag{3.2}$$

On the other hand, Givan et al. [2000] introduce the maxmax method as an optimistic approach for computing the optimal policy.

$$\pi^* = \text{argmax}_{\pi \in \Pi} \max_{\bar{\lambda} \in \Lambda} \bar{\lambda} \cdot \bar{V}^{\pi}$$

In [McMahan et al., 2003] the problem 3.2 is solved with a Bender's decomposition approach. The model will be defined clearly in the rest of this section. The Interesting

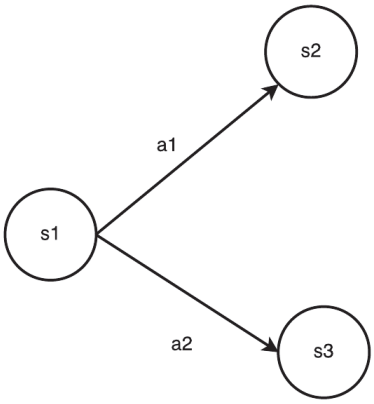

FIGURE 3.1: An MDP for comparing Minimax regret and Maximin solutions

part of their work is that, they observe a running example to study Maximin computation method. They consider a robot path planning problem where rewards are influenced by some sensors. The sensors have been placed by an adversary player in the environment (For more information refer to Section 2.2 in [McMahan et al., 2003]). The explored strategy is the best, if the sensor selector player is omniscient.

The maximin criterion is a conservative setting for finding the optimal policy of IR-MDPs. There is another approach that despite of maximin criterion, the agent does not want to miss too many opportunities and accepts some risk to gain better results. The corresponding criterion is *Minimax Regret* following [Xu and Mannor, 2009] and [Regan and Boutilier, 2009], provides more accurate policies for IRMDPs. It determines a policy with the minimum regret or loss with respect to the exact optimal policy, if the rewards had been known. In general, Minimax regret method can be implemented on MDPs with unknown rewards . Afterwards, it is possible to implement this technique on various structures such as VMDPs with an unknown polytope of weights Λ. Minimax regret definition is given precisely in Section 2.2.1.1.

To see the difference between Minimax Regret and Maximin performance, consider the following example similar to the one in [Xu and Mannor, 2009].

Example 3.2. *Figure 3.1 represents an MDP with three states and two actions and a bounded set of rewards* $\mathcal{R} = [0, 3] \times [1, 2]$. *That means* $r(s_1, a_1) \in [0, 3]$ *and* $r(s_1, a_2) \in [1, 2]$. *By implementing MaxMin method, the optimal policy proposes* a_2 *for state* s_1 *because its minimal performance is equal* 1. *On the other side, the minimax regret criterion*

find both actions a_1 and a_2 with a maximum regret equal 2. So, the optimal policy selects either action a_1 or a_2 in state s_1. If the optimal policy selects the a_1 action with p probability, it can select action a_2 with probability $1-p$.

This example illustrates how different criterion can find different policies for the same IRMDP problem. The following theorem is proven in [Xu and Mannor, 2009]:

Theorem 3.1. *Let $\mathcal{R}$ be a polytope defined by a set of $|\mathcal{R}|$ linear inequalities*[1]*. Then evaluating the minimax regret for an MDP with unknown rewards is NP-hard with respect to $|S|, |A|$ and $|\mathcal{R}|$.*

A variety of methods have been developed for computing the minimax regret value (the definition is given in 2.8). To the best of our knowledge, the majority of the methods in the literature optimize this value using series of linear programmings or mixed integer programs. Considering the equivalence between a policy and its occupancy function (see Definition 2.4 and Equation 2.12), we have:

$$V^{\pi} = \sum_{s} \sum_{a} r(s,a) f^{\pi}(s,a)$$

The most exploited solutions assume that the feasible set of rewards $\mathcal{R}$ is a convex polytope and is given by $\mathbf{A}r \leq \mathbf{b}$, where $\mathbf{A}$ is a matrix that has the same number of rows as number of bounds on rewards. And $\mathbf{b}$ is a vector of the same dimension. Thus, the minimax regret computation can be done by the following optimization problem [Boutilier et al., 2006b, Regan and Boutilier, 2009, Xu and Mannor, 2009] (see dual formulation 2.11):

$$\min_{f \in \mathcal{F}} \max_{g \in \mathcal{F}} \max_{r \in \mathcal{R}} \sum_{s \in S} \sum_{a \in A} [r(s,a) g(s,a) - r(s,a) f(s,a)] \tag{3.3}$$

This problem is a min-max quadratic program. So to solve it, we need to reformulate this problem such that it becomes more tractable. Regan and Boutilier [2009] show that it is equivalent to the following quadratic program, with a linear objective:

[1] $|\mathcal{R}|$ represents the umber of constraints restricting the $\mathcal{R}$ polytope

$$\begin{aligned} &\text{minimize}_{f\in\mathcal{F},\delta} \ \ \delta \\ &\text{subject to :} \\ &\sum_{s\in S}\sum_{a\in A}[r(s,a)g(s,a) - r(s,a)f(s,a)] \leq \delta \ \text{ for each } \ < g, r > \in \mathcal{F}\times\mathcal{R} \end{aligned} \tag{3.4}$$

This quadratic program has unfortunately an infinite number of constraints, because the set $\mathcal{F} \times \mathcal{R}$ is infinite. To cope with that, Regan and Boutilier [2009] use Bender's Decomposition [Benders, 2005]. The idea of bender's Decomposition is to replace the infinite set $\mathcal{F} \times \mathcal{R}$ by a finite set, which we will call the GEN set. Then, two problems are defined : the master problem and the sub-problem. The task of the master problem is to compute the solution to 3.4, but with the constraint $< g, r > \in$ GEN instead of $< g, r > \in \mathcal{F} \times \mathcal{R}$. The task of the sub-program is to compute the relevant (f, r) pairs to be added to the GEN. The master problem can be formulated as a linear program and the sub-problem as a quadratic program, as described below. The algorithm 3 is used to call iteratively both linear programs, in order to compute an ϵ-approximation of the min-max regret policy.

Algorithm 3 Master problem and sub-problem algorithm

$\delta \longleftarrow \infty$
GEN $\longleftarrow \emptyset$
while $\delta > \epsilon$ **do**
  $f, \delta_{\text{master}}$:= solve master-problem(GEN) ▷ minimax regret
  $< g, r >, \delta_{\text{sub}}$:= solve subproblem(f) ▷ max regret
  GEN $\longleftarrow$ GEN $\cup\{< g, r >\}$ ▷ add constraint for $< g, r >$ to master
  $\delta = \delta_{\text{sub}} - \delta_{\text{master}}$ ▷ δ_{master} = max regret - minimax regret

The algorithm is an iterative approach on two linear programs: master problem and sub-problem. The master problem — originally the problem 3.4 without constraints — starts with an empty set of constraints, namely GEN set. In each iteration, it first solves the master problem with respect to the GEN set. After, the solution of master problem f is sent to the sub-problem. The sub-problem generates the maximum violated constraints in the master problem. This constraint is added to the GEN set for the next iteration step. This process continues until the solution to the master problem and sub-problem converge.

The master problem on a GEN set of constraints is as following:

$$\text{minimize}_{f\in\mathcal{F},\delta} \quad \delta$$

subject to

$$\sum_{s\in S}\sum_{a\in A} r_i(s,a)g_i(s,a) - r_i(s,a)f(s,a) \leq \delta \quad \text{for each} \quad < g_i, r_i > \in \text{GEN}$$

And the sub-problem to find maximally violated constraints of master problem is defines as:

$$\max_{r\in\mathcal{R},g\in\mathcal{F}} \sum_s \sum_a [r(s,a)g(s,a) - r(s,a)f(s,a)]$$

The sub-problem uses the current solution f of master problem and computes the policy g and reward function r that maximizes regret of f.This sub-problem is itself a quadratic program. Recall that to solve the overall problem, we now need to iteratively solve the master problem (a linear program) and this sub-problem (which is a quadratic program). Note that the original formulation was a min-max quadratic program, so the current formulation is much more tractable. Regan and Boutilier [2009] add another reformulation step, writing this sub-problem as a mixed-integer linear program, which they solve with CPLEX.

[Xu and Mannor, 2009] introduces two other approaches for computing Minimax regret solution 3.4. One is a sub-gradient descent algorithm that iterates on a master problem and a sub-problem. This algorithm gives a solution worse than the one based on Bender's decomposition. In their second approach, they solve the curse of dimensionality of $\mathcal{R}$ polytope constraints by supposing that $\mathcal{R}$ is a convex hull such that:

$$\mathcal{R} = \text{convex-hull}\{r_1, \cdots, r_t\} = \{\sum_{i=1}^{t} c_i r_i \;\mid\; \sum_{i=1}^{t} c_i = 1;\;\; c_i \geq 0\}$$

It means the $\mathcal{R}$ polytope has a small number of vertices while each $r \in \mathcal{R}$ can be written as a linear combination of these vertices. Optimizing optimal solutions w.r.t a bounded polytope of unknown rewards does not get a precise response. Thus, we will

present some approaches to extract more information on reward sets and to yield a better approximation of the optimal policy.

3.2 Learning from Observed Behaviours

When specifying rewards is difficult, another alternative is letting an expert to demonstrate her optimal behavior and then the agent should reproduce the behavior demonstrated by the expert. It means, the system attempts to learn the optimal policy after receiving the experts' preferred trajectories or her preferred actions in any given state. This subject area is not directly related to our work, although getting a set of user's preferences and attempting to find the optimal policy can be considered as closely related to our work.

Example 3.3. *Giving driving lessons is an example where demonstrating a good driver performance is easier than assigning a reward to each couple of (state, action) of the RL model [Abbeel and Ng, 2004].*

Another interesting example of driving learning is conducting autonomous helicopter which is implemented by [Abbeel et al., 2007].

In this section, we will present two different approaches including Inverse Reinforcement Learning (IRL) and Apprenticeship Learning (AL) for solving these types of problems. Second, we will demonstrate various methods in the literature of solving IRL and AL. Although too many number of researches have been done in IRL, we are going to present some works that have been cited more than the rest.

Considering learning policies in an MDP where reward functions are not given explicitly, there exist two main approaches for learning the optimal policy:

- *Inverse Reinforcement Learning (IRL)* that finds a reward function that explain the behavior of the expert as close as possible [Abbeel and Ng, 2004, Ng and Russell, 2000, Pietquin, 2013]. Using the extracted reward function, the optimal policy can be computed with the help of classical methods in Reinforcement learning.
- *Apprenticeship learning (AL)* that learns the optimal policy directly without extracting the unknown rewards [Abbeel and Ng, 2004].

Learning from demonstrations is like learning an optimal policy with respect to a set of examples. If the set of examples are given, one obvious idea is using *Supervised Learning (SL)* techniques to find the optimal policy w.r.t the given set. In the literature, there are some regression methods implemented on similar problems for autonomous navigation [Pomerleau, 1991] and human robot interaction [Grudic and Lawrence, 1996].

Moreover, there are several algorithms in AL that use supervised learning methods to directly mimic the expert's behavior [Boularias et al., 2011, Klein et al., 2012]. As an example, Lagoudakis and Parr [2003] learn the optimal trajectory using Least Square Policy Iteration (LSPI) technique. But, the supervised learning solutions are not reliable because sometimes the gathered examples of expert's performance are not sufficient or general enough for the learning.

Precisely the IRL problem can be defied as an RL problem owning an expert's policy π^E, while the objective is to find a reward function such that its solution is almost the same as the expert's policy π^E:

1. Given parameters
   - VMDP$(S, A, p, \bar{r}, \gamma, \beta)$ with bounded polytope Λ on weight vectors
   - and expert policy π^E (preferences can be extracted from the expert's policy)
2. Goal: determine a set of possible reward weight $\bar{\lambda}^*$ such that π^E is the optimal policy for the result MDP $(S, A, p, \bar{\lambda}^* \cdot \bar{r}, \gamma, \beta)$.

We first describe the equations stating how to find a set of possible solutions for $\bar{\lambda} \in \Lambda$ producing a given policy π^E.

Let a finite state space S, a set of actions $A = \{a_1, \cdots, a_m\}$, transition probabilities p, a discount factor $\gamma \in (0, 1)$ and a policy π^E be given. Then, if $\bar{\lambda}$ is known, according to the Bellman Equation (regarding Equations 2.5 and 2.6) [Ng and Russell, 2000] we have:

$$V^{\pi^E}(s) \geq Q^{\pi^E}(s, a) \Longrightarrow \bar{\lambda} \cdot \bar{V}^{\pi^E}(s) \geq \bar{\lambda} \cdot \bar{Q}^{\pi^E}(s, a) \;\; \forall\, a \in A \backslash \pi^E(s),\; s \in S \tag{3.5}$$

One natural way to choose rewards $\bar{\lambda}$ is to first demand that it makes π^E optimal. And make any single step deviation from π^E as costly as possible. Ng and Russell [2000] choose the reward weight vector $\bar{\lambda}$ such that:

$$\max_{\bar{\lambda}\in\Lambda} \sum_{s\in S} \min_{a\in A\setminus\pi^E(s)} \left(\bar{\lambda}\cdot\bar{V}^{\pi^E}(s) - \bar{\lambda}\cdot\bar{Q}^{\pi^E}(s,a))\right) \tag{3.6}$$

It means maximizing the sum of the differences between the quality of the optimal action and the quality of the next-best action. For more information about the linear program, we invite readers to see [Ng and Russell, 2000]. After adding a reward function $r = \bar{\lambda}^* \cdot \bar{r}$ satisfying Equation 3.6 to the given MDP with imprecise reward, the solution of MDP should be the π^E policy.

If VMDP is a huge with large set of states, the idea is to maximize a summation on a subset of S like $S' \subset S$ [Ng and Russell, 2000]. Therefore, problem 3.6 should be rewritten as:

$$\max_{\bar{\lambda}\in\Lambda} \sum_{s\in S'} \min_{a\in A\setminus\pi^E(s)} \; q\left(\bar{\lambda}\cdot\bar{V}^{\pi^E}(s) - \bar{\lambda}\cdot\bar{Q}^{\pi^E}(s,a))\right) \tag{3.7}$$

where q is a penalty function with a positive constant c[2]

$$q(x) = \begin{cases} x & \text{if } x \geq 0 \\ cx & \text{if } x < 0 \end{cases}$$

It penalizes the violation of $\bar{\lambda}\cdot\bar{V}^{\pi^E} \geq \bar{\lambda}\cdot\bar{Q}(s,a)$. The c parameter reduces the sensitivity of this maxmin problem considering the number of states. It means, it gets a similar answer with moderately larger number of states [Ng and Russell, 2000].

Despite IRL, in AL, the apprentice is observing the expert behaving optimally with respect to an unknown reward function in an MDP. In this case, the environment can be modeled as a VMDP$(S, A, p, \bar{r}, \gamma, \beta)$ with unknown weight $\bar{\lambda} \in \Lambda$ while there is no expert policy π^E except the expert demonstration. The goal of the apprentice is — via some demonstrations or trajectories of the expert — to learn the expert policy or a policy which is as good as the expert policy relatively to the unknown reward directly.

Abbeel and Ng [2004] have proposed another version of IRL based on learning from trajectories. First, they require to compute an approximation of $\bar{V}_E^{\pi}$ that can be done by averaging K iterations of demonstrations of expert's performance. It means, to compute

[2] [Ng and Russell, 2000] have selected $c = 2$.

the $\bar{V}^{\pi^E}$, they demonstrate the expert performance K times. For each demonstration, they calculate its observed return value $\bar{V}$ and finally they make average on the results.

In this case, the optimal policy is computed according to the following maxmin problem:

$$\max_{\bar{\lambda}\in\Lambda}\min_{\{\pi \text{ s.t. } \bar{V}^{\pi}\neq\bar{V}^{\pi^E}\}} \bar{\lambda}\cdot(\bar{V}^{\pi^E}-\bar{V}^{\pi}) \tag{3.8}$$

This problem is similar to problem 3.6, except that $\bar{Q}^{\pi^E}(s,a)\ \forall s,a$ have been replaced by set of value functions for all policies Π. Since the expert policy π^E is not given in AL setting, its value has been compared with the rest of policies' values.

Abbeel and Ng [2004] propose an algorithm to solve this problem. They introduce a method similar to Benders Decomposition (given in 3). Except that GEN set is a subset of $\{\pi \text{ s.t. } \bar{V}^{\pi}\neq\bar{V}^{\pi^E}\}$ and the iteration continues until there is no more new policy π violating the program. Afterward, they can find the optimal policy from the approximated GEN set.

3.3 Preference Elicitation

In the previous sections, all presented methods develop exact or approximate algorithms for solving MDPs with imprecise reward values. All mentioned approaches attempt to solve the MDP with unknown rewards problem w.r.t the available polytope of feasible rewards $\mathcal{R}$. It means either they approximate the optimal policy w.r.t bounded polytope $\mathcal{R}$ or they attempt to find it based on a given set of user's preferences on trajectories. During or at the end of the process, they have no more connections with users. However, by pruning part of feasible rewards, we can get a better result using the same approximation methods. The problem of specifying more accurately rewards can be cast as preference elicitation so as to get an optimal policy as close as possible to the exact solution. In short, if there is an oracle, expert or user, who prefers a subset of $\mathcal{R}$ polytope, then we have more information about rewards and consequently can better approximate the optimal solution. Recall that for the user case, there is type of users who are uncertain about their preferences. It means, they sometimes give false responses to preference elicitation questions regarding their real preferences and their final goal. In

general, solving IRMDPs with polytope $\mathcal{R}$ using preference elicitation includes several parts:

- How to compute the optimal policy w.r.t uncertain rewards $\mathcal{R}$,
- How to generate the queries that should be proposed to the tutor,
- What type of queries we need and
- When ask the queries to the user.

This section summarizes some researches in specifying rewards by preference elicitation methods and improving optimal solution in the same time for IRMDP with unknown polytope $\mathcal{R}$ for rewards or VMDP with unknown polytope Λ for reward weights. Some of the previously described methods will be used for policy exploration w.r.t the feasible set of rewards.

3.3.1 Elicitation Based on Minimax Regret

In this section, we study some approaches in the literature and present their approaches considering reward elicitation methods [Regan and Boutilier, 2008, 2009]. First we demonstrate how the generated regret from minimax regret criterion methods can be reduced by efficiently eliciting reward information using bound queries. We will show how the queries are generated regarding minimax regret criterion.

If the minimax regret problem (explained in Section 3.1) does not have enough information about reward preferences of the user, it can give a solution considerably far from the exact optimal policy [Regan and Boutilier, 2009]. To increase accuracy of minimax regret computations, Algorithm 4 should be used to generate queries and weave the minimax regret calculation. It is an interactive algorithm between the user and the optimal policy calculator.

The query is generated by function *SelectAndAskQuerys*. This function generates the queries of the type "Is $r(s,a) \geq d$?". Regan and Boutilier [2009] have presented two main heuristic methods to make these queries regarding function *SelectAndAskQuerys*$(f, g, r, \mathcal{R}_t)$:

Algorithm 4 interactive algorithm for reward elicitation and minimax regret computation

Inputs: IRMDP$(S, A, p, \mathcal{R}, \gamma, \beta)$, ϵ
Outputs: the optimal policy π^*

mmr $\longleftarrow \infty$
$\mathcal{R}_0 \longleftarrow \mathcal{R}$
while mmr $> \epsilon$ **do**
    mmr, $f, g, r \longleftarrow$ ComputeMMR$(\mathcal{R}_t)$ ▷ It is computed from linear program 3.3
    response $\longleftarrow$ SelectAndAskQuery$(f, g, r, \mathcal{R}_t)$
    $\mathcal{R}_{t+1} \longleftarrow$ refine(response, $\mathcal{R}_t$)
$\pi^* \longleftarrow f$
Return π^*

first- *Halve Largest Gap (HLG)*: It selects the pair (s, a) as the following:

$$(s^*, a^*) = \text{argmax}_{a \in A, s \in S} \underbrace{(\max_{r' \in \mathcal{R}_t} r'(s, a) - \min_{r \in \mathcal{R}_t} r(s, a))}_{\Delta(s,a)} \tag{3.9}$$

second- *Current Solution (CS)*: It uses the minimax optimal visitation frequencies f to weight each gap. It selects the pair (s^*, a^*) such that:

$$(s^*, a^*) = \text{argmax}_{a^* \in A} \text{argmax}_{s^* \in S} \; f(s^*, a^*) \Delta(s^*, a^*) \tag{3.10}$$

Now the question is, if $r(s^*, a^*)$ is greater than the half of its gap $b = \min_{r \in \mathcal{R}_t} r(s, a) + \frac{\Delta(s^*, a^*)}{2}$ or not.

In Algorithm 4 *ComputeMMR*$(\mathcal{R}_t)$ computes the minimax regret of given IRMDP with respect to feasible polytope of rewards $\mathcal{R}$. After querying to the user and asking her opinions through function *SelectAndQuery*, the added bound on $\mathcal{R}_t$ should elicit part of the polytope using the *refine* function.

The iterative generation approach between master problem and subproblem for computing exact minimax regret is only possible for small size of MDPs. Therefore, to solve this problem, Regan and Boutilier [2010, 2011a] have done an extensive research based on using non-dominated policies (given in Definition 2.10).

3.3.2 Accelerate Minimax Regret Elicitation Method

Recalling the minimax regret formulation (given in 3.3):

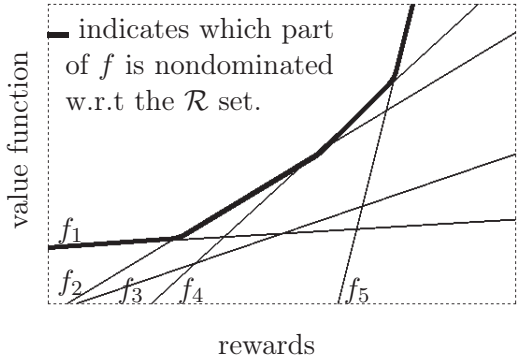

FIGURE 3.2: Illustration of value as a linear function of reward

$$\min_{f\in\mathcal{F}} \max_{g\in\mathcal{F}} \max_{r\in\mathcal{R}} \sum_{s\in S}\sum_{a\in A}[r(s,a)g(s,a) - r(s,a)f(s,a)] \tag{3.11}$$

algorithm 4 using this calculation is complex and is not applicable for even the medium size MDPs. Thus, there exist some other approaches that intend to reduce the complexity [Regan and Boutilier, 2010, 2012].

Rewriting the same definition 2.10 on occupancy functions introduce non-dominated occupancy functions as well as non-dominated policies. It means they can be used instead of each other:

Definition 3.2. An uncertain reward MDP with feasible set of reward weights $\mathcal{R}$ being given, policy f is non-dominated with respect to Λ if and only if

$$\exists\, r \in \mathcal{R} \text{ s.t. } \forall f' \in \mathcal{F} \sum_s \sum_a r(s,a)f(s,a) \geq \sum_s \sum_a r(s,a)f'(s,a)$$

It is more convenient to write non-dominated definition in the vector form:

$$\exists\, r \in \mathcal{R} \text{ s.t. } \forall f' \in \mathcal{F} \; r \cdot f \geq r \cdot f'$$

where functions r, f and f' are viewed as three $|S||A|$ dimensional vectors. The set of non-dominated policies with respect to $\mathcal{R}$ is noted as $ND_{\mathcal{R}}(\Pi)$ or equivalently $ND_{\mathcal{R}}(\mathcal{F})$ (Π / $\mathcal{F}$ are the set of all policies / occupancy functions). Using the non-dominated policies set simplifies the minimax regret computation in IRMDPs.

Observation 3.1. *For any IRMDP and policy f, $argmax_{g\in\mathcal{F}}$ Regret$(f,r) \in ND(\mathcal{F})$ [Regan and Boutilier, 2010].*

Using the vector form, we have $\text{argmax}_{g\in\mathcal{F}}\text{max}_{r\in\mathcal{R}}\ r\cdot g - r\cdot f \in ND(\mathcal{F})$. That means, the policy g that maximizes the regret of f should be inside $ND(\mathcal{F})$. With a slight abuse of notation, the set of non-dominated policies can be written as $ND(\Pi)$, if the bounded polytope of rewards $\mathcal{R}$ is known.

Example 3.4. *Figure 3.2 illustrates a set of 5 given policies on a given IRMDP. Policies are two dimensional while one dimensional unknown rewards are given on the x-axis. In this figure, $ND(\Pi) = \{f_1, f_2, f_4, f_5\}$ is the set of non-dominated policies. Policy f_3 is dominated because there is no reward (on the horizontal axis) that makes f_3 an optimal policy with respect to $\Pi\backslash f_3$. [Regan and Boutilier, 2010].*

This observation indicates that instead of solving a minimax regret problem as given in 3.11 and maximizing regrets on the whole policies $\mathcal{F}$, the problem can be solved on the set of non-dominated policies. The $ND(\mathcal{F})$ set is smaller than $\mathcal{F}$ while the dominated polices are useless for minimax regret computation. Thus, The corresponding optimization problem is:

$$\text{min}_{f\in\mathcal{F}}\ \text{max}_{g\in ND(\mathcal{F})}\text{max}_{r\in\mathcal{R}}\ \sum_{s\in S}\sum_{a\in A}[r(s,a)g(s,a) - r(s,a)f(s,a)] \tag{3.12}$$

There are some works in the literature so far that study various approaches for exploring non-dominated policies. Regan and Boutilier [2010, 2011a] introduced two important algorithms to find the set of nondominated policies:

1- Witness algorithm, inspired by POMDP methods.

2- Region Vertex algorithm, inspired by Classic linear support methods for POMDPs.

The witness algorithm is presented in Algorithm 5. It starts with a set of non-dominated policies ND (the set has a single element at the beginning). At each iteration, it checks if there is an improvement of $f \in ND$ or not. It means, a policy that is different in only one (state,action) than f, namely $f^{s:a}$ such that $f^{s:a}\cdot r > f'\cdot r$ for all $f' \in ND$. This

Algorithm 5 π witness Algorithm

Inputs: MDP$(S, A, p, \mathcal{R}, \gamma, \beta)$
Outputs: set of non-dominated policies ND(Π)

$r \longleftarrow$ an arbitrary $r \in \mathcal{R}$
$f \longleftarrow$ findBest(r)
ND $\longleftarrow \{f\}$
agenda $\longleftarrow \{f\}$
while agenda $\neq \emptyset$ **do**
    $f \longleftarrow$ next item in agenda
    for each s, a **do**
        $r^w \longleftarrow$ findWitnessReward$(f^{s:a}, ND)$
        while witness found **do**
            $f_{\text{best}} \longleftarrow$ findBest(r^w)
            add f_{best} to ND
            add f_{best} to agenda
            $r^w \longleftarrow$ findWitnessReward$(f^{s:a}, ND)$
Return ND

iteration continues until the algorithm finds no more improvement for the explored set ND.

For the sake of abbreviation, we use vector notation for the rest of this section. In the algorithm, function *findBest* finds the optimal policy for any given reward r i.e.,

$$\text{findBest}(r) = \text{argmax}_{f \in \mathcal{F}} \sum_s \sum_a f(s,a) r(s,a) = \text{argmax}_{f \in \mathcal{F}} f \cdot r$$

Suppose $f^{s:a} \in \mathcal{F}$ is an occupancy function related to policy π' such that π' policy is defined as the following w.r.t the given policy π.

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

They use occupancy functions in their calculations instead of policies directly. The reason of using this definition is that, if f (the occupancy function of policy π) is given, $f^{s:a}$ can be computed easily with few calculations. In fact *findWitnessReward*($f^{s:a}$, ND) attempts to find an r for which $f^{s:a}$ has higher value than any f inside the current explored set of non-dominated policies ND by solving the following LP:

Algorithm 6 online adjustment during elicitation

Inputs: IRMDP$(S, A, p, \mathcal{R}, \gamma, \beta)$
Outputs: optimal policy

$\mathcal{R}_0 \longleftarrow$ initial reward polytope
$\Gamma_0 \longleftarrow$ initial non-dominated policies (computing offline) ▷ like generating 100 non-dominated policies from Algorithm 5
$\epsilon \longleftarrow$ level of regret
for each step $t \geq 1$ **do** ▷ Elicitation part
    mmr, $f, g, r \longleftarrow$ ComputeMMR$(\mathcal{R}_{t-1}, \Gamma_{t-1})$
    response $\longleftarrow$ SelectAndAskQuery$(f, g, r, \mathcal{R}_{t-1})$
    $\mathcal{R}_t \longleftarrow$ Refine(response, $\mathcal{R}_{t-1}$)
    $\Gamma_{t-1} \longleftarrow$ Prune($\mathcal{R}_t, \Gamma_{t-1}$) $\cup$ Add$(\mathcal{R}_t, \Gamma_{t-1})$
    if mmr $< \epsilon$ **then**
        terminate and return minimax optimal policy f

$$\text{maximize}_{\delta,r}\ \delta$$

subject to:

$$\delta \leq f^{s:a} \cdot r - f' \cdot r \quad \forall f' \in ND$$

$$\mathbf{A}r \leq \mathbf{d}$$

The run-time of the π-Witness algorithm is polynomial in inputs $|S|, |A|, |\mathcal{R}|$[3] and output $|ND(\Pi)|$[4] [Regan and Boutilier, 2010].

Since non-dominated policies with respect to unknown reward polytope $\mathcal{R}$ include too many elements, Regan and Boutilier [2011a] propose another method for exploring non-dominated policies. This process is based on Algorithm 6. This algorithm is an iterative alternation between minimax regret computation of optimal policy and reward eliciting by proposing a query to the user.

Iteration starts with a set of nondominated policies Γ_0 as a small subset of all nondominated policies. Then, in each iteration *ComputeMMR* finds a solution for minimax regret as $\min_{f \in \mathcal{F}} \max_{r \in \mathcal{R}_t} \max_{g \in \Gamma_t} g \cdot r - f \cdot r$.

The difference between this new method and simple interactive algorithm 4 lies in the two functions *Prune* and *Add*. One major issue with Algorithm 4 is that it includes a minimax regret calculation in every iteration, which is memory and time consuming. On

[3] as mentioned in Chapter 2 $|\mathcal{R}|$ is the number of constraints on polytope $\mathcal{R}$
[4] set of all non-dominated policies

the other hand, extracting all non-dominated policies once is computationally complex (as in Algorithm 5). For these reasons, Regan and Boutilier [2011a] suggest to update the set of non-dominated policies Γ_t in each iteration. It should be updated because some policies inside Γ_t become dominated after querying the user and updating the $\mathcal{R}$ polytope.

In fact, the *Prune* function receives the updated polytope $\mathcal{R}_t$ and the old subset of non-dominated policies Γ_{t-1}, and removes new dominated policies from the old set w.r.t the new feasible set of rewards. On the other hand, new non-dminated policies are added after updating $\mathcal{R}$ and getting more information about unknown rewards, which is done by the function *Add*. Regan and Boutilier [2011b] have introduced an algorithm, namely non-dominated region vertex for finding new non-dominated policies for function add (see page 4 in [Regan and Boutilier, 2011b] for more information).

3.3.3 Reward Elicitation with Policy Iteration

Since the previous query selection methods are not precise, in this chapter we present how to ask queries systematically using a policy iteration based approach [Fürnkranz et al., 2012]. This algorithm will success, if it receives a response for all pairwise comparisons during the policy iteration process. For this reason, it should query to an oracle or a user who have an answer for all state action comparison questions.

Fürnkranz et al. [2012] introduce an approach based on the policy iteration method for MDPs [Fürnkranz et al., 2012]. According to the Policy Iteration algorithm (given in Section 2), in each iteration, it is required to find the best action in each state using:

$$\pi_{t+1}(s) = \text{argmax}_{a \in A} Q_t(s, a) \quad \forall s \tag{3.13}$$

In both approaches, the Q-value function is not computed directly. They assume that, $Q(s, a)$ induces an ordering on the actions a in each state. Algorithm 7 describes the Preference-based Approximate Policy Iteration (PAPI), introduced in [Fürnkranz et al., 2012]. The algorithm starts with a random policy π_0. In each iteration, it improves policy π by modifying its information on pairwise comparisons of (state, action) pairs

Algorithm 7 Preference-based Approximate Policy Iteration

Inputs: Sample states $S' \subset S$, initial random policy π_0, maximum number of policy iterations p, procedure

1: $\pi' \longleftarrow \pi_0$, $i \longleftarrow 0$
2: **repeat**
3: $\quad \pi \longleftarrow \pi'$, $\mathcal{E} \longleftarrow \emptyset$
4: $\quad$ **for each** $s \in S'$ **do**
5: $\quad\quad$ **for each** $(a_k, a_j) \in A \times A$ **do**
6: $\quad\quad\quad$ *EvaluatePreference*(s, a_k, a_j, π)
7: $\quad\quad\quad$ **if** $a_k \succeq a_j$ **then**
8: $\quad\quad\quad\quad \mathcal{E} \longleftarrow \mathcal{E} \cup \{a_k \succeq a_j, \pi\}$
9: $\quad \pi' \longleftarrow$ *LearnLabelRanker*$(\mathcal{E})$, $i \longleftarrow i + 1$
10: **until** *StoppingCriterion*(π, π', p, i)
11: **Return** $\bar{V}_t$

w.r.t the policy π. This iteration continues until the algorithm meets its stopping criteria based on similarity of policies.

Two functions are used in Algorithm 7: *EvaluatePreference* and *LearnLableRanker*. The combination of these two functions plays a role similar to Equation 3.13. The *EvaluatePreference*(s, a_k, a_j, π) function determines the preference between two actions a_k and a_j for a given policy π on state s. The *LearnLableRanker* function finds the policy π' regarding an explored set of preferences between actions on states produced by the *EvaluatePreference* function. It attempts to select the most preferred action for each state.

The PAPI Algorithm has been originally implemented for completely known MDPs. Fürnkranz et al. [2012] use this approach for the case of long distance trajectories while they have reward values r as well. In comparison with our problem, the PAPI approach can deal with VMDPs with given polytope Λ. This is due to the fact that *theEvaluatePreference* function does not depend on the reward values. Considering our problem, the *EvaluatePreference* function can be re-defined in two ways: it can either communicate with the user to know her preferences among state action pairs or it can approximate the preference vector $\bar{\lambda} \in \Lambda$ using the inequalities induced by the evaluated preferences.

Considering the PAPI method, there is a compact representation for the preferences of action for each state. This implies that it can be used only for solving VMDPs of small size. The algorithm complexity depends on $|S|$ and on the average number of observed preferences per state m i.e. $O(m|S|)$.

3.3.4 Reward Elicitation with Value Iteration

A problem with heuristic querying to the user is that sometimes the query could be answered without interaction, or it adds very few information to the system. In this section we introduce a *value iteration* approach on VMDPs with reward weights polytope Λ which interweaves the reward elicitation and optimization phases [Weng and Zanuttini, 2013]. They have shown that their method asks less number of queries to get the optimal policy with the same precision than other similar works. Because they introduce a new approach for asking more informative questions. This algorithm is implemented on a VMDP with bounded reward vectors Λ.

In Algorithm 8, there are d unknown rewards $\{\lambda_1, \cdots, \lambda_d\}$, and $\mathcal{K}$ denotes a set of constraints defining the polytope Λ. The initial information is that the λ_i are confined between 0 and 1 i.e. $\mathcal{K} = \{\lambda_i \geq 0,\ \lambda_i \leq 1 \ \text{ for } \ i = 1, \cdots, d\}$ at the beginning.

In order to implement the value iteration method on the VMDP, they have two separate parts: the VMDP with vector reward functions $\bar{r}$ and unknown bounded reward weights $\bar{\lambda}$s inside the polytope Λ (see Sections 2.2.3.1 and 2.2.4).

Each iteration of the value iteration method must compare two vectors, say $\bar{V}_{\text{best}}$ and $\bar{Q}(s, a)$. From the user point of view, the comparison amounts to deciding "which of $\bar{V}_{\text{best}}$ or $\bar{Q}(s, a)$ is the highest?". If the system could predict the user preferences exactly, it would assign a reward weight $\bar{\lambda}^*$ to the user. Then the question is equivalent to deciding the sign of $\bar{\lambda}^* \cdot (\bar{V}_{\text{best}} - \bar{Q}(s, a))$. But as $\bar{\lambda}^*$ is unknown, the algorithm is going to approximate its numerical value inside the Λ polytope and define the equation sign in the same time.

Since Weng and Zanuttini [2013] goal is to find the optimal policy while asking less questions to the user, they first utilize several possible comparison methods to compare any two vectors w.r.t the Λ polytope without imposing too much pressure to the users. In detail, the algorithm uses three types of vector comparison methods, 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 makes a query to the user, which refines our knowledge about user preferences and prunes the Λ polytope [Weng and Zanuttini, 2013].

Algorithm 8 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**
    $\bar{V}_{\text{best}} \longleftarrow (0, .., 0)$
    for each a **do**
      $\bar{Q}(s, a) \leftarrow \bar{r}(s, a) + \gamma \sum_{s'} p(s'|s, a)\bar{V}_{t-1}(s')$
      $(\bar{V}_{\text{best}}, \mathcal{K}) \longleftarrow$ **getBest**$(\bar{V}_{\text{best}}, \bar{Q}(s, a), \mathcal{K})$
    $\bar{V}_t(s) \leftarrow \bar{V}_{\text{best}}$
until $||\mathbb{E}_{s\sim\beta}[\bar{V}_t] - \mathbb{E}_{s\sim\beta}[\bar{V}_{t-1}]|| < \epsilon$
Return $\bar{V}_t$

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

   Definition 3.3.1. 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:

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

2. *KDominance* comparison succeeds when all $\bar{\lambda}$ of Λ verify $\bar{\lambda} \cdot \bar{V}_{\text{best}} \geq \bar{\lambda} \cdot \bar{Q}(s, a)$ or when they all verify $\bar{\lambda} \cdot \bar{Q}(s, a) \geq \bar{\lambda} \cdot \bar{V}_{\text{best}}$. It is the second least expensive test. It can be formulated as a linear program [Weng and Zanuttini, 2013]:

   $$\min_{\bar{\lambda}\in\Lambda} \bar{\lambda} \cdot (\bar{V}_{\text{best}} - \bar{Q}(s, a))$$

   $\bar{V}_{\text{best}} \succeq_K \bar{Q}(s, a)$, if the above LP has a non-negative solution.

3. Ask the query to the user: "$\bar{V}_{\text{best}} \succeq \bar{Q}(s, a)$?". This query proposes a cut w.r.t unknown reward weights $\bar{\lambda}$ as "Is $\bar{\lambda} \cdot \bar{V}_{\text{best}} \geq \bar{\lambda} \cdot \bar{Q}(s, a)$?"

The final solution is the most expensive and less desired option, because the algorithm aim is finding the optimal solution with less interactions and interruptions with the user. In fact this last option devolves answering to the user. Note that for any two vectors $\bar{V}_i$ and $\bar{V}_j$, the certain user answers the comparison question $\bar{V}_i \succeq \bar{V}_j$ all the time correctly regarding her preferences. On the other hand, the uncertain user sometimes can not give an exact answer to the query regarding her final priorities on the objectives.

Algorithm 9 getBest($\bar{V}, \bar{V}', \mathcal{K}$)

Inputs: $\bar{V}$, $\bar{V}'$, $\mathcal{K}$
Outputs: The most preferred vector between $\bar{V}$ and $\bar{V}'$ and modified constraints $\mathcal{K}$

```
if ParetoDominates(V̄, V̄′) then
    Return (V̄, K)
if ParetoDominates(V̄′, V̄) then
    Return (V̄′, K)
if KDominates(V̄, V̄′, K) then
    Return (V̄, K)
if KDominates(V̄′, V̄, K) then
    Return (V̄′, K)
(V̄best, K) ⟵ query(V̄, V̄′, K)
Return (V̄best, K)
```

Algorithm 10 query($\bar{V}, \bar{V}', \mathcal{K}$)

Inputs: $\bar{V}$, $\bar{V}'$, $\mathcal{K}$
Outputs: The most preferred vector between $\bar{V}$ and $\bar{V}'$ and modified constraints $\mathcal{K}$

```
Build query q for V̄ ⪰ V̄′
if answer to q from user is yes then
    Return (V̄, K ∪ {(V̄ − V̄′) · λ̄ ≥ 0})
else
    Return (V̄′, K ∪ {(V̄′ − V̄) · λ̄ ≥ 0})
```

Definition 3.3. $\succeq_{\bar{\lambda}*}$ is used as a comparison relation for two given vectors $\bar{V}_{\text{best}}$ and $\bar{Q}(s,a)$ with respect to the given preference vector $\bar{\lambda}*$ satisfying the user's preferences. It is defined as:

$$\bar{V}_{\text{best}} \succeq_{\bar{\lambda}^*} \bar{Q}(s,a) \Leftrightarrow \bar{\lambda}^* \cdot \bar{V}_{\text{best}} \geq \bar{\lambda}^* \cdot \bar{Q}(s,a)$$

The *getBest* function compares $\bar{V}_{\text{best}}$ and $\bar{Q}(s,a)$ according to the three listed comparison methods and modifies $\mathcal{K}$ if the method involves a new constraint on Λ. *getBest* is presented in Algorithms 9 and 10 [Weng and Zanuttini, 2013].

Since the goal is to reduce the number of queries, there is another approach that modifies Algorithm 8 by searching how and when asking the queries inside the *Interactive Value Iteration (IVI)* algorithm [Gilbert et al., 2015]. Their method does not ask the query immediately after dealing with a new one. Instead, they delay the question as much as possible. They gather queries and determine an order on them, avoiding asking unnecessary ones. Because some queries can resolve some other queries, if they are asked sooner that the others.

3.4 Conclusion

In this chapter we have presented a survey of methods for solving MDPs with unknown rewards. In the first part of this section, we have discussed several robust methods for solving MDPs with unknown rewards. Sometimes these solutions are not enough precise because the systems do not have enough information on bounded rewards. Using the robust approaches on a set of general information about rewards can not explore a precise solution.

In the second part, we studied some methods that explores the optimal policy by observing the expert behaviors. It means, they receive the experts' preferences among trajectories or actions of each state from the expert performance observation. After they approximate the reward function with respect to the given preferences. After exploring the exact reward and gaining a completely known MDP, finding the optimal policy can be done using classical algorithms in RL.

Sometimes the expert does not know which performance is the optimal one or which trajectory is the best one. But she can compare trajectories with each other and signals back the system which one is preferred to another one in order to do a task in the perfect way. Thus, another existed methods in the literature attempt to discover the optimal policy by observing expert performance without approximating the reward function. They compute the optimal policies directly from a set of comparisons among trajectories.

There are many useless questions or comparisons in each system for exploring the optimal policy. Therefore, another approach is to approximate the optimal policy while asking necessary comparison questions during the process. These methods are effective, because they can manage type of queries, number of asked queries to the user and complexity of computations. In the final section, we have presented some reward elicitation methods so far in the literature and we have explained how do they deal with this problem.

Chapter 4

Elicitation Methods with Iteration Based Approaches

Forward

Vector-valued MDPs (VMDPs) offer a rich framework to optimize the optimal strategy in dynamic environments under uncertainty. This setting allows us to measure rewards as vectors, while each reward element has a weight regarding user priorities. Since user preferences are not known, we can assign a vector to any policy in the environment by adding expectation of sum of rewards. Thus, each policy is assigned with its vector value function. MDP solution is the policy with the maximum value among all existing policies in the model, while in the VMDP case we should find the policy with maximum value vector. Since comparing value vectors does not have any solution in several situations, it is needed to communicate with the user and query her about her preferences among several policies. These informative queries reveal more information on reward weights and elicit useless information on set of unknown reward weights. Thus, there are two main aspects that need to be considered when designing an algorithm for this setting: its computational complexity and the number of interactions with the user generated by such an algorithm.

In this Chapter we present two new techniques that allow to reduce the number of queries to the users. In Section 4.4 we present an algorithm that compile all information in a

small set of non-dominated policies to find out the optimal one from that small set by querying the user.

4.1 VMDP General Properties

In the last paragraph of Chapter 2 we briefly illustrated how VMDP problems can be divided into two separate polytopes. Assume a VMDP(S, A, p, $\bar{r}$, γ, β) with admissible polytope Λ is given. If the reward vectors are d-dimensional vectors, then the reward weight vectors $\bar{\lambda} \in \Lambda$ have also dimension d. This means that the reward vectors $\bar{r}$s and the rewards weight vectors $\bar{\lambda}$ are both located in $\mathbb{R}^d$. Without loss of generality, we assume that the Λ polytope is inside a d-dimensional unit cube:

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

This polytope represents all the knowledge so far about user's preferences. In Figure 4.1 we provide an example of a VMDP with 2 objectives, namely objective1 and objective2. The system supposes that all vector preferences are inside the given square.

For instance, a user with vector preference $\bar{\lambda} = (1, 0)$ is the one who prefers objective1 than objective2 in all situations.

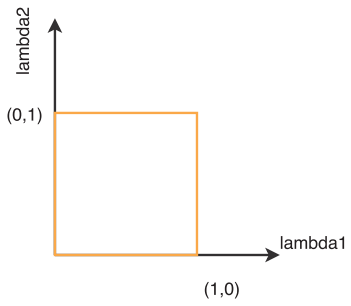

FIGURE 4.1: An example of polytope Λ for VMDP with reward vectors of dimension 2 ($d = 2$).

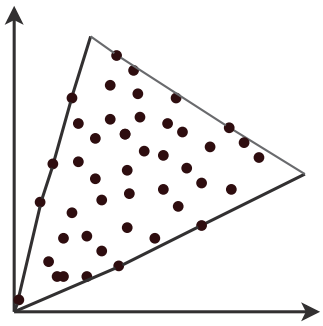

FIGURE 4.2: An example of $\bar{\mathcal{V}}$ polytope including vector-valued functions for VMDPs with two objectives.

For a given VMDP (or MDP in general), we indicate by Π the set of possible policies, with $|\Pi| = |A|^{|S|}$ We know that for any policy $\pi \in \Pi$, there exists a vector-valued

function that can be calculated from Equation 2.16 (For more details we refer the reader to Section 2.2.2).

The vector-valued functions related to the Π members, induce the following set: $\{\bar{V}_\beta^\pi : \pi \in \Pi\}$. Since the vector-valued functions $\bar{V}_\beta^\pi$ can not be represented compactly, we define a polytope $\bar{\mathcal{V}}$ including vector-valued functions related to the members of Π members. Figure 4.2 presents an example of the $\bar{\mathcal{V}}$ polytope, including some vector-valued functions. The $\bar{\mathcal{V}}$ polytope alone has all the information needed to evaluate the best policy associated to any preference vector. In other words, the VMDP can be replaced by the $\bar{\mathcal{V}}$ polytope.

Suppose $\bar{\mathcal{V}}$ and Λ are given. Our aim is to explore the optimal value function as: $\bar{\lambda}^* \cdot \bar{V}_\beta^*$ such that $\bar{V}_\beta^* \in \bar{\mathcal{V}}$ is the optimal vector-valued function for the VMDP (given in 2.19), and $\bar{\lambda}^* \in \Lambda$ is the preference vector. To tackle this problem, several types of questions must be handled:

- How to compute the $\bar{\mathcal{V}}$ members.
- Which members of $\bar{\mathcal{V}}$ are more effective to be explored.
- How to approximate $\bar{\lambda}^* \in \Lambda$ satisfying the user preferences.
- Which vector pairs from $\bar{\mathcal{V}}$ should be compared with each other for exploring $\bar{\lambda}^*$ such that it is close enough to the user preferences.

The last option is especially important because it produces the queries that should be proposed to the user. More precisely, comparing any two vectors $\bar{V}_\beta^{\pi_i}$ and $\bar{V}_\beta^{\pi_j}$ inside $\bar{\mathcal{V}}$ creates a cut for the polytope Λ. The user opinion regarding this comparison prunes part of the Λ polytope.

4.2 Comparing Policies Produces Cuts on Polytope Λ

Comparing two vectors $\bar{V}_\beta^{\pi_i}$, and $\bar{V}_\beta^{\pi_j}$ means deciding "which of $\bar{\lambda}^*.\bar{V}_\beta^{\pi_i}$ or $\bar{\lambda}^*.\bar{V}_\beta^{\pi_j}$ is the highest?". Since the user preferences are not given, $\bar{\lambda}^*$ is unknown and hence it could be not possible to compare the two vectors, unless a query is asked to the user. Comparing two vectors produces the following hyper-plane:

$$\pi_i \text{ is prefered to } \pi_j \Rightarrow \bar{\lambda}^* \cdot \bar{V}_\beta^{\pi_i} \geq \bar{\lambda}^* \cdot \bar{V}_\beta^{\pi_j} \tag{4.1}$$

If the hyper plane does not intersect the Λ polytope, an interaction with the user is not required. Otherwise, we should find a response for the comparison question with the procedure presented in Algorithm 9, if neither the Pareto dominance or the KDominance are able to provide an answer, a query to the user is required to decide which side of the hyper-plane should be kept. In fact, each query generates a cuts on the Λ polytope where all $\bar{\lambda}$ such that $\bar{\lambda} \cdot (\bar{V}_\beta^{\pi_i} - \bar{V}_\beta^{\pi_j}) \leq 0$ will be discarded (because they correspond to region of the polytope that represents not preferred parts), decreasing the volume of the Λ polytope.

The types of queries, when to perform a query and which queries should be proposed to the users are three important issues. In the following sections we will introduce new approaches that allows to better answer to the listed questions.

4.3 Advantages

According to the basic definition of advantages given in Definition 2.2, we introduce the vector advantages on VMDPs and we study some of their useful characteristics and properties.

Definition 4.1. A VMDP with a finite set of states S and actions A, a vector reward function $\bar{r}$ and the initial distribution on states β is given. Vector-valued advantage of a policy π, namely vector *Advantage* with respect to the VMDP on state s and action a is defined as follows:

$$\bar{A}^\pi(s,a) = \bar{Q}^\pi(s,a) - \bar{V}^\pi(s) \tag{4.2}$$

Similarly, by taking the initial distributions on state s into account, the *weighted advantage* vector is defined as

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

Referring to the policy iteration Algorithm 2 and to the policy improvement theorem (given in Equation 2.6), to find the optimal policy for each MDP we are interested in improving a policy π to the policy π' such that: $\pi'(s) \in \text{argmax}_{a \in A} Q^{\pi}(s,a)$.

Since in VMDPs with bounded polytope Λ, the preference vector $\bar{\lambda}$ is not given, we are looking for approaches that allow to improve the policies regardless of the $\bar{\lambda}$ vectors.

In the VMDP setting, the value function V and the Q-value function will be considered as vector functions. Considering Equation 2.21, if preference vector $\bar{\lambda}$ is known, we have:

$$\pi'(s) \in \text{argmax}_{a \in A} \bar{\lambda} \cdot \bar{Q}^{\pi}(s,a) \tag{4.3}$$

Definition 4.2. For a policy π, $\pi^{\hat{s}\uparrow\hat{a}}$ represents the same policy as π, except from the fact that it chooses the action $\hat{a}$ in state $\hat{s}$ instead of choosing the action $\pi(\hat{s})$ following policy π [Baird, 1993, Kakade and Langford, 2002]:

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

Assuming that for a given policy π, $\pi^{\hat{s}\uparrow\hat{a}}(s)$ has an higher value than policy π. Since π and $\pi^{\hat{s}\uparrow\hat{a}}$ policies only differ in the $\hat{s}$ state, we have:

$$\bar{\lambda} \cdot \bar{Q}^{\pi}(\hat{s}, \pi^{\hat{s}\uparrow\hat{a}}(\hat{s})) = \bar{\lambda} \cdot \bar{V}^{\pi^{\hat{s}\uparrow\hat{a}}}(\hat{s}) \geq \bar{\lambda} \cdot \bar{V}^{\pi}(\hat{s}) \tag{4.4}$$

In the light of Equation 4.4, we analyze the advantage of $\pi^{\hat{s}\uparrow\hat{a}}$ over π by taking the initial states distribution β into account, *i.e.* the difference of vectors $\bar{V}_{\beta}^{\pi^{\hat{s}\uparrow\hat{a}}}$ and $\bar{V}_{\beta}^{\pi}$. Since the only difference between π and $\pi^{\hat{s}\uparrow\hat{a}}$ is the state $\hat{s}$, we have:

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

This reminds us that we can explore new policies with more than one advantage vector differences from policy π. Let $\pi'' = \pi^{\hat{s}_1\uparrow\hat{a}_1, \cdots, \hat{s}_k\uparrow\hat{a}_k}$ be the policy that 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 $\bar{Q}^{\pi}(\hat{s}_i, \hat{a}_i) \geq \bar{V}^{\pi}(\hat{s}_i)$, then we have:

$$\bar{\lambda} \cdot \bar{V}_{\beta}^{\pi''} \geq \bar{\lambda} \cdot \bar{V}_{\beta}^{\pi^{\hat{s}_1 \uparrow \hat{a}_1}} \geq \bar{\lambda} \cdot \bar{V}_{\beta}^{\pi}$$
$$\bar{\lambda} \cdot \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi''}(s)] \geq \bar{\lambda} \cdot \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi^{\hat{s}_1 \uparrow \hat{a}_1}}(s)] \geq \bar{\lambda} \cdot \mathbb{E}_{s\sim\beta}[\bar{V}^{\pi}(s)]$$

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

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

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

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

Therefore, for an arbitrarily selected policy π, we are interested in finding a large values of $\bar{\lambda} \cdot \sum \bar{A}_{\beta}^{\pi}(s, a)$ w.r.t $\bar{\lambda} \in \Lambda$. Based on this objective, we concentrate on the advantage sets $\mathcal{A} = \{\bar{A}_{\beta}^{\pi}(s, a) | s \in S, a \in A\}$ and their characterizations.

4.4 Propagation-Search Algorithm for VMDPs

The method proposed in this section for finding the optimal strategies uses extensively the two following independent polytopes: the set of admissible vector-valued functions $\bar{\mathcal{V}}$ and the Λ polytope of admissible reward weight vectors (Explained in Section 4.1). In contrast with the majority of the existing algorithms for reward-uncertain MDPs, our approach does not require interactions with the user during the optimal policies generation.

Any user's optimal policy is a non-dominated policy (the definition is given in 2.10 and they will be explained in detail in next section). Thus, since computing the exact $\bar{\mathcal{V}}$ polytope is not tractable, we approximate a subset of its non-dominated vectors $\bar{\mathcal{V}}_{\epsilon}$ with ϵ precision. Once we build the set of non-dominated policies, we do not need the

VMDPs anymore and this allows us to speed up the optimal policy search. Afterward, it is sufficient to find the optimal policy complying the given user from the $\bar{\mathcal{V}}_\epsilon$ set by querying comparison questions to the user.

Recall that a polytope has two presentations, the H-representation (including set of hyper-planes) and V-representation (including set of vertices). We build an approximated set of non-dominated policies $\bar{\mathcal{V}}_\epsilon$ that is a V-representation of the $\bar{\mathcal{V}}$.

Thus, to solve a VMDP problem:

1- we first propose an algorithm for discovering the approximated set of non-dominated policies with precision ϵ, namely *Propagation Algorithm*,

2- second, we introduce an approach for searching the (nearly) optimal policy according to the user priorities from the set of approximated non-dominated policies, in parallel with interactively narrowing the Λ polytope. This algorithm is called the *Search Algorithm*.

$\bar{\mathcal{V}}$ is built with the help of classification methods on advantages adapted to VMDPs (see Section 4.3). The idea is to generate vector-valued functions first by generating a tree of advantages. Since the advantage tree expansion grows exponentially, we will reduce tree size by clustering advantages in each level (see Section 4.4.1). Although clustering advantages reduce the computation complexity, this method is still computationally heavy. Therefore, by defining a precision ϵ, we reduce the calculation complexity and we produce an ϵ approximate subset of non-dominated vectors, namely $\bar{\mathcal{V}}_\epsilon$ (see Section 4.4.2).

After approximately knowing the set of non-dominated $\bar{V}_\beta$ vectors, the next point is to find the optimal $\bar{V}^*_\beta$ according to the user preferences. Our approach allows the interaction with the user in order to compare two $\bar{V}_\beta$ vectors and find the optimal one according to the user priorities.

Regan and Boutilier [2010] propose an exact method, the π-witness algorithm, for generating non-dominated policies. The algorithm attempts to explore all non-dominated policies by solving thea very elevated number of linear programming problems. This method is a complex method and it is based on the minimax regret computation idea. However, Since they are forced to generate all non-dominated policies, they can not

define when to stop the generation of non-dominated to have a according to a desired precision for their optimal policy response.

Another work of [Regan and Boutilier, 2011b] is an online generation method that queries the user during the generation process. It allows to shrink the Λ polytope, and it reduces the non-dominated policies search complexity. They then use minimax regret computation method on the non-dominated policies to explore the optimal one. The second part of our algorithm (search) is easier than theirs method because it compares policies pairwise, instead of using minimax regret calculation. We believe that using the idea of leveraging the user preferences during non-dominated policies generation can improve our propagation search method. We will implement this idea in our algorithm in the future.

4.4.1 Describing $\bar{\mathcal{V}}$ Members Using Advantages

Following the non-dominated policy definition 2.10, a non-dominated vector-valued function with respect to Λ is defined:

Definition 4.3. A VMDP with a feasible set of weights Λ being given, $\bar{V}_\beta \in \bar{\mathcal{V}}$ is non-dominated (in $\bar{\mathcal{V}}$ w.r.t Λ) if and only if:

$$\exists\, \bar{\lambda} \in \Lambda \;\; s.t. \;\; \forall\, \bar{U}_\beta \in \bar{\mathcal{V}}, \;\; \bar{\lambda} \cdot \bar{V}_\beta \geq \bar{\lambda} \cdot \bar{U}_\beta$$

Suppose that $\bar{\mathcal{V}}$ is given while we are looking for the optimal vector $\bar{V}^*_\beta$ satisfying a user with preference vector $\bar{\lambda}^*$. According to the latter definition, this vector is a non-dominated vector-valued function *i.e.* $\bar{V}^*_\beta \in \mathrm{ND}(\bar{\mathcal{V}})$. This means that we should find an approximation of non-dominated vector-valued functions.

Since each $\bar{V}_\beta \in \bar{\mathcal{V}}$ should be computed for a selected policy $\pi \in \Pi$, our idea is to generate (some of) the $\bar{V}_\beta$ vectors using advantages. According to Section 4.3, for each $\bar{V}^\pi_\beta$ vector, there exist $|S||A|$ advantages $\{\bar{A}_\beta(s_1, a_1), \bar{A}_\beta(s_1, a_2), \cdots, \bar{A}_\beta(s_{|S|}, a_{|A|})\}$. This implies that if we start with a given vector and we continue the exploration for each new generated $\bar{V}^\pi_\beta + \bar{A}_\beta(s_i, a_j)$, the total number of generated vectors increases exponentially after few layers of advantage generation.

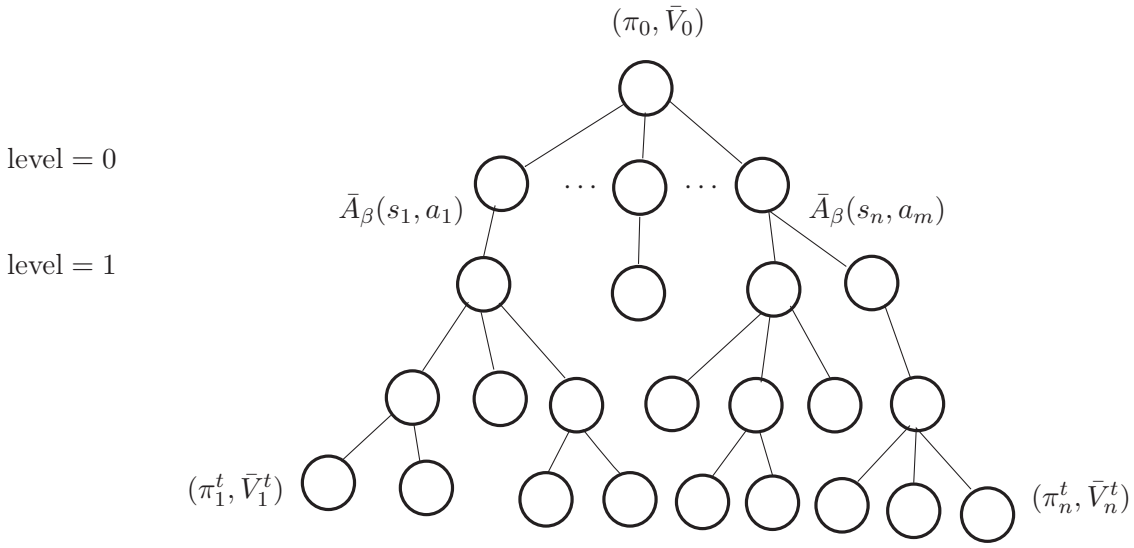

FIGURE 4.3: Structure tree of $\bar{\mathcal{V}}$ exploration.

In order to reduce the exponential growth of $\bar{V}_\beta$ vectors generation while keeping non-dominated vectors, our new algorithm suggests grouping several advantages in each iteration. In fact, in place of adding each $\bar{A}_\beta(s, a)$ to the $\bar{V}_\beta^\pi$ vector, we group several advantages and make a new equivalent vector from them. We implement that strategy with the help of the same clustering method on advantages explained in Section 4.5.

Classifying the advantages allows us to decrease the number of new explored vectors. If a cluster c includes k advantages $\bar{A}_\beta(s_{i_1}, a_{i_1}), .., \bar{A}_\beta(s_{i_k}, a_{i_k})$, the new explored vector becomes $\bar{V}_\beta^{\pi'} = \bar{V}_\beta^\pi + \sum_{j=1}^{k} \bar{A}_\beta(s_{i_j}, a_{i_j})$[1]. Classifying the advantages in $|C|$ clusters enables us to generate less $\bar{V}_\beta$ vectors and more useful (policy, vector-valued function) pairs.

Figure 4.3 illustrates a tree with a root including policy π_0 and its related vector-valued function $\bar{V}_0$. Taking $\bar{V}_0$ and computing $\bar{Q}$-function produce $|S||A|$ advantages. Without clustering on the advantages, the root will have $|S||A|$ children nodes in the first expansion level while the total number of nodes increase exponentially by expanding the tree levels. Clustering the advantages allows us to have significantly less nodes in the advantage tree, while keeping only the more effective and useful nodes. If we cluster the generated advantages at each node, we can still assign a pair $(\pi_i^t, \bar{V}_i^t)$ to each node. π_i^t and $\bar{V}_i^t$ are computed according to the latter explanations after classifying the advantages. We can say that an effective classification of each iteration (i.e. level of tree) is the

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

one including policies (i.e. nodes) with great vector values with approximately the similar directions. Because the assigned vector to a class of vectors with different directions may have a smaller size in comparison with the other classes of the same level. In the following section we will discuss how to generate an approximate set of non-dominated vector-valued functions using a clustered advantages tree.

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

Even if, after the clustering, the number of generated children in each level decreases, the total number of nodes in the tree remains exponential. To further reduce this size we use two methods: 1) we work with the convex hull of the generated vector-valued functions after each extension level and 2) we introduce a stopping criteria dependent on a given precision ϵ. The advantage of working at each step withconvex-hull is to reduce the total number of vectors generated in each step.

More precisely, this method allows us to keep only the greatest vector among several vectors with the same direction after each tree expansion. Therefore, appending the advantages will converge faster to the optimal vectors in the following levels. Moreover, the ϵ parameter allows us to limit the number of extended vectors at the end of the propagation setting.

This section introduces an algorithm to generate an approximation of the non-dominated vector-valued functions set $\bar{\mathcal{V}}_\epsilon$. This approximation at ϵ precision is a subset of $\bar{\mathcal{V}}$ such that

$$\forall \bar{V}_\beta \in \bar{\mathcal{V}} \;\; \exists\, \bar{V}'_\beta \in \bar{\mathcal{V}}_\epsilon \;\text{ s.t. }\; ||\bar{V}'_\beta - \bar{V}_\beta|| \leq \epsilon \tag{4.5}$$

Recall that the $\bar{\mathcal{V}}_\epsilon$ set is an approximated set of non-dominated vector-valued functions including all possible optimal policies for different users. For the sake of simplicity we do not write the β index of $\bar{V}$ vectors in the rest of this chapter. Vector $\bar{V}$ is a d-dimensional vector representing a policy with initial state distribution β; i.e. $\bar{V}_\beta$.

better explain the propagation algorithm we introduce a tree structure (see Figure 4.3). Each node n in tree is indicated with a pair (π, $\bar{V}$) of a policy π and its related vector-valued function $\bar{V}$. In order to explore a set of non-dominated vectors first the tree is initialized with a random policy $\pi_0 \in \Pi$ and a d-dimensional $\bar{V}_0$. The $\bar{V}_0$ vector is the

vector-valued function of policy π_0 that can be calculated using classical methods such as value iteration. Before introducing our main algorithm (Algorithm 12), we present some preliminary functions and algorithms.

Essential to the algorithm is the function *expand*, described in Algorithm 11. 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 of advantages $\mathcal{A}$. 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 $\bar{V}_{s,a} = \bar{V} + \bar{A}_\beta(s,a)$. Then the algorithm calls the *Cluster-Advantages* function. This function takes as input a node n and a set of advantages $\mathcal{A}$. the Node n includes a policy π and its related vector-valued function $\bar{V}$. The function first classifies the advantages set $\mathcal{A}$ and produces a set of clusters $C = \{c_1, .., c_k\}$. Then, it returns back the set of n node's children in the tree. This set is indicated with N and is defined as follows:

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

If several $\bar{A}_\beta(s,a) \in c_j$ are associated to the same state s, we randomly select one of them to produce the policy π_j. Afterwards, $\bar{V}_j$ considers only the selected advantage from several advantages with the same sate s.

Algorithm 11 expand: Expand Children for given node n

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

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

In spite of classification, adding all the nodes generated by the *Expand* function remains exponential with respect to time and space. For this reason, we look for an approach that avoids expanding unnecessary nodes of the search tree and rolls up more non-dominated $\bar{V}$s of $\bar{\mathcal{V}}$.

Algorithm 12 Propagation: Generate approximation of non-dominated vectors $\bar{\mathcal{V}}_\epsilon$ using Advantages for a given VMDP

Inputs: VMDP$(S, A, p, \bar{r}, \gamma, \beta)$, $\{(\pi_i^0, \bar{V}_i^0)\}_{0 \leq i \leq d}$[2], $\mathcal{K}$ set of constraints of polytope Λ, ϵ
Outputs: set $\bar{\mathcal{V}}_\epsilon$

$N \longleftarrow \{(\pi_0^0, \bar{V}_0^0), .., (\pi_d^0, \bar{V}_d^0)\}$
$\bar{\mathcal{V}}^{\text{old}} \longleftarrow$ **ConvexHull**(**getVectors**(N))
do
    $N \longleftarrow \{\}$
    $\mathcal{C} \longleftarrow \{\}$
    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**$(N))$
    $\bar{\mathcal{V}}^{\text{old}} \longleftarrow N$
while $\bar{\mathcal{V}}^{\text{new}} \neq \bar{\mathcal{V}}^{\text{old}}$
return $\bar{\mathcal{V}}^{\text{new}}$

If we use the *expand* function on each generated node in each iteration, the clustering on advantages does not reduce enough the size of the tree. Because of that, we compute the convex hull[3] of the set of generated nodes from one level to another. Suppose that, in the t-th step of the node expansion, we have the set of the n generated nodes in $N_t = \{(\pi_1^t, \bar{V}_1^t), \cdots, (\pi_n^t, \bar{V}_n^t)\}$ and the set $\bar{\mathcal{V}}_t$, containing the vector-valued functions of the N_t elements, i.e. $\bar{\mathcal{V}}_t = \{\bar{V}_1^t, .., \bar{V}_n^t\}$. The *expand* function on N_t will produce a new set of nodes. In fact, due to the convexity of $\bar{\mathcal{V}}$, the vertices of the convex hull of $\bar{\mathcal{V}}_t$ are enough to describe $\bar{\mathcal{V}}$. This means that to build N_{t+1} from the given set of nodes N_t, we can compute the union of N_t and its expanded children. In other words, only the children nodes that have a vector-valued function coordinates outside the convex hull of the father node are useful, all the other nodes can be discarded.

For instance, in Figure 4.4, the square points represent the vectors obtained after the t-th iteration and the dashed polygon represents their convex hull. The red round points are the expanded nodes obtained after applying the *Expand* function on the points of the t-th iteration. The polygon with straight lines represents the convex hull at the $t+1$-th iteration and $\bar{\mathcal{V}}_{t+1}$ contains only the vertices of this polygon.

Two remarks show that this strategy does not prune optimal points. First, the fact that an optimal solution of a linear function over a polytope is always attained on one of its

[3] the convex hull definition is given in Definition 2.12

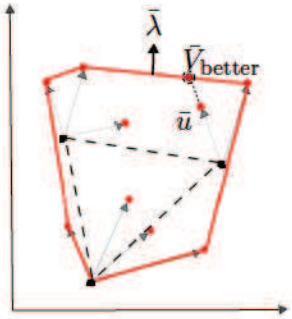

FIGURE 4.4: $\bar{\mathcal{V}}_{t+1}$ vectors selection after tree extension of $\bar{\mathcal{V}}_t$

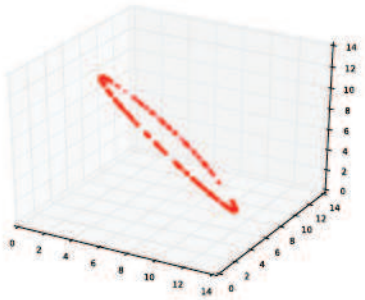

FIGURE 4.5: Generated non-dominated vectors for an MDP with 128 states, 5 actions, $d = 3$ and $\epsilon = 0.01$ (Algorithm 12)

vertices. 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 $\bar{\mathcal{V}}_{t+1}$. It shows that the proposed strategy does not prune any optimal point.

Using the previous ideas and observations, we propose to use Algorithm 12 to identify $\bar{\mathcal{V}}_\epsilon$, a non-dominated subset of $\bar{\mathcal{V}}$. This algorithm uses four main functions: *ConvexHull*, *CheckImprove*, *expand* and *getVectors*.

ConvexHull gets a set of d-dimensional vectors, and generates its convex hull. This function returns the vertices of the convex hull. Function *expand* has just been described (Algorithm 11). *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 ϵ precision. Its return value is defined below:

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

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

Algorithm 12 inputs a VMDP, a set of d initial nodes, a stopping threshold ϵ and a set of linear constraints representing the Λ polytope. Since the given Λ polytope is a unit hyper-cube in d dimensional space, we select d unit vectors in $\mathbb{R}^d$ as preference vectors λ for the extreme users. 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. This yields d scalar MDP's where the VI method [Sutton and Barto, 1998] can discover the optimal policy and related vector-valued function $\bar{V}_i$ for any $1 \leq i \leq d$. The goal of using d initial nodes related to the d extreme users is to explore optimal policies in d various directions and to speed up the algorithm convergence.

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.

Applicability of Algorithm12 depends on the size of the studied MDP. Practically speaking, it is useful for IRMDPs with average dimension d. For MDPs with high dimension of d, the number of non-dominated vector-valued functions becomes too elevated and exploring all these points becomes too time and memory consuming. Figure 4.5 gives an example of the vectors generated by Algorithm 12 for an MDP with $d = 3$, 128 states and an average precision $\epsilon = 0.01$. In this figure every point is an explored non-dominated $\bar{V}$ vector. It depicts how this algorithm generates considerable number of non-dominated vectors for the small dimension $d = 3$.

4.4.3 Searching Optimal V^* by Interaction with User

After discovering (offline and regardless of the user preferences) an approximated set of all possible optimal $\bar{V}$ vectors for a given VMDP, 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 the vector $\bar{\lambda}^* \in \Lambda$ embedding the 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 while $\mathcal{K}$ is its set of constraints. As detailed in Section 2.2.2, 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 user. Recall that $\succeq_P$ indicates the comparison regarding the Pareto dominance, $\succeq_K$ verifies only the Kdominance comparison and $\succeq$ is the user's preference.

Algorithm 13 Half-Λ-Query: Select a pair from a set of ambiguous pairs T such that their concluded cut divides the Λ polytope almost in half.

Inputs: T a set of not comparable pairs using Pareto nor Kdominance comparisons, $\mathcal{K}$ set of constraints of polytope Λ
Outputs: selected pair $(\bar{V}_i, \bar{V}_j)$ that should be proposed to the user.

for $(\bar{V}_i, \bar{V}_j)$ in T **do**
  $c_{i,j} = c_{j,i} = 0$
  repeat
    choose random $\bar{\lambda} \in \Lambda(\mathcal{K})$
    for $(\bar{V}_i, \bar{V}_j)$ in T **do**
      if $\bar{\lambda} \cdot \bar{V}_i > \bar{\lambda} \cdot \bar{V}_j$ **then**
        $c_{i,j} = c_{i,j} + 1$
  until 1000 times
$(\bar{V}_i, \bar{V}_j) = \text{argmin}_{(\bar{V}_i, \bar{V}_j) \in T}(|c_{i,j} - 500|)$
return $(\bar{V}_i, \bar{V}_j)$

Algorithm 16 is proposed as an approach for finding the optimal $\bar{V}^*$ from an approximated set of non-dominated vector-valued functions $\bar{\mathcal{V}}_\epsilon$ according to the user preferences. The main characteristic of this algorithm is that,$\bar{V}^*$ (and consequently the optimal policy π^*) is explored by querying few number of pairs $(\bar{V}_i, \bar{V}_j) \in \bar{\mathcal{V}}_\epsilon$ to the user.

Before presenting the main algorithm of this section, we are going to introduce some algorithms, functions and notations. Let T be the set of ambiguous pairs (To be determined pairs). Here, a pair of vectors is ambiguous if it is neither comparable by Pareto nor by Kdominance comparisons. D is the set of compared pairs (Determined pairs). Each pair $(\bar{V}_i, \bar{V}_j)$ inside the determined set D has a label. The label is defined as below:

$$\text{label}(\bar{V}_i, \bar{V}_j) = \begin{cases} 1 & \text{if } \bar{V}_i \succeq_P \bar{V}_j \text{ or } \bar{V}_i \succeq_D \bar{V}_j \text{ or } \bar{V}_i \succeq \bar{V}_j \\ -1 & \text{if } \bar{V}_i \prec_P \bar{V}_j \text{ or } \bar{V}_i \prec_D \bar{V}_j \text{ or } \bar{V}_i \prec \bar{V}_j \end{cases}$$

Algorithm 13 (*Halve-Λ-Query*) gets a set of ambiguous pairs T and bounded reward weights polytope Λ. It searches for the pair $(\bar{V}_i, \bar{V}_j) \in T$ which would cut the Λ polytope in half as much as possible. This query is the most informative one because despite of user's response, it will eliminate half of the Λ polytope. For each pair of T, the algorithm randomly selects 1000 values $\bar{\lambda}$ inside the Λ polytope, and counts how many of these tests prefer $\bar{V}_i$ or $\bar{V}_j$. The random $\bar{\lambda}$ points are generated as a linear combination of Λ polytope vertices [4].

In fact, the algorithm relies on a Monte-Carlo method to estimate the ambiguity of a pair $(\bar{V}_i, \bar{V}_j)$ with respect to the given Λ [Gilbert et al., 2015]. It returns $(\bar{V}_i, \bar{V}_j) \in T$ such that $Pr_{\bar{\lambda}\sim\Lambda}(\bar{V}_i \succeq_{\bar{\lambda}} \bar{V}_j)$ is as close to $\frac{1}{2}$ as possible[5].

Algorithm 14 RemovePair: takes pair of not-ambiguous vectors $(\bar{V}_j, \bar{V}_i)$ and updates To be determined set T and the determined set D.

Inputs: $\bar{V}_i, \bar{V}_j, T, D$
Outputs: T, D

1: remove $(\bar{V}_i, \bar{V}_j)$ and $(\bar{V}_j, \bar{V}_i)$ from T
2: add $(\bar{V}_i, \bar{V}_j)$ to D
3: **return** T, D

Algorithm 15 RemoveDominatedPairs: Updates two sets T and D by removing comparable pairs from ambiguous set T and adding them to the determined set D by defining their labels.

Inputs: T, D
Outputs: T, D

1: **for** $(\bar{V}_i, \bar{V}_j)$ in T **do**
2: **if** $\bar{V}_i \succeq_P \bar{V}_j$ or $\bar{V}_i \succeq_K \bar{V}_j$ **then**
3: $T, D \longleftarrow$ **RemovePair**$(\bar{V}_i, \bar{V}_j, T, D)$
4: **return** T, D

The two algorithms 14 and 15 are used for updating the T and the D set. *RemovePair* receives a pair of comparable pairs and updates the two sets according to this new comparable pair. *RemoveDominatedPairs* updates the same sets T and D by removing comparable pairs. Note that we do not communicate with users in this algorithm, so we can test comparable pairs just by applying the Pareto dominance or the KDominance methods. In both algorithms, each appended pair $(\bar{V}_i, \bar{V}_j)$ to the D set should be marked with its label.

[4] Each pair comparison $(\bar{V}_i, \bar{V}_j)$ generates a cut that passes through the origin on the Λ polytope. So, this type of random point generation does not make any problem for our method. There are more reliable method such as Hit and run method [Ya and Kane, 2015] that will be considered in our future work.

[5] $\succeq_{\bar{\lambda}}$ has been defined in Definition 3.3

Function *Query* receives a pair of vectors $\bar{V}_i$, $\bar{V}_j$ and a set of constraints $\mathcal{K}$ on Λ polytope. This function proposes the query like "is $\bar{V}_i \succeq \bar{V}_j$?" to the user and appends a new cut to set $\mathcal{K}$ regarding the user preferences while assigning a label to this pair.

Algorithm 16 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: $\bar{V}^*$

$T \longleftarrow \{(\bar{V}_i, \bar{V}_j) \in \bar{\mathcal{V}}_\epsilon^2 | i < j\}$
$D \longleftarrow \{\}$
$\mathcal{K} \longleftarrow \{0 \leq x_i \leq 1 \text{ s.t. } 0 \leq i \leq d\}$
while $T \neq \emptyset$ **do**
    $T, D \longleftarrow$ **RemoveDominatedPairs**$(T, \mathcal{K})$
    $\bar{V}_i, \bar{V}_j \longleftarrow$ **Halve-Λ-Query**$(T, \mathcal{K})$
    $\mathcal{K}$, ans $\longleftarrow$ **Query**$(\bar{V}_i, \bar{V}_j, \mathcal{K})$
    if ans = yes **then**
        $T, D \longleftarrow$ **RemovePair**$(\bar{V}_i, \bar{V}_j, T, D)$
    else
        $T, D \longleftarrow$ **RemovePair**$(\bar{V}_j, \bar{V}_i, T, D)$
return FindBest(D)

Finally, Algorithm 16 ranks objects using pairwise comparisons. It starts with a set of vector pairs in $\bar{\mathcal{V}}_\epsilon$ set, *i.e.*, $T = \{(\bar{V}_i, \bar{V}_j) \in \bar{\mathcal{V}}_\epsilon^2 | i < j\}$. It removes all comparable pairs from set T, and it finds the most informative query using the *Halve-Λ-Query* function. Then, It continues by asking this question to the user and updating the Λ polytope, T and D sets based on the user's response.

This iterative algorithm continues until the T set (set of To be Determined pairs) is empty, and all pairs are comparable inside set D. Knowing the labeled elements in set D, the *FindBest* algorithm can compute the most preferred vector $\bar{V}^*$ in a linear time. It first assign a counter to each $\bar{V} \in \bar{\mathcal{V}}_\epsilon$, i.e. $c_i = 0 \; \forall i \in \{1, \cdots, |\bar{\mathcal{V}}_\epsilon|\}$. For any $\bar{V}_i \in \bar{\mathcal{V}}_\epsilon$, this factor increases when $(\bar{V}_i, \bar{V}_j) \in D$. At the end, the most preferred vectors $\bar{V}_i$'s are those with $c_i = 0$.

4.4.4 Theoretical implications

Number of vectors in $\bar{\mathcal{V}}_\epsilon$: In Section 4.4.2, we have shown how to find an approximated subset of optimal policies. This is because computing all the vector-valued functions (equally vector values of all policies) is not feasible. Therefore, we have tried to find a subset of non-dominated policies $ND(\bar{\mathcal{V}})$ with respect to a given Λ polytope.

According to the three suggested comparisons methods (given in Section 3.3.4), if a vector-valued function $\bar{V}_\beta$ is non-dominated with respect to the Λ polytope, it should be non-dominated with respect to the Pareto comparison, i.e.:

$$\text{If } \bar{V}_\beta \in ND(\bar{\mathcal{V}})^6 \text{ then } \bar{V}_\beta \not\prec_P \bar{U}_\beta \;\; \forall \bar{U}_\beta \in \mathcal{V} \backslash \bar{V}_\beta.$$

In fact, $ND(\bar{\mathcal{V}}) \subset ND_{\text{Pareto}}(\bar{\mathcal{V}})$. The advantage of using the ϵ precision for computing an approximate subset of $ND(\bar{\mathcal{V}})$ is based on the results of Perny et al. [2013] work on ϵ-covering concept and on Papadimitriou and Yannakakis [2000] results.

Observation 4.1. *For a given number of objectives d and for any precision $\epsilon > 0$, the size of non-dominated vectors with precision ϵ i.e. $\bar{\mathcal{V}}_\epsilon$ is exponential w.r.t d. It implies that if $V_{max} = max_{\bar{V}_\beta \in \bar{\mathcal{V}}} ||\bar{V}_\beta||_\infty$ then $|\bar{\mathcal{V}}_\epsilon| \sim d(V_{max}/\epsilon)^d$*

This bound does not depend on the MDP parameters or on its size. This observation indicates how the size of non-dominated vector-valued functions changes exponentially with respect to d. This shows how increasing the d parameter makes the non-dominated vectors exploration more difficult. Another parameter that plays a significant role in our calculation is the ϵ precision. The size of $\bar{\mathcal{V}}_\epsilon$ changes with respect to $O(\frac{1}{\epsilon^d})$, showing that the proposed method for searching non-dominated vector-valued functions depends on the size of d while it has no dependency on the size of the MDP.

Note that an alternative way of defining an ϵ-covering (see [Perny et al., 2013]) would be a logarithmic bound in $\frac{1}{\epsilon}$ that is not different from our ϵ precision s-assumptions.

4.4.5 Experimental Evaluation

Our experimental results include two parts: the first part analyzes the Propagation Algorithm (algorithm 12), while the second part focuses on the Propagation-Search Value Iteration algorithm (algorithm 12 and algorithm 16) and it compares it with the IVI algorithm, an existing interactive value iteration method for exploring the optimal policy regarding agent preferences Weng and Zanuttini [2013]). All experiments have been implemented with Python version 2.7 and CPLEX has been used as a solver for the linear programming problems.

[6] For more information see Definition 4.3

4.4.5.1 Simulation domains: random MDPs

A random MDP model is the same defined in Section 4.5.2.1. Recall that the weight polytope Λ is initialized as a unite d-dimensional hyper-cube.

$\lvert S\rvert = 128$	$\epsilon = 0.5$	$\epsilon = 0.2$	$\epsilon = 0.1$	$\epsilon = 0.05$
$d = 3$	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 4.1: $|\mathcal{V}_\epsilon|$ as a function of precision ϵ. Results are averaged on 10 random MDPs with $|A| = 5$.

Table 4.1 indicates how the number of non-dominated vector-valued functions changes with respect to the accuracy ϵ (refer to Algorithm 12). $\mathcal{V}_\epsilon$ is a polytope on a finite number of states. This table demonstrates —as expected— that $|\mathcal{V}_\epsilon|$ increases when the precision increases. when $|\mathcal{V}_\epsilon| = d$ we have that the algorithm stops after only one iteration As expected, the algorithm converges quickly for greater values of ϵ. This is due to the fact that increasing the precision produces too many non-dominated vector-valued functions. we have noted with «timeout» the cases where the algorithm is not able to converge.

To better understand the behaviour of our algorithm, we study more in details one of the setting that reached the timeout. The two graphs in Figures 4.6 and 4.7 compare two random MDPs with $|A| = 5$, $d = 4$, $\epsilon = 0.05$ and different number of states ($|S| = 128$ and $|S| = 256$). The results have been averaged on 10 different MDPs. Diagram 4.7 indicates how the number of non-dominated vectors generated changes at each iteration of Algorithm 12. Also, diagram 4.6 shows the time elapsed at each iteration (in minutes) when generating non-dominated vectors. Graph 4.6 indicates that the propagation algorithm is computationally expensive and time consuming, but Figure 4.7 illustrates how $|\mathcal{V}_\epsilon|$ converges to a constant number after a small number of iterations. This part is a pre-processing stage of MDPs and we believe that it is possible to significantly speed up this process in the future.

Our approximation method of the 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 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 the optimal policy (Algorithm 16) on the 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 the Λ polytope and the results have been averaged on random selection of agents. That means that the error of our approach $\bar{V}^{\pi^*}$ is an average

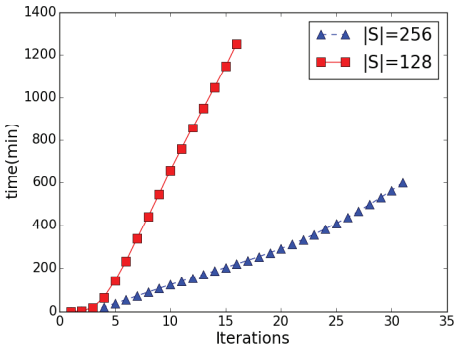

FIGURE 4.6: Average cardinal of non-dominated vectors generated by Algorithm 12

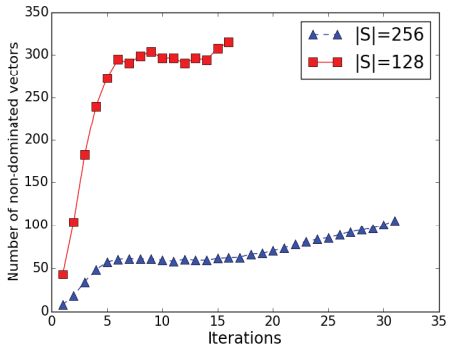

FIGURE 4.7: These graphs illustrate how algorithm 12 behaves on MDPs with 5 actions and two different number of states 128 and 256. Also $d = 4$ and ϵ precision is 0.05

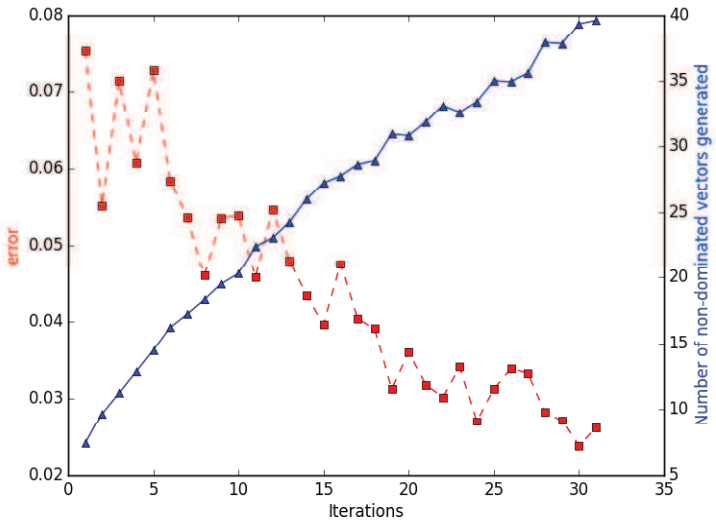

FIGURE 4.8: Error and number of non-dominated vector-valued functions v.s. iterations in Algorithm 12

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 12 indicates how the error and the number of generated vectors change after each iteration on non-dominated vectors generation. Note that the generated vectors during generation are not all non-dominated after any iteration step, but the algorithm converges to the set of non-dominated vectors at the end. The propagation algorithm generates a finite number of non-dominated vectors at each iteration. This figure illustrates how the number of generated vectors increases iteration after iteration. To calculate the error after each iteration, the search algorithm (Algorithm 16) gets the set of generated vectors and finds the optimal policy satisfying the 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 4.2: 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	$\lvert\bar{\mathcal{V}}_\epsilon\rvert$	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
	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

Finally, Tables 4.2 and 4.3 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 12) while the propagation time is the time of accomplishing this process in seconds. To examine the search optimal policy method (Algorithm 16) 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 the interactive value iteration in two measures: number of queries asked to the user, and computing time.

TABLE 4.3: 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	$\lvert\bar{\mathcal{V}}_\epsilon\rvert$	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
	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

These 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 the IVI algorithm in order to find the optimal policy with the same accuracy. For instance, regarding Table 4.3, 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 one 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 the advantages.

4.5 Advantage Based Value Iteration Algorithm for VMDPs

Since the propagation-search value iteration method has a time consuming and sometimes complex pre-processing part on MDPs, in this section we present a new algorithm, namely *Advantage-Based Value Iteration (ABVI)*, to find an approximation of the optimal policy for a given VMDP with bounded polytope of reward weights Λ. In this approach processing the MDPs and communicating with the users are done in the same time. For this reason we are not obliged to finish a long pre-processing stage before finding the optimal policy. Instead, the optimal policy can be found during the processing phase by asking effective queries to the user.

4.5.1 ABVI Algorithm

The advantage based value iteration algorithm is inspired by the *interactive value iteration (IVI)* algorithm [Weng and Zanuttini, 2013]. It attempts to improve IVI's drawbacks by reducing number of queries to the user and speeding up the algorithm convergence. To achieve this goal, it regroups similarly close policies in one group. Comparing group of policies instead of policies reduces the total number of comparisons. That allows the iterative method converges earlier to its stopping criteria. In the rest of this chapter, we explain first how to regroup policies using advantages in each iteration part of IVI algorithm and second we will present the ABVI algorithm in detail.

Regarding the IVI algorithm, there is an iterative part inside Algorithm 8 that has been repeated here:

Algorithm 17 iterative part inside IVI algorithm

for each state s **do**
  $\bar{V}_{\text{best}} \longleftarrow (0, \cdots, 0)$
  for each action a **do**
    $\bar{Q}(s,a) \longleftarrow \bar{r}(s,a) + \gamma \sum_{s'} p(s'|s,a)\bar{V}_{t-1}(s')$
    $\bar{V}_{\text{best}} \longleftarrow getBest(\bar{V}_{\text{best}}, \bar{Q}(s,a))$

The *getBest* function (given in Algorithm 9) compares $\bar{V}_{\text{best}}$ and $\bar{Q}(s,a)$ according to the three comparison methods listed in section 3.3.4: Pareto dominance, Kdominance and querying the user.

Considering the function *getBest* and the iterative step in IVI, in each inner loop two d-dimensional vectors should be compared. Remind that each vector represents the value function of a policy. As an example, suppose in $getBest(\bar{V}_{\text{best}}, \bar{Q}(\hat{s},\hat{a}))$, the two vectors $\bar{V}_{\text{best}}$ and $\bar{Q}(\hat{s},\hat{a})$ represent two policies π and $\pi^{\hat{s}\uparrow\hat{a}}$ respectively. According to Algorithm 17, the equivalent policies of any two comparable vectors are different from each other only in one (state, action) pair. Here π and $\pi^{\hat{s}\uparrow\hat{a}}$ are different from each other only in state $\hat{s}$ and action $\hat{a}$, i.e. $\pi(\hat{s}) = a$ and $\pi^{\hat{s}\uparrow\hat{a}}(\hat{s}) = \hat{a}$, while in the rest of states $s \in S - \{\hat{s}\}$ they are exactly the same.

In our setting, a learning agent is acting in the environment and she gets a d-dimensional vector reward feedback from the environment after any interaction. MDP goal is to approximate the optimal policy with the highest vector-valued function for each state. According to IVI algorithm, to select the action with the highest vector-value ($\max_a \bar{Q}(s,a)$)

our learning system should be aware of comparison over vector rewards (has been implied in Algorithm *getBest*). We assume that our learning system can query the tutor, whenever it does not know which vector is preferred for the given pair.

Since querying the user or the expert is expensive, we intend to keep number of asked queries as minimum as possible. To delay communication with the user and avoid to ask unnecessary questions to her, our idea is to generate more informative queries in each iteration. In Algorithm 17 as an iterative part of Algorithm 8, each state s generates $|A|$ different increments of the form $\bar{A}(s,a) = \bar{Q}(s,a) - \bar{V}_{t-1}(s)$ in every iteration. In order to find a solution for function *getBest* on each state s, $\bar{V}_{t-1}(s)$ is compared with the set of new vectors $\{\bar{V}_{t-1}(s) + \bar{A}(s,a_1), \ldots, \bar{V}_{t-1}(s) + \bar{A}(s,a_m)\}$ (where $A = \{a_1, .., a_m\}$ is the set of actions for the given VMDP). Therefore, the Algorithm 17 can be written as:

for each s **do**

$\bar{V}_{\text{best}} \longleftarrow (0, \cdots, 0)$

for each a **do**

$\bar{V}_{\text{best}} \longleftarrow getBest(\bar{V}_{\text{best}}, \bar{V}_{t-1}(s) + \bar{A}(s,a))$ [7]

Note that, there are $|S| \times |A|$ total number of comparisons among vectors in each iteration. The worst case is that all these comparisons should be compared using the Kdominance comparison method as a linear programming problem. It means, $|S| \times |A|$ number of LP problems should be solved.

Example 4.5.1. Suppose Figure 4.9 as a VMDP with two states {hometown, beach} and six actions {swimming, reading book, going to exhibition, biking, wait, move}. Reward vectors are such that the first element represents "sportive" coefficient of activity, while the second one shows its "artistic" value. For instance, in $\bar{r}$(hometown, biking) vector the second element is zero, because biking is not an artistic activity. Suppose each

actions / states	going to exhibition	wait	biking	move	reading book	swimming
hometown	$(0,1)$	$(0,0)$	$(0.3,0)$	$(0,0)$	-	-
beach	-	$(0,0)$	-	$(0,0)$	$(0,0.5)$	$(1,0)$

TABLE 4.4: A table for advantages $\bar{A}(s,a)$

state has the initial value $(0,0)$: $\bar{V}$(hometown) $= \bar{V}$(beach) $= (0,0)$. Using interactive value iteration, in every state the vector value should be compared set of vectors in each

[7] $\bar{Q}(s,a) \longleftarrow \bar{V}_{t-1}(s) + \bar{A}(s,a)$

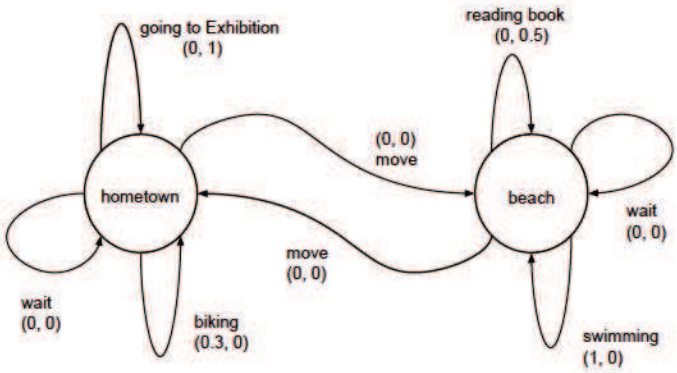

FIGURE 4.9: An example of small VMDP with 2 states and 6 actions.

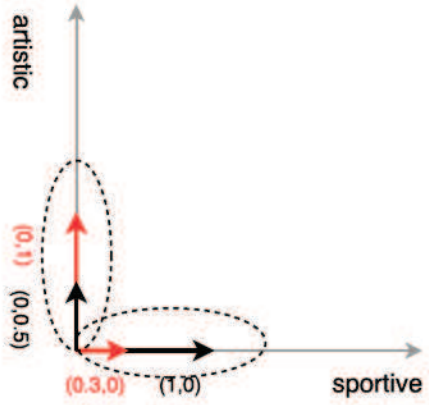

FIGURE 4.10: Plotted Advantages in 2-dimensional space including sportive and artistic activities.

iteration. Regarding to table 4.5.1, for example in state "home town", $(0,0)$ vector is compared to 4 new vectors related to "going to exhibition","move", "biking" and "wait" (respectively $\{(0,0)+(0,1),(0,0)+(0,0),(0,0)+(0.3,0),(0,0)+(0,0)\}$). Similarly in state "beach", $\bar{V}(\text{beach}) = (0,0)$ should be compared with 4 vectors $\{(0,0)+(0,0.5),(0,0)+(0,0),(0,0)+(1,0),(0,0)+(0,0)\}$.

In order to accelerate value iteration, diminish number of generated queries and reduce the IVI algorithm 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 $\bar{V}_t(s)+\bar{A}_{s,a}$ for each s,a, we try to compare $\bar{V}_t(s)$ with groups of state-action pairs representing vector $\bar{V}_t(s)+\sum_{(s,a)\in\text{group}}\bar{A}_{s,a}$.

Example 4.5.2. Following example 4.5.1, regrouping advantages reduces the number of total comparisons among vectors. Regarding Figure 4.10, we can regroup all advantages (illustrated in table 4.5.1) in two groups. Unlike the previous example that includes 8 vectors to be compared, regrouping advantages produce only two vectors. One group generates vector $(0, 1.5)$ representing all artistic activities and the other one $(1.3, 0)$ proposes just the sportive activities. Note that each group can indicate a policy for instance, $(0, 1.5)$ is a policy π such that π(hometown) = going to exhibition and π(beach) = reading book.

Algorithm 18 cluster advantages and returns back a set of vector-valued functions and their related policies

Inputs: set of advantages A_{adv}, vector-valued function $\bar{V}$, policy π
Outputs: S_Π set of policies and their equivalent vector-valued functions

1: $S_\Pi \longleftarrow \{\}$
2: $C \longleftarrow$ cluster A_{adv} using cosine similarity metrics
3: **for** $c \in C$ **do**
4: $\quad \pi_c \longleftarrow$ **makePolicy**(c, π)
5: $\quad$ **value**$(\pi_c) \longleftarrow \sum_s \beta(s)\bar{V}(s) + \sum_{\bar{A} \in c} \bar{A}$
6: $\quad S_\Pi \longleftarrow S_\Pi \cup (\pi_c,$ **value**$(\pi_c))$
7: **return** S_Π

Algorithm 19 implements this idea. For the sake of simplicity, we first introduce function *Cluster-Advantages* given in Algorithm 18. It takes a set of advantages $\mathcal{A}_{\text{adv}}$, a policy π and its equivalent vector-valued function $\bar{V}$. It first clusters the $\mathcal{A}_{\text{adv}}$ set. Then ,for each cluster c it produces a new policy π_c w.r.t the given policy π. It is done by *makePolicy* function as below:

$$\pi_c(s) = \begin{cases} a & \text{such that } \bar{A}_\beta(s, a) \in c \\ \pi_{\text{best}}(s) & \text{if no such } a \text{ exist.} \end{cases}$$

If there are several $\bar{A}_{s,a} \in c$ for the same state s, the algorithm randomly selects one of them, say $\bar{A}_{s,a'}$, and assigns action a' to $\pi_c(s)$. The equivalent vector-valued function of the new policy π_c should be computed too, i.e. :

$$value(\pi_c) = \sum_s \beta(s)\bar{V}(s) + \sum_{\bar{A} \in c} \bar{A}$$

To summarize, each cluster c produces a (policy, vector-valued function) pair $(\pi_c,$ **value**$(\pi_c))$

Algorithm 19 receives a VMDP, a d-dimensional unit cube Λ and a precision ϵ as input. In this algorithm we start with a random policy $\pi_{\text{best}} \in \Pi$ and as IVI algorithm we suppose $\forall s \in S,\ \bar{V}_0(s) = \mathbf{0}$; $\mathbf{0}$ is a zero vector of dimension d as the number of preferences or objectives in MDP with unknown reward values.

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 criterion is discussed at the end of this section. In the following, C denotes the set of clusters generated from A_{adv}.

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

Algorithm 19 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**
    $\bar{Q}_t(s,a) \longleftarrow \bar{r}(s,a) + \gamma \sum_{s'} p(s'|s,a)\bar{V}_t(s')$
    $\bar{A}_\beta(s,a) \longleftarrow \beta(s)\{\bar{Q}_t(s,a) - \bar{V}_t(s)\}$
    $A_{\text{adv}} \longleftarrow$ Add $\bar{A}_\beta(s,a)$ to A_{adv}
  $S_\Pi \longleftarrow$ **Cluster-Advantages**$(A_{\text{adv}}, \bar{V}_t, \pi_{\text{best}})$
  for $(\pi_c, \textbf{value}(\pi_c)) \in S_\Pi$ **do**
    $(\pi_{\text{best}}, \mathcal{K}) \leftarrow$ **getBest**$(\textbf{value}(\pi_c), \textbf{value}(\pi_{\text{best}}), \mathcal{K})$
  for each s **do**
    $\bar{V}_{t+1}(s) = \bar{r}(s, \pi_{\text{best}}(s)) + \gamma \sum_{s'}\ p(s'|s, \pi_{\text{best}}(s))\bar{V}_t(s')$
  $t \longleftarrow t + 1$
until $||\sum_s \beta(s)\bar{V}_{t-1} - \textbf{value}(\pi_{\text{best}})|| \leq \epsilon$

Assume *getBest* receives two vectors $value(\pi_i) = \bar{V}_i$ and $value(\pi_j) = \bar{V}_j$. 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 14 and 15). 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 δ for clustering will define the final number of classified vectors. Suppose a threshold of δ, meaning that clusters satisfy

$$\forall c \in C,\ \forall \bar{A}^i_\beta, \bar{A}^j_\beta \in c \quad d_{\text{cosine-similarity}}(\bar{A}^i_\beta, \bar{A}^j_\beta) \leq \delta \tag{4.6}$$

Definition 4.4. For two given vectors $\bar{a}$ and $\bar{b}$, the cosine similarity is represented as:

$$\text{similarity}(\bar{a}, \bar{b}) = \frac{\bar{a} \cdot \bar{b}}{||\bar{a}||||\bar{b}||}$$

This threshold defines a maximum angle between any two vectors of each cluster. For instance in Figure 4.11, 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 less comparisons or queries.

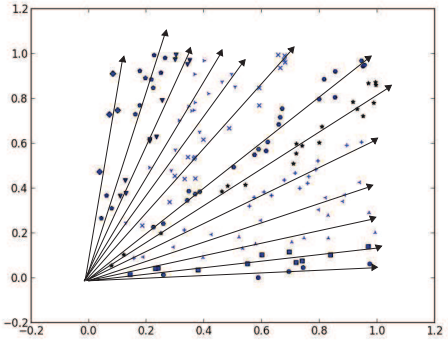

FIGURE 4.11: Clustering on advantages for a given VVMDP with 2-dimensional $\bar{\lambda}$s. Each different group points constitutes a cluster and each vector is equivalent to the sum of $\bar{\lambda}$s in the corresponding cluster.

In Algorithm 19, there are $|S_\Pi|$ comparisons in each iteration (see line 13. $|S_\Pi|$ is the number of clusters on advantages). In comparison, Algorithm 8 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 4.11 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 $\bar{A}_{s,a}$. Clustering on advantages returns 13 clusters. This allows us to compare $\sum_s \beta(s)\bar{V}_t(s)$ with 13 vectors instead of comparing it with 150 vectors for the IVI algorithm.

4.5.2 Experimental Evaluation

ABVI algorithm targets querying more informative questions and reducing computational complexity. It is another form of IVI that computes vector values of policies iteratively and communicates with user whenever it fails to compare two vectors with each other. IVI method enumerates all policies and compares their equivalent vector-values without skipping less informative comparisons, while ABVI selects more informative vectors by classifying advantages. It means comparing two clusters in ABVI finds an answer for several queries in IVI by asking only one query. ABVI is also tolerant toward inconsistent and unreliable user feedback. It means this algorithm can find the correct solution for uncertain users who make mistakes in their comparisons some few times. Or the users who do not know answer of some comparison questions regarding their final goal.

This section evaluates empirical performance of ABVI algorithm on several models with two types of users. First one is a confident user who answers all queries correctly, and the second one is an uncertain user who makes mistakes in answering comparison questions few times. All experiments in this section 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.

In the following, we will test the ABVI approach on random MDPs with different number of states, actions and parameter d. The results have been tested on two types of users including confident and uncertain users.

4.5.2.1 Simulation Domains: Random MDPs with Confident User

This section demonstrates the performance of ABVI algorithm (given in Algorithm 19) on some simulated random MDPs for confident users who answer all queries correctly. It also includes a comparison between ABVI performance and IVI algorithm (algorithm 8).

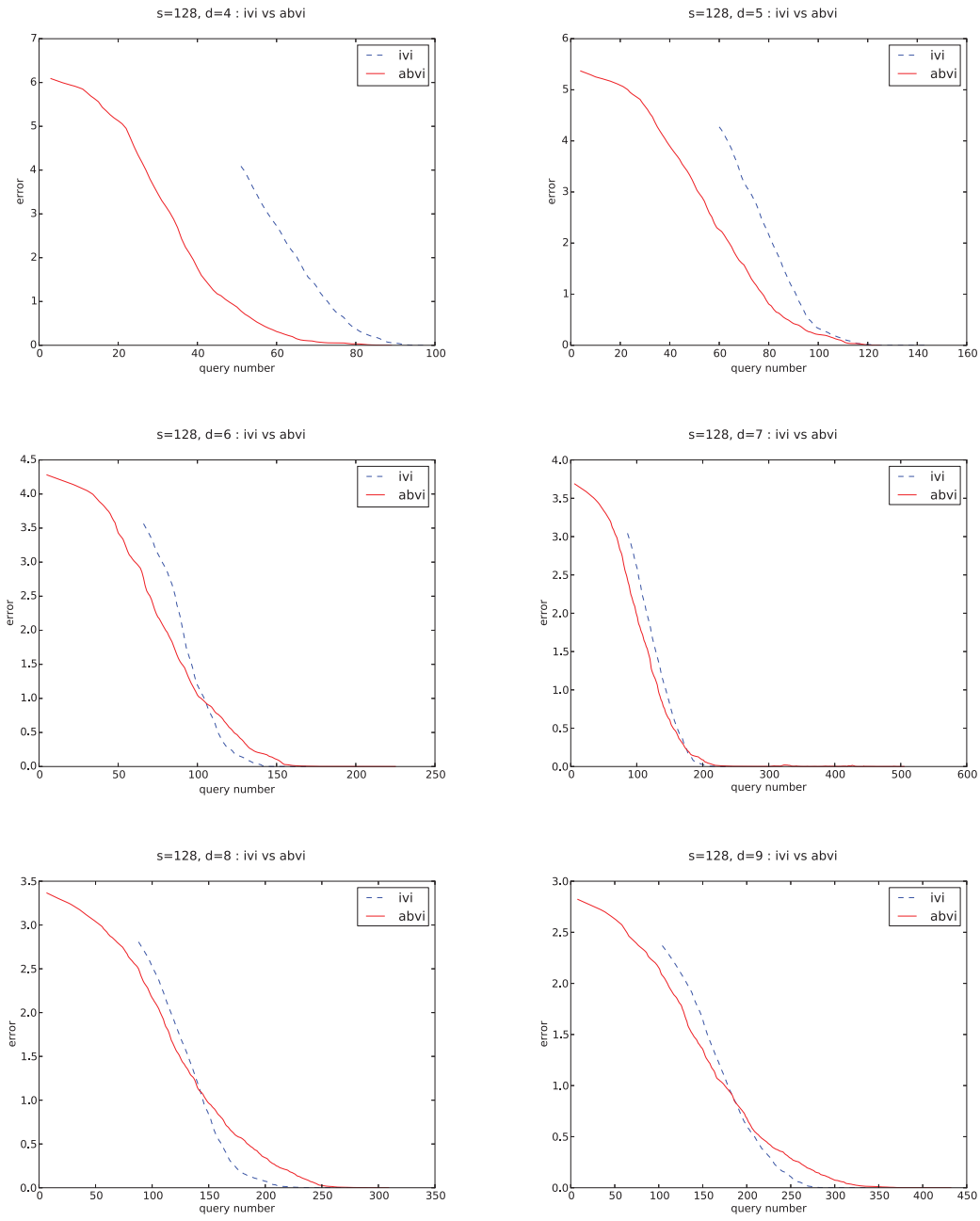

FIGURE 4.12: These graphs illustrate errors vs the number of queries answered by a **confident user** where $|S| = 128$ and $m = 5$ for $d = 4, 5, 6, 7, 8, 9$.

Definition 4.5. A random VMDP is defined by several parameters including its number of states n, its number of actions m and its weight space dimension d. Real parameters have several properties:

- All rewards are bounded between 0 and 1.
- From any state s transitions only reach $\lceil \log_2(n) \rceil$ states.
- For each pair (s, a), reachable states are drawn based on uniform distribution over the set of states.
- For drawn states, transition probabilities are formed based on Gaussian distribution with mean 0.5 and variance 0.5.
- The initial state distribution β is uniform
- The discount factor is chosen $\gamma = 0.95$.

Remind that weight polytope Λ is initialized as a unite d-dimensional hyper-cube.

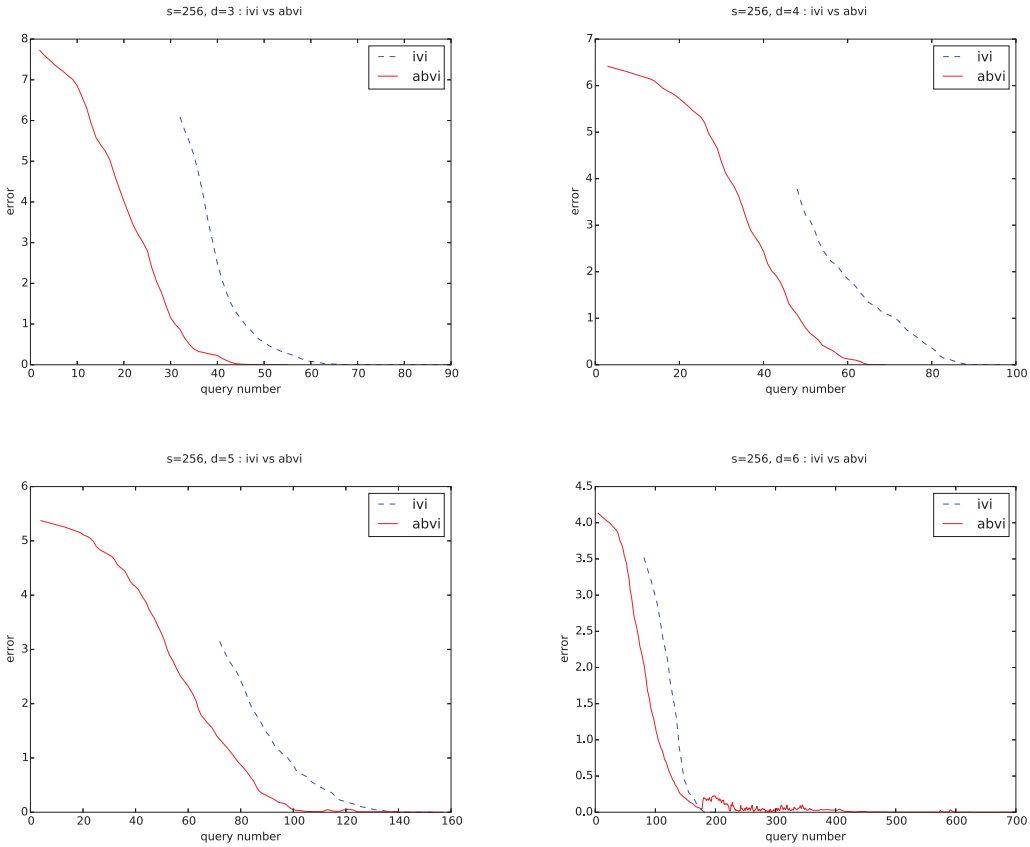

FIGURE 4.13: These graphs illustrate errors vs the number of queries answered by a **confident user** where $|S| = 256$ and $d = 3, 4, 5, 6$

To simulate the user and her answers to queries, a $\bar{\lambda}$ vector is chosen embedding her particular priorities. So the exact vector-valued function is obtainable, and assigned as $\bar{V}_\beta^{\text{exact}}$. Thus, demonstrated error in these section's figures is defined as the difference between the exact vector and the vector obtained at the end of each iteration for methods IVI and ABVI. For instance, assume $\mathbf{value}(\pi^*) = \bar{V}_\beta^{\text{exact}}$ is the exact answer, and $\mathbf{value}(\pi') = \bar{V}_t$ is the result obtained after t-th iteration for a given VMDP. The exact error w.r.t the given $\bar{\lambda}$ is calculated by $\bar{\lambda} \cdot \bar{V}_\beta^{\pi'} - \bar{\lambda} \cdot \bar{V}_\beta^{\text{exact}}$. But, the exact value of $\bar{\lambda}$ is not known and we use the following formula to calculate the errors approximately:

$$\text{error} = ||\bar{V}_\beta^{\pi'} - \bar{V}_\beta^{\text{exact}}||_\infty$$

Our experiments have been done on a random MDP with 5 actions and two various number of states 128 and 256. In Figure 4.12, the number of states is $n = 128$ and the error is compared to the number of asked queries for $d = 4, 5, 6, 7, 8$. The figure displays that errors in ABVI algorithm are always less than IVI algorithm for small d-dimensions. For example when $d = 5$, after proposing 90 queries to user, ABVI error falls down to 0.4, while IVI algorithm still has an error around 1.1. For higher d dimensions such as $6, 7, 8$ and 9, error in ABVI reduces quicker than in IVI , except to gain the optimal solution with error 10^{-4}, in which case ABVI asks more queries 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 optimal policy with less precision, ABVI will converge to the optimal solution by asking less number of queries.

Similarly, Figure 4.13 compares number of required queries with respect to the error for both approaches where $|S| = 256$ and $d = 3, 4, 5, 6$. Considering these figures for all given dimensions d, ABVI converges faster that IVI approach, but the difference between two generated errors reduces for higher dimensions.

The series of Figures 4.12 and 4.13 indicate that clustering advantages is less effective on higher dimension d and at the end of ABVI algorithm. The reason is that closer to the convergence point, advantages size reduce and this decrease the general vector growth. Thus our idea is to focus on clustering parameters. Referring to Equation 4.6, the δ parameter determines clusters measure. Higher value of δ gather more number of points in the same cluster and reduce number of new generated vectors. Figures 4.14

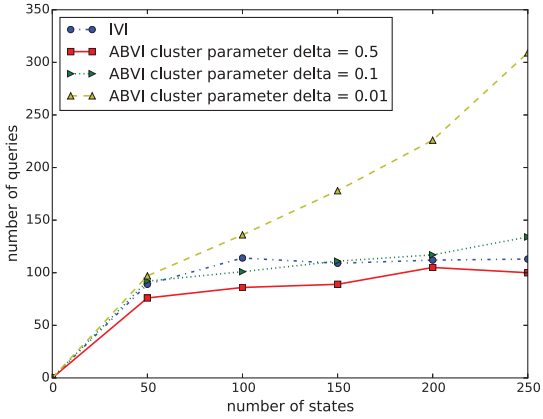

FIGURE 4.14: number of queries vs number of states for ivi and three different ABVI with various value of parameter δ for clustering advantages.

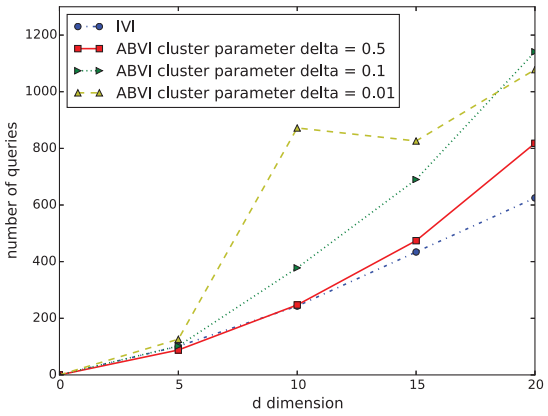

FIGURE 4.15: number of queries vs dimension d for ivi and three different ABVI with various value of parameter δ for clustering advantages.

and 4.15 investigate δ changing efficiency on clustering advantages. As can be observed in Figure 4.14, for random MDPs with 5 actions and $d = 5$, ABVI approach with $\delta = 0.5$ asks less number of queries for various number of states.

On the other hand, Figure 4.15 illustrates how the number of queries changes with respect to the d dimension for random MDP with 5 actions and 50 states. This figure shows that ABVI with $\delta = 0.5$ has better results than other approaches until $d = 10$. By rising dimensions d, the IVI method will produce less number of queries to find the optimal policy. In two given Figures 4.14 and 4.15 number of queries indicate the total number of asked queries to the user until reaching a convergence with precision $\epsilon = 10^{-4}$.

According to these results and Section 4.4.5 similar to IVI method, we expect that ABVI approach queries more number of questions than Propagation-search algorithm (given in Section 4.4). It also will find an optimal policy with higher accuracy than the PSVI response. Comparing two algorithms PSVI and ABVI on two different types of users (confident and uncertain users) will be one of future work as well.

4.5.2.2 Simulation Domains: Random MDPs with Uncertain User

This section studies the performance of ABVI algorithm in comparison with IVI for random MDPs with uncertain users who answer some comparison questions wrong considering their final goal. An uncertain user replies back queries with a noise generated

with respect to parameter θ. It means for an uncertain user, her preferences on objectives is defined by reward weight $\bar{\lambda}'$ vector[8], therefore for comparing two given vectors $\bar{U}$ and $\bar{V}$ the uncertain user response will be defined as the following comparison:

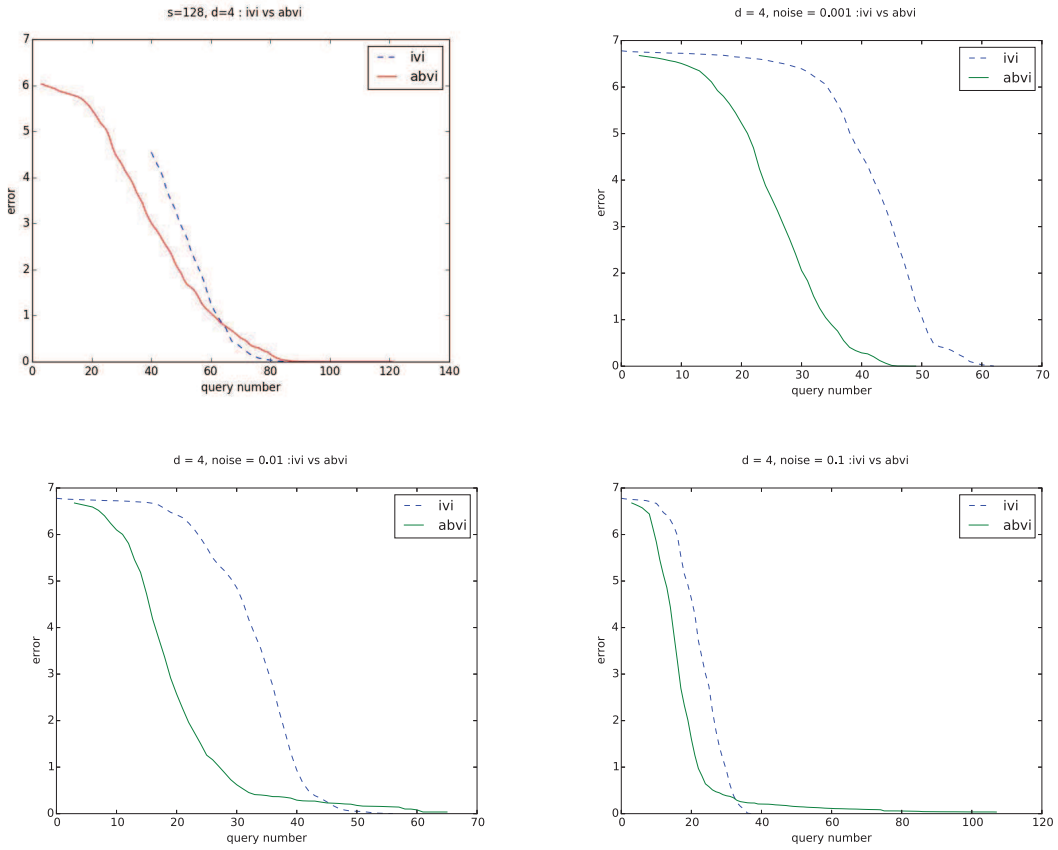

FIGURE 4.16: These graphs illustrate errors vs the number of queries answered by **confident user** and **uncertain user** where $|S| = 128$ and $d = 4$. The upper left figure illustrates the performance of algorithms for **confident user** while the rest of figures are for **uncertain user** with $\theta = 0.001, 0.01$ and 0.1 respectively for top-right, bottom-left and bottom-right.

$$\text{is } \bar{\lambda}' \cdot \bar{U} \ \geq \ \bar{\lambda}' \cdot \bar{V} + \text{noise?}$$

where noise is randomly selected from a normal distribution $\mathcal{N}(0, \theta)$. Consider if the user was confident, her response would be the exact comparison between two scalars $\bar{\lambda}' \cdot \bar{V}$ and $\bar{\lambda}' \cdot \bar{U}$.

Figures 4.16 examine ABVI and IVI performance for $|S| = 128$, $|A| = 5$ and $d = 4$ for various noise parameters θ. In these figures clustering parameter δ has been selected

[8]The $\bar{\lambda}'$ vector is unknown to our learning system

equal 0.01. They demonstrates ABVI is more robust for uncertain users rather than confident users.

Based on our experimental results, unlike IVI algorithm that requires reliable users who answer comparison queries without any mistakes, clustering advantages gives an approximated solution for VMDPs and it has a better performance for unreliable users with a slight percentage of mistakes in answering comparison questions.

4.6 Conclusion and Discussion

In this chapter we have proposed a novel methods for the preference-based sequential decision problems using vector-valued MDPs with unknown reward weights bounded in a polytope, namely Λ using interactions with users. First, we proposed a propagation-search method that is able to ask a reduced number of queries to the user. Second, we indicated how to tackle Interactive Value Iteration method drawbacks — IVI is given in [Weng and Zanuttini, 2013] — by reducing the calculation time and the number of interactions with the user.

Concerning the propagation-search method, we have shown that it is possible to explore the set of non-dominated policies using the clustering advantages. We have shown that the number of explored non-dominated policies are exponentially related to the number of preferences d and the ϵ precision of the environment. After exploring non-dominated policies, we have explained how to communicate with user to ask less possible number of queries in order to find the optimal policy from satisfying her preferences and priorities. We showed that if the number of non-dominated policies increases exponentially w.r.t d and ϵ, the number of communications with users are very few. We have compared our result with the IVI algorithm as a proof of this claim.

For tackling IVI flaws, we have presented ABVI algorithm based on clustering advantages. In this iterative approach regrouping advantages accelerate the value iteration algorithm and produces queries that answers some less informative queries generated inside IVI. Therefor in comparison with IVI, ABVI algorithm converges faster with proposing less number of queries to the user. The only problem with this algorithm is that at the end of its convergence, it converges slower because of the regrouping of some advantages in the opposite direction of convergence.

Chapter 5

Computing Robust Policies with Minimax Regret Methods

In previous chapters, we designed algorithms able to find good policies by querying the user repeatedly. The more queries are asked to the user, the smaller the Λ polytope becomes, and the better the policy is. In case we are not allowed to ask too many queries to the user, the Λ polytope at the end of the elicitation process might still be big. Intuitively, this means that there are still a lot of uncertainty about the user's preferences. Thus, we need a criteria to select a policy which is robust with respect to this uncertainty. As discussed in the earlier chapters, the min-max regret criteria allows us to choose a policy in such a way that even if the user's preferences is not precisely known, the chosen policy will still be acceptable. There are many existing algorithms to compute an approximation of the min-max regret optimal policy [Regan and Boutilier, 2011b, 2012], but these algorithms are often intractable for large MDPs [Xu and Mannor, 2009]. In this chapter, we propose a new algorithm able to quickly compute approximately optimal policies w.r.t. the min-max regret criterion.

In this chapter, we present an heuristic method, namely *selected random points method*, to solve the minimax regret problem. An advantage of this method is that it approximates the optimal policy forMDPs with higher size without requiring many queries. In Section 5.1, we introduce our new approach and in Section 5.2 we will provide some experimental results.

5.1 Selected Random Points Method

The exact existing methods for the minimax regret computation are complex and can not be applied on large scale MDPs with unknown set of rewards [Regan and Boutilier, 2008, 2009]. To overcome this problem, we propose an heuristic method for solving the minimax regret problem called Selected Random Points Method (SRPM). The SRPM is applicable to VMDPs where the weight reward is defined by a bounded polytope Λ.Recall that a VMDP can be transformed into an MDP, if the vector $\bar{\lambda}$ is given. Thus, we have: $\forall\, s, a \;\; r(s,a) = \bar{\lambda} \cdot \bar{r}(s,a)$, where $\bar{\lambda}$ and $\bar{r}$ are d-dimensional vectors.

The SRPM is inspired by Algorithm 3 in Section 3.1. Algorithm 3 consists of a master problem and a sub-problem. At each iteration, after the master problem is solved, the sub-problem generates constraints that improves the current solution. The iteration between two linear programs continues until the optimal solution (or a given stopping criterion) is found. SRPM uses a new approach for generating constraints in the sub-problem and, to be able to solve instances of bigger size, it relies on one single iteration between master and sub-problem.

SRPM is examined on a VMDP$(S, A, p, \bar{r}, \gamma, \beta)$with a feasible polytope of reward weights given by a convex polytope $\Lambda = \{\bar{\lambda} \mid \mathbf{C}\bar{\lambda} \leq \mathbf{d}\}$ ($|S| = m$ and $|A| = n$). This setting can be also implemented on IRMDPs as it has been introduced in Section 2.2.1.

The optimal policy $\bar{f}^*$[1] can be computed from Equation 3.3, after replacing the rewards $r(s,a)$ with $r(s,a) = \bar{\lambda} \cdot \bar{r}(s,a)$:

$$f^* = \text{argmin}_{f \in \mathcal{F}} \text{max}_{\bar{\lambda} \in \Lambda} \text{max}_{g \in \mathcal{F}} \sum_s \sum_a \left(\bar{\lambda} \cdot \bar{r}(s,a)\right) f(s,a) - \sum_s \sum_a \left(\bar{\lambda} \cdot \bar{r}(s,a)\right) g(s,a) \tag{5.1}$$

Referring to linear program 2.11, set of all occupancy functions is calculable using set of inequalities. It means, any $f \in \mathcal{F}$ satisfies in the following set of constraints:

[1] the equivalent policy π^* to occupancy function f^* can be obtained from Equation 2.12

$$\sum_{a \in A} f(s', a) - \sum_{s} \sum_{a} \gamma p(s'|s, a) f(s, a) = \beta(s') \;\; \forall \; s' \in S$$
$$f(s, a) \geq 0 \;\; \forall s, a$$

Model 5.1 can be written as the following linear program (similar to Equation 3.4):

$$\begin{aligned} &\text{minimize}_{f,\delta} \;\; \delta \\ &\text{subject to} \\ &\delta \geq \sum_{s,a} (\bar{\lambda} \cdot \bar{r}(s, a)) g(s, a) - \sum_{s,a} (\bar{\lambda} \cdot \bar{r}(s, a)) f(s, a) \quad \text{for all pairs } g \in \mathcal{F}, \; \bar{\lambda} \in \Lambda \\ &f \in \mathcal{F} \end{aligned} \tag{5.2}$$

Model 5.2 is a linear program with an infinite number of constraints (due to the infinite number of λ vectors). To solve it, we start with a master problem that does not contain the first series of constraints. We use a sub-problem to separate a subset of constrains $\delta \geq \sum_{s,a} (\bar{\lambda} \cdot \bar{r}(s, a)) g(s, a) - \sum_{s,a} (\bar{\lambda} \cdot \bar{r}(s, a)) f(s, a)$ for all pairs $g \in \mathcal{F}, \bar{\lambda} \in \Lambda$. Our goal is to deal with a sub set of all the possible $(\bar{\lambda}_i, g_i)$ pairs, called generating set (GEN set) in the following. To obtain the GEN set, we first randomly generate the $\bar{\lambda}_i$ points from the polytope Λ. The procedure used to calculate the g_i associated to a given $\bar{\lambda}_i$ presented in Algorithm 20, where the associated g_i is computed as follows:

$$g_i = \text{Maximize}_{g \in \mathcal{F}} \;\; \sum_{s,a} (\bar{\lambda}_i \cdot \bar{r}(s, a)) g(s, a) \tag{5.3}$$

Algorithm 20 Find-**g**($\bar{\lambda}_i$)

Require: $\bar{\lambda}_i$
Ensure: g_i
1: $g_i \longleftarrow \text{Maximize}_{g \in \mathcal{F}} \;\; \sum_{s,a} \bar{\lambda}_i \cdot \bar{r}(s, a) g$
2: **return** g_i

The SRPM is presented in Algorithm 22. As already mentioned, since the number of constraints is infinite, it is needed to generate an approximated finite set of constraints. This set is the generation set GEN and is computed by the function *Calculate-GEN-set* (given in Algorithm 21). This method heuristically produces N constraints by choosing

Algorithm 21 Calculate-GEN-set(N, Λ)

Require: let N be the number of randomly selected points from polytope Λ
Ensure: GEN set is including constraints for master problem
1: $\Lambda_{random} \longleftarrow \emptyset$ random selected points from polytope Λ
2: $GEN \longleftarrow \emptyset$
3: **for** $i = 1$ to $i \leq N$ **do**
4: choose $\bar{\lambda}_i$ randomly in Λ
5: $\Lambda_{random} \longleftarrow \Lambda_{random} \cup \bar{\lambda}_i$
6: **for** $\bar{\lambda}_i \in \Lambda_{random}$ **do**
7: $g_i \longleftarrow$ find-$\mathbf{g}(\bar{\lambda}_i)$
8: $GEN \longleftarrow GEN \cup \{(\bar{\lambda}_i, g)\}$
9: **return** GEN

Algorithm 22 SRPM Calculate f^* for a given polytope $\Lambda = \{\bar{\lambda} | \mathbf{C}\bar{\lambda} \leq \mathbf{d}\}$

Require: N is the number of selected points from polytope Λ
Ensure: f^*
1: Constraint $\longleftarrow \emptyset$
2: $GEN \longleftarrow$ Calculate-GEN-set(N, Λ)
3: **for** $i = 1$ to N **do**
4: Choose $(\bar{\lambda}_i, g) \in GEN$
5: Constraint $\longleftarrow$ Constraint $\cup \{\sum_{s,a} \bar{\lambda}_i \cdot \bar{r}(s,a) g_i(s,a) - \sum_{s,a} \bar{\lambda}_i \cdot \bar{r}(s,a) f(s,a) \leq \delta\}$
6: Constraint $\longleftarrow$ Constraint $\cup$ constraints of $\mathcal{F}$
7: $f^* \longleftarrow Min_{\delta, \bar{f}}\ \delta$
8: $subject\ to:$ Constraint
9: **return** f^*

N random $\bar{\lambda} \in \Lambda$ and, for each selected $\bar{\lambda}$, it finds the optimal policy $g \in \mathcal{F}$with the Algorithm 20.

The use of Algorithm 22 allows us to changed from a linear program with an infinite number of constraints (Equation 5.1) to a linear programming with $|S|.|A| + 1$ number of variables and $N + |S|$ number of constraints (where N is the number of randomly selected points from Λ polytope). However, every constraint is obtained after solving an LP with $|S|.|A|$ number of variables and $|S|$ number of constraints.

5.2 Experimental Results

The experimental results have been done with R version 3.0.1 and "lp- SolveAPI" package to solve the linear programs. All tests has been averaged on 10 times iteration.

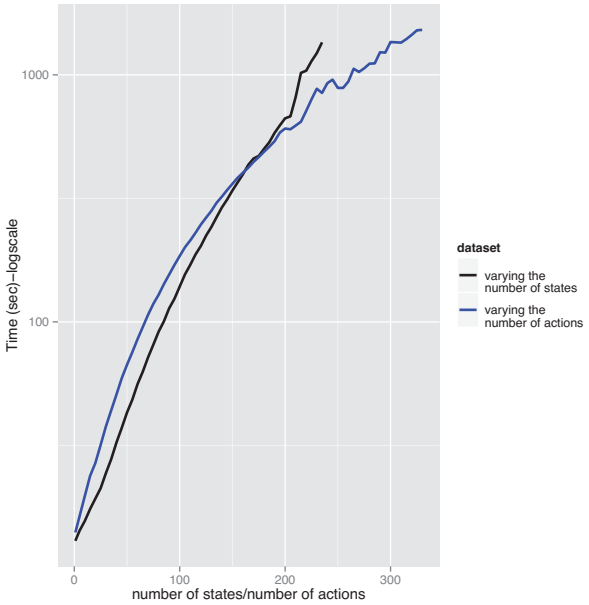

FIGURE 5.1: Changing of states and actions vs calculation time for minimax regret.

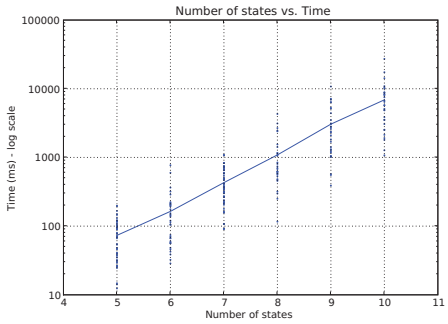

FIGURE 5.2: Scaling of ICG algorithm, number of states vs time [Regan and Boutilier, 2008]

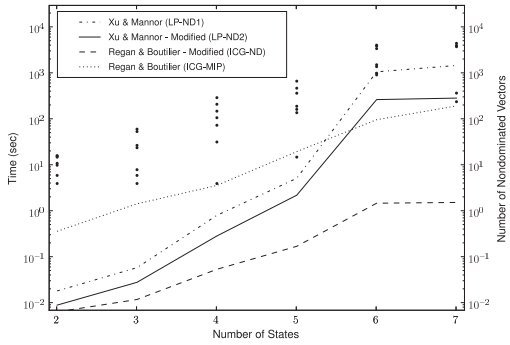

FIGURE 5.3: Scaling of minimax regret computation (line plot on left y-axis) and nondominated policies (scatter plot on right y-axis) w.r.t. number of states [Regan and Boutilier, 2010]

5.2.1 Simulation Domains

In this section, the experiments are based on the Random MDP models presented in Section 4.5.2.1. To measure the performance of our SRPM, we first show how the average minimax regret computation time varies with respect to the MDP size. Figure 5.1 shows how the computing time as function of the number of states or the number of actions. In all the experiment we have $d = 10$. To obtain the states-dependent curve we fixed the

number of actions at 5 and we varied the number of states, while in the actions-dependent curve we fixed the number of states at 10 and we changed the number of actions.

Figures 5.2 and 5.3 show two results from Regan and Boutilier work. The former shows minimax calculation time using an integer linear programming method named ICG proposed in [Regan and Boutilier, 2008] and the latter indicates the minimax calculation time using non-dominated policies. Both methods are computationally heavier than SRPM and hence they are not able to deal with MDPs of huge size, while our method is applicable on larger MDPs (Figure 5.1). For instance, we can calculate minimax regret for an MDP with 200 states, while the previous approaches do not have any results with more than 10 states. Therefore, our approach is effective for MDPs with an high number of states and actions.

5.3 Conclusion and Discussion

This chapter presents our modification on Bender's decomposition method for solving the minimax regret method [Regan and Boutilier, 2009]. We presented a new technique to generate the inequalities in the subproblem, it starts with a random generation of the $\bar{\lambda}$s from polytope Λ and then it finds the best possible g_i by solving a sub-problem. In our approach we showed how to reduce the time calculation and the complexity by generating the master problem constraints only once. In this wey we are able to handle instances of bigger size in comparison to the other methods present in the literature.

Chapter 6

Conclusions and Perspectives

Sequential decision-making under uncertainty is relevant to a large number of fields, from manufacturing to robotics to medical diagnosis and economics. In majority of areas, it means to learn a policy that defines which action to select in any state in order to perform a task correctly. These actions should be robust toward noises in the system. As an example Boger et al. [2006] have designed an assistant robot that should select its action regarding to the dilemma patient behavior and accompany her to wash her hands[1].

The basic model of this dissertation has been based on an MDP with unknown rewards under uncertainty. A first model is fully general: the uncertainty produces an MDP with imprecise rewards (IRMDP) where acceptable reward functions are defined as a set. Then, we have presented a vector valued MDP with vector rewards (VMDP) where the uncertainty is put on unknown weights on rewards that should be predicted during the computation approaches. In fact, this weight is a key point of VMDP model and if it was given, the problem could be transformed to a classical MDP. Therefore, it can be solved easily using classical algorithms on MDPs. A VMDP model with bounded polytope Λ of reward weights provides an imprecise knowledge of the rewards which entails a partial ordering of policies.

This second model enables us to compare policies with respect to the Λ polytope using optimization techniques, while submitting to the user unsolved comparisions between policies, states or (state , action) pairs prunes part of Λ and improves further accuracy

[1] This paragraph has been inspired by Pineau [2004].

of optimization solution. Thus querying users to shrink the Λ polytope and attaining more information about uncertain reward weights have been important issues in this dissertation. Our goal has been to:

- ask less and more effective questions to the user,
- And find optimization methods to solve higher size of VMDPs with respect to polytope Λ.

This goal has been pursued along two lines: reward weight elicitation and computation methods.

6.1 Reward Weight Elicitation

Since reward weights $\bar{\lambda} \in \Lambda$ (embedding user preferences) are not known, we have presented several techniques to optimize optimal solution for VMDPs by approximating $\bar{\lambda}$ close enough to the user preferences. For this reason in our approaches we required to querying users.

Our first technique includes two parts: propagation and research. In propagation part, we have presented a method for computing an approximate set of non-dominated policies. Recall that non-dominated policies are the policies that are at least optimal for one value of preferences (each $\bar{\lambda}$ represent a value of preferences). The propagation part does not require to communicate with users and all computation is carried by the system. We have shown that it generates an approximate set of non-dominated policies pretty well for VMDP with small dimension d. We have proved theoretically that number of generated optimal policies does not depend on MDP sizes such as number of sates or actions. This is interesting because, the d parameter is actually the number of objectives/preferences in the model; it means this method is not suitable for too many objectives in the system.

Research part as the second part of our method is completely independent from propagation. It finds the optimal policy satisfying user preferences from the propagated set of non-dominated policies. Regarding this part, we showed that approximating the set of optimal policies offline, allows us to discover the optimal policy by asking a considerably lesser number of questions to the user. Our experimental results demonstrate the value

of our approach and show how we ask less queries in comparison with other settings in literature so far. In order to compare less number of pairs from the explored set of non-dominated policies, our algorithm searches the more informative ones. Since, comparing each pair produce a cut on polytope Λ, the algorithm attempts to chooses the cuts that divide a given reward polytope into two almost equal parts. To fulfill this goal, our Monte-Carlo based technique generate many random points inside the Λ polytope. We believe that the time for generating these random points could be reduced in a different approach as dividing polytope Λ in approximately equal parts. Another idea is to use hit and run method to select completely random points inside the Λ polytope [Ya and Kane, 2015].

Another algorithm presented in this thesis, ABVI, addresses differently the question of optimal policy for an MDP with a polytope Λ of admissible reward weights. We follow a vector form of Value Iteration method as long as Λ allows to decide the necessary comparisons, and start an interaction with user when an undecided comparison is met. To reduce the number of vector comparisons, our approach computes advantages in each iteration, classifies and regroups them, and performs comparisons on groups of advantages. Experiments have indicated how classification technique accelerates our iteration method and converges faster to the optimal solution. This method is also more robust with respect to 'uncertain' users that make mistakes in answering queries. One drawback of this method is that at the end of the iterative algorithm, convergence speed reduces because negative advantages are produced. To improve the tail convergence speed, we must find a solution to get rid of negative advantages during the clustering process.

6.2 Solve VMDP with Polytope Λ approximately

We have presented a robust policy computation method extended to VMDP with unknown reward weights Λ by using minimax regret criterion. The robust approach attempts to find an optimal policy with respect to the bounded Λ polytope. Our new computation method is less complicated and computationally faster than previous works. The minimax regret calculation is based on Bender's decomposition method and includes two linear programs master-problem and sub-problem. The sub-problem generates some constraints for the master problem and it continues until the difference between two

regrets generated by sub-problem and master problem reaches a give precision ϵ. In our presented method, constraints have been generated by selecting random points from polytope Λ. Despite of previous approach in literature, this method does not iterate between two LPs (master problem and sub-problem), instead the master-problem is computed only once. In comparison with Bender's decomposition approach, the number of constraints for master problem are higher while master problem should be solved just once. For this reason, our method is faster and applicable on higher sizes of MDPs. On the other hand the final solution is less precise than the previous methods because this approach can not receive a precision parameter as ϵ.

6.3 Long Term Perspective and Applications

In this section we describe our perspectives related to this dissertation in the future.

A problem with ABVI algorithm (given in Section 4.5) is that for kdominance comparison methods, our algorithm adds each new constraint to the linear programming problem. All the added constraints are not necessary while they nevertheless increase complexity of algorithm. We have an idea based on machine learning methods — such as k nearest neighbors — for working on the added bound and verify them before being added to the linear program. This allows us to reduce the number of unnecessary constraints and will accelerate computation time drastically.

Another remark considers the non-dominated vectors generation (see section 4.4.2). In our setting, we have generated all non-dominated vectors offline. A first idea is to eliminate some of useless non-dominated vectors after any iteration during offline generation method. Another idea is to do generation online while communicating with users. Because having more information on reward weights helps us to update explored set of non-dominated vectors after any iteration. In fact, after eliminating part of polytope Λ, there are many explored vectors that are not non-dominated anymore and there are some new non-dominated vectors that should be added to the explored set. In the other hand, this technique can cause a reduction in the computation complexity.

Another worthwhile direction of this thesis is working on the query types proposed to the user. This means, refining our query formulation algorithms to make simpler answerable queries by users. For instance, comparing two objectives in one state is understandable

by user, while asking a user if she prefers 0.2 times of one objective rather than 0.35 times of another objective is not easy to be replied back. In our experiments we have simulated user responses automatically but for implementing our approach in communication with real users, we need to work on this part.

Our next idea is to work on theoretical aspect of non-dominated vector-valued functions research (see Section 4.4.4) such as finding more theoretical and reliable proofs.

Another question is whether we can implement our algorithms on situations with more than one user. What about a group of users who attempt to find a common decision and common optimal policy. If our algorithms are applicable on social choice concept, we think that it can be a good direction of continuing research after finishing this thesis.

Appendix A

Some Robust Approaches in MDPs with Ambiguous Transition Probabilities

The State of the Art of this thesis considers models where the ambiguity is represented by ambiguous reward functions. At least one other approach has been considered. In this Appendix, we introduce a few methods in the literature that solve MDPs with ambiguous transition probabilities using the robust approaches. It means, these methods look for an optimal solution w.r.t. bounded probability transitions among states and actions in the MDP.

In general, a solution of an MDP (either all parameters are known or they are partially known) is to find a policy with the greatest expectation of sum of rewards i.e.

$$\pi^* = \text{argmax}_{\pi \in \Pi} \, \mathbb{E}_{\pi,p}[\sum_t \gamma^t r(s)] \tag{A.1}$$

This expectation value also depends on transition probabilities between states and actions. Generally, probability functions p is not mentioned as a parameter of the expectation of rewards in Equation A.1, but we append it in the formula because p happens not to be given. In many practical problems, the transition matrices should be estimated

from data and sometimes it is not possible in practice. On the other hand, planning in MDPs sensitively relies on the transition probabilities, so that a slight error in defining probabilities may have a huge impact on final results. An example of this problem has been given by [Chades et al., 2012] w.r.t. managing a population of threatened Goldian finch birds. The response of the bird population with different management actions is uncertain and this leads to uncertain probabilities in the model.

One proposed solution in the literature to solve this problem is to find the actions that perform well on all models in the set of uncertain transition probabilities. To compute the stationary policy, the model compares the policies relying on the minimal expected reinforcement received by each policy [Bagnell et al., 2001, Nilim and El Ghaoui, 2004]:

$$\pi^* \text{ s.t. } V^{\pi^*} = \max_{\pi \in \Pi} \min_{p \in \mathcal{P}} \mathbb{E}_{p,\pi}[\sum_t \gamma^t r(s_t)]$$

Finding policies that are good respecting the unrestricted sets is computationally complex.

Proposition A.1. *Finding the stationary infinite policy that maximizes the least expected reward over an uncertainty set of MDP is NP-hard [Bagnell et al., 2001, Nilim and El Ghaoui, 2004].*

After choosing a robust computation method such as the minmax method, the issue is how to define the set of uncertain transition probabilities. Can we assume all the $p(s'|s,a)$ are confined in a polytope? Bagnell et al. [2001] consider this set as a compact and convex polytope, while Nilim and El Ghaoui [2004] take just the convex property into account. Because they claim that choosing a polytope model for the uncertainty often incurs an additional computational effort to handle the uncertainty. An exception is the Bagnell et al. [2001] work because they have described the uncertainty by an interval matrix intersected by the constrained with probabilities sum to one [Nilim and El Ghaoui, 2004].

Since an exact computation is out of reach, both Bagnell et al. [2001] and Nilim and El Ghaoui [2004] propose a robust approximation. First, Bagnell et al. [2001] have restricted ambiguous transition functions to a convex set and their reason is tractability

of this structure. They proposes a *Robust Value Iteration* algorithm on MDPs with a compact and convex uncertainty set $\mathcal{P}$. This value iteration algorithm is as below:

Enumerate Algorithm 1.

1- Initialize $\bar{V}$ to zero vector of $|S|$[1] size

2- Repeat until $\bar{V}$ converges

3- For each $s \in S,\ a \in A$

$$Q_{\min(s,a)} = \min_{p \in \mathcal{P}(.|s,a)} \mathbb{E}_p[\bar{V}(s) + \bar{r}(s)]$$

4- update $\bar{V}$ by maximizing over actions $|A|$:

$$\bar{V}(s) = \max_{a \in A} Q_{\min(s,a)}$$

Recall that $\bar{r}$[2] is a reward vector of dimension $|S|$ and Q has a scalar value. They claim the minimization problem in Algorithm 1 (step 3) on uncountable set of probability distributions is computable and also tractable. They indicated that this algorithm solves MDPs with a convex and compact set of uncertain probabilities in a polynomial time. On the other hand, Nilim and El Ghaoui [2004] demonstrate that this algorithm is NP-hard on their MDP model with a convex set on unknown probability functions $\mathcal{P}$.

Another similar approach has been examined by [Nilim and El Ghaoui, 2004]. They consider transition matrices P^a on S, indexed by the chosen action a. For sake of clarity, for a given state $s \in S$, action $a \in A$, and transition matrix P^a, the next state distribution $p(.|s,a)$ is the row related to state s in matrix P^a. Now, for each action a, a set $\mathcal{P}^a$ of admissible transition matrices is provided, so the full set of transition matrices is $\mathcal{T} = (\otimes_{a \in A}\ \mathcal{P}^a)$.

The defined robust value iteration on infinite horizon MDPs with partially known probability functions is a different formulation of maximin method too [Nilim and El Ghaoui, 2004]:

Enumerate Algorithm 2.

[1] size of the sate space

[2] in this model reward function is defined only on state $r : S \longrightarrow \mathbb{R}$

1. set $\epsilon > 0$, the initial value vector[3] $\bar{V}_1 = 0$ and iteration counter $k = 1$

2. (a) For each s, a compute the $V(s,a)$ using the given maximization problem such that

$$V(s,a) - \delta \leq \min_{p \in \mathcal{P}(.|s,a)} \bar{V}_k \cdot p \leq V(s,a)$$

where $\delta = \frac{(1-\gamma)\epsilon}{2\gamma}$

(b)For each s, a compute the $\bar{V}(s)$:

$$\bar{V}_{k+1}(s) = \max_{a \in A}(r(s,a) + \gamma V(s,a))$$

3. If $||\bar{V}_{k+1} - \bar{V}_k|| < \frac{(1-\gamma)\epsilon}{2\gamma}$ go to 4
Otherwise, $k \longleftarrow k+1$ and go to 2

4. For each $s \in S$ the optimal policy is

$$\pi^*(s) = \text{argmax}_{a \in A}\{r(s,a) + \gamma V(s,a)\} \ \ \forall\ s \in S$$

The interesting remark of their approach considering this manuscript is that they have transformed the problem to the inner scalar product between two vectors $p \in \mathcal{P}(.|s,a)$ and $\bar{V}$ (vector value at some given stage in value iteration algorithm). p and $\bar{V}$ are two $|S|$ dimensional vectors such that i-th element in $p \in \mathcal{P}(.|s,a)$ represents $p(s_i|s,a)$, thus we have:

$$V(s,a) = \bar{V} \cdot p$$

The presented ϵ precision defines an ϵ-suboptimal policy in at most $O(|S|\ |A| \log(1/\epsilon)^2)$ iterations. These methods sometimes give very pessimistic results because they search for the optimal policy satisfying all possible cases on partially known MDPs. The worst-case complexity of the robust algorithm is the same as the original Bellman recursion. Hence, robustness can be added at practically no extra computing cost.

[3] it has dimension $|S|$

Bibliography

Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In Schölkopf, B., Platt, J. C., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19*, pages 1–8. MIT Press.

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *In Proceedings of the Twenty-first International Conference on Machine Learning*. ACM Press.

Akrour, R., Schoenauer, M., and Sebag, M. (2012). APRIL: active preference-learning based reinforcement learning. *CoRR*, abs/1208.0984.

Bagnell, J. A. D., Ng, A. Y., and Schneider, J. (August, 2001). Solving uncertain markov decision problems. Technical Report CMU-RI-TR-01-25, Robotics Institute, Carnegie Mellon University.

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

Bäuerle, N. and Rieder, U. (2011). *Markov Decision Processes with Applications to Finance*. Springer Berlin Heidelberg.

Bellman, R. (1957). *A Markovian Decision Process*, volume 6. Indiana Univ. Math. J.

Benders, J. F. (2005). Partitioning procedures for solving mixed-variables programming problems. *Computational Management Science*, 2(1):3–19.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, 1st edition.

Bhattacharya, A. and Das, S. K. (2002). Lezi-update: An information-theoretic framework for personal mobility tracking in pcs networks. *Wirel. Netw.*, 8(2/3):121–135.

Boger, J., Hoey, J., Poupart, P., Boutilier, C., Fernie, G., and Mihailidis, A. (2006). A planning system based on markov decision processes to guide people with dementia through activities of daily living. *Trans. Info. Tech. Biomed.*, 10(2):323–333.

Boularias, A., Kober, J., and Peters, J. (2011). Relative Entropy Inverse Reinforcement Learning. In *Proceedings of the 14th International Con- ference on Artificial Intelligence and Statistics*, volume 15, pages 182–189.

Boutilier, C., Das, R., Kephart, J. O., Tesauro, G., and Walsh, W. E. (2003). Cooperative negotiation in autonomic systems using incremental utility elicitation. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, UAI'03, pages 89–97. Morgan Kaufmann Publishers Inc.

Boutilier, C., Patrascu, R., Poupart, P., and Schuurmans, D. (2006a). Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artif. Intell.*, 170:686–713.

Boutilier, C., Patrascu, R., Poupart, P., and Schuurmans, D. (2006b). Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artif. Intell.*, 170(8-9):686–713.

Chades, I., Carwardine, J., Martin, T. G., Nicol, S., Sabbadin, R., and Buffet, O. (2012). Momdps: A solution for modelling adaptive management problems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

Chajewska, U., Koller, D., and Parr, R. (2000). Making rational decisions using adaptive utility elicitation. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 363–369.

Fearnley, J. (2010). Strategy iteration algorithms for games and markov decision processes.

Fürnkranz, J., Hüllermeier, E., Cheng, W., and Park, S.-H. (2012). Preference-based reinforcement learning: A formal framework and a policy iteration algorithm. *Machine Learning*, 89(1-2):123–156. Special Issue of Selected Papers from ECML PKDD 2011.

Gilbert, H., Spanjaard, O., Viappiani, P., and Weng, P. (2015). Reducing the number of queries in interactive value iteration. In *Algorithmic Decision Theory - 4th International Conference, ADT 2015, Lexington, KY, USA, September 27-30, 2015, Proceedings*, pages 139–152.

Girard, J. (2014). Concurrent markov decision processes for robust robot team learning under uncertainty.

Givan, R., Leach, S. M., and Dean, T. L. (2000). Bounded-parameter markov decision processes. *Artif. Intell.*, 122(1-2):71–109.

Grudic, G. Z. and Lawrence, P. D. (1996). Human-to-robot skill transfer using the spore approximation. In *In Proc. IEEE Int. Conf. Robot. Autom. (ICRA*, pages 2962–2967.

Howard, R. A. (1960). Dynamic programming and markov processes. *MIT Press*, 69:296–297.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*, pages 267–274.

Kakade, S. M. (2003). On the sample complexity of reinforcement learning.

Klein, E., Geist, M., PIOT, B., and Pietquin, O. (2012). Inverse Reinforcement Learning through Structured Classification. In *Advances in Neural Information Processing Systems (NIPS 2012)*.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149.

Littman, M. L., Dean, T. L., and Kaelbling, L. P. (1995). On the complexity of solving markov decision problems. In *IN PROC. OF THE ELEVENTH INTERNATIONAL CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE*, pages 394–402.

McMahan, H. B., Gordon, G. J., and Blum, A. (2003). Planning in the presence of cost functions controlled by an adversary. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 536–543.

Moffaert, K. V. and Nowé, A. (2014). Multi-objective reinforcement learning using sets of pareto dominating policies. *Journal of Machine Learning Research*, 15:3663–3692.

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

Nilim, A. and El Ghaoui, L. (2004). Robustness in markov decision problems with uncertain transition matrices. NIPS.

Papadimitriou, C. H. and Yannakakis, M. (2000). On the approximability of trade-offs and optimal access of web sources. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, pages 86–.

Pažek, K. and Rozman, Č. (2009). Decision making under conditions of uncertainty in agriculture: a case study of oil crops. *Poljoprivreda*, 15(1):45–50.

Perny, P., Weng, P., Goldsmith, J., and Hanna, J. (2013). Approximation of lorenz-optimal solutions in multiobjective markov decision processes. *CoRR*.

Pietquin, O. (2013). Inverse reinforcement learning for interactive systems. In *Proceedings of the 2Nd Workshop on Machine Learning for Interactive Systems: Bridging the Gap Between Perception, Action and Communication*, MLIS '13, pages 71–75. ACM.

Pineau, J. (2004). Tractable planning under uncertainty: Exploiting structure. *PhD Thesis*.

Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Comput.*, pages 88–97.

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

Puterman, M. L. (2005). *Markov decision processes : discrete stochastic dynamic programming*. Wiley series in probability and mathematical statistics. J. Wiley & Sons, Hoboken (N. J.).

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.

Regan, K. and Boutilier, C. (2009). Regret-based reward elicitation for markov decision processes. In *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pages 444–451.

Regan, K. and Boutilier, C. (2010). Robust policy computation in reward-uncertain mdps using nondominated policies. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

Regan, K. and Boutilier, C. (2011a). Eliciting additive reward functions for markov decision processes. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2159–2164.

Regan, K. and Boutilier, C. (2011b). Robust online optimization of reward-uncertain mdps. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2165–2171.

Regan, K. and Boutilier, C. (2012). Regret-based reward elicitation for markov decision processes. *CoRR*, abs/1205.2619.

Roijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *J. Artif. Intell. Res. (JAIR)*, 48:67–113.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

Viappiani, P. and Boutilier, C. (2010). Optimal bayesian recommendation sets and myopically optimal choice query sets. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 2352–2360. Curran Associates, Inc.

Wakuta, K. (1995). Vector-valued markov decision processes and the systems of linear inequalities. *Stochastic Processes and their Applications*, 56:159 – 169.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK.

Weng, P. (2011). Markov decision processes with ordinal rewards: Reference point-based preferences. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*.

Weng, P. (2012). Ordinal Decision Models for Markov Decision Processes. In *European Conference on Artificial Intelligence*, volume 242, pages 828–833.

Weng, P. and Zanuttini, B. (2013). Interactive Value Iteration for Markov Decision Processes with Unknown Rewards. In *Proc. 23th International Joint Conference Artificial Intelligence (IJCAI2013)*, Beijing, China.

Xu, H. and Mannor, S. (2009). Parametric regret in uncertain markov decision processes. In *CDC*, pages 3606–3613. IEEE.

Ya, A. and Kane, D. (2015). walkr: Mcmc sampling from nonnegative convex polytopes.

Ziebart, B. D., Maas, A. L., Dey, A. K., and 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, pages 322–331. ACM.