Reducing the Number of Queries in Interactive Value Iteration

Hugo Gilbert, Olivier Spanjaard, Paolo Viappiani, Paul Weng

▶ **To cite this version:**

Hugo Gilbert, Olivier Spanjaard, Paolo Viappiani, Paul Weng. Reducing the Number of Queries in Interactive Value Iteration. 4th International Conference on Algorithmic Decision Theory (ADT 2015), Sep 2015, Lexington, KY, United States. pp.139-152, ⟨10.1007/978-3-319-23114-3_9⟩. ⟨hal-01213280⟩

HAL Id: hal-01213280

https://hal.science/hal-01213280v1

Submitted on 30 Jun 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

Reducing the Number of Queries in Interactive Value Iteration

Hugo Gilbert[1], Olivier Spanjaard[1], Paolo Viappiani[1], Paul Weng[2,3]
[1]Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, Paris, France
[2]SYSU-CMU Joint Institute of Engineering, Guangzhou, China
[3]SYSU-CMU Shunde International Joint Research Institute, Shunde, China
{hugo.gilbert,olivier.spanjaard,paolo.viappiani}@lip6.fr,
paweng@cmu.edu

Abstract. To tackle the potentially hard task of defining the reward function in a Markov Decision Process (MDPs), a new approach, called Interactive Value Iteration (IVI) has recently been proposed by Weng and Zanuttini (2013). This solving method, which interweaves elicitation and optimization phases, computes a (near) optimal policy without knowing the precise reward values. The procedure as originally presented can be improved in order to reduce the number of queries needed to determine an optimal policy. The key insights are that 1) asking queries should be delayed as much as possible, avoiding asking queries that might not be necessary to determine the best policy, 2) queries should be asked by following a priority order because the answers to some queries can enable to resolve some other queries, 3) queries can be avoided by using heuristic information to guide the process. Following these ideas, a modified IVI algorithm is presented and experimental results show a significant decrease in the number of queries issued.

1 Introduction

In problems of sequential decision-making under uncertainty, an agent has to repeatedly choose according to her current state an action whose consequences are uncertain in order to maximize a certain criterion in the long run. Such problems can be represented as Markov Decision Processes (MDPs) [9]. In this model, the outcome of each action is stochastic and numeric rewards are granted each time an action is performed. The numerical values of rewards are defined either by the environment or by a human user and the goal of the agent is to choose a policy (which specifies which action to take in every state) such as to maximize the expectation of the discounted sum of future rewards. The latter case, where a human must specify the reward values, represents a difficulty when using MDP

methods as this task can be cognitively hard, even for an expert user. And yet, it is well known that the optimal policy is extremely sensitive to the numerical reward values. This problem has motivated much work aiming at mitigating the burden of defining precisely the reward function. In the literature, four main approaches can be distinguished, although their boundaries may be blurry.

Robust approach. In the first approach, the parameters of the MDP (i.e., rewards and possibly also probabilities) are assumed to be imprecisely known. A natural way [2, 7] to handle such situation is to search for robust solutions, i.e., solutions that are as good as possible even in the worst case. However, this method often leads to solutions that are too pessimistic. A better approach is based on minimax regret [20]. Here, one tries to minimize the gap between the value of the best policy (after the true reward values are revealed) and that of the chosen policy. However, this leads to NP-hard problems.

Non standard decision criterion. A second approach is to change the decision criterion optimized by the agent. Different criteria have been proposed. For instance, Delage and Mannor [6] proposed to use a criterion based on quantiles (on distributions over history values). Unfortunately, by changing the decision criterion, one loses all the nice properties satisfied by the standard criterion (e.g., existence of an optimal stationary deterministic policy). Weng [16, 17] proposed two new decision criteria that could be used when only the order over rewards, but not the exact values, is known. In both cases, all or some of the nice properties of the standard criterion are preserved. However, those approaches have not yet been experimentally evaluated.

Preference learning. Another approach, which has been mainly developed for reinforcement learning (i.e., a context more general than MDP), aims at learning the reward values, either from demonstrations (live [1] or from recorded logs [8]) or from interactions with a human tutor [15]. One drawback of this approach is that it generally assumes that demonstrations from human tutors can be easily translated into the state/action representation of the learning agent, which may be difficult as humans and agents evolve in different state/action spaces.

Preference elicitation. A final approach assumes a human tutor is present and the agent may query her to get more precise information about reward values. In a series of papers, Regan and Boutilier [10–13] show how to compute policies which optimize minmax regret with respect to all candidate functions, and discuss how this criterion can be used to generate informative queries to ask the tutor about the true reward func-

tion. Iteratively issuing such queries is shown to allow convergence to the optimal policy for this function. However the problem of computing such robust policy reveals to be NP-hard [20] and their algorithm issues bound queries which can be cognitively difficult to answer. Following the same line of research, Weng and Zanuttini [18], revisited a well known algorithm for solving MDPs, *Value Iteration*, by incorporating the elicitation process in the solving procedure. In this new algorithm called *Interactive Value Iteration* (IVI), a human tutor is queried about multi-sets of rewards when the information acquired so far does not allow to continue solving the MDP. This procedure is appealing as answering comparison queries is much less cognitively demanding than giving the reward function to optimize. However, the original IVI procedure does not try to explicitly minimize the number of queries issued and may lead to a prohibitive effort for the tutor.

In this paper, we address the problem of modifying IVI in order to reduce the number of queries issued. To that aim, we propose a variation of IVI based on the ideas that 1) delaying the queries open the possibility that some of them meanwhile become unnecessary, 2) the order in which the queries are asked matters, and this order should be optimized, 3) queries can be avoided by using heuristic information to guide the iteration process. We show empirically that these combined techniques can greatly reduce the number of queries issued.

The paper is organized as follows. After recalling the main features of interactive value iteration (Section 2), we present our proposed optimizations (Section 3). Finally, we provide the results of numerical tests that show a significant decrease in the number of queries issued (Section 4).

2 Background

2.1 Markov Decision Process

A *Markov Decision Process* (MDP) [9] is defined by a tuple $\mathcal{M} = (S,A,p,r,\gamma)$ where S is a finite set of states, A is a finite set of actions, $p : S \times A \to \mathcal{P}(S)$ is a transition function with $\mathcal{P}(S)$ being the set of probability distributions over states, $r : S \times A \to \mathbb{R}$ is a reward function and $\gamma \in [0, 1[$ is a discount factor.

A (stationary, deterministic) *policy* $\pi : S \to A$ associates an action to each state. Such a policy is evaluated by a *value function* $v^\pi : S \to \mathbb{R}$ and

a *Q-function*, $Q^\pi : S \times A \to \mathbb{R}$ defined as follows:

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

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

Then a preference relation is defined over policies by: $\pi \succsim \pi' \Leftrightarrow \forall s \in S, v^\pi(s) \geq v^{\pi'}(s)$. A solution to an MDP is a policy, called *optimal policy*, that ranks the highest with respect to $\succsim$. Such a policy can be found by solving the *Bellman equations*.

$$v^*(s) = \max_{a \in A} r(s, a) + \gamma \sum_{s' \in S} p(s, a, s') v^*(s') \quad (3)$$

As can be seen, the preference relation $\succsim$ over policies is directly induced by the reward function r. In a setting where the reward function is not known with certainty, taking the maximum over actions (as done in (3)) can be problematic. In this paper we will query ordinal information from a tutor to unveil the maximal actions.

2.2 Ordinal Reward MDP

In this paper, while the rewards' numerical values are assumed to be unknown, we suppose that the order over rewards is given. Such situation can be represented as an *Ordinal Reward MDP* (ORMDP) [16] defined by a tuple $(S, A, p, \hat{r}, \gamma)$ where the reward function $\hat{r} : S \times A \to E$ takes its values in a set $E = \{r_1 < r_2 \ldots < r_k\}$ of unknown ordered rewards.

In order to count the number of each unknown reward obtained by a policy, an ORMDP can be reformulated as a Vector Reward MDP (VMDP) $(S, A, p, \overline{r}, \gamma)$ where $\overline{r}(s, a)$ is the vector in $\mathbb{R}^k$ whose i^{th} component is 1 for $\hat{r}(s, a) = r_i$, and 0 on the other components. Like in standard MDPs, in such a VMDP we can define the value function $\overline{v}^\pi$ and the Q-function $\overline{Q}^\pi$ of a policy π by :

$$\overline{v}^\pi(s) = \overline{r}(s, \pi(s)) + \gamma \sum_{s' \in S} p(s, \pi(s), s') \overline{v}^\pi(s') \quad (4)$$

$$\overline{Q}^\pi(s, a) = \overline{r}(s, a) + \gamma \sum_{s' \in S} p(s, a, s') \overline{v}^\pi(s') \quad (5)$$

where additions and multiplications are componentwise. The i-th component of a vector $v \in \mathbb{R}^k$ can be interpreted as the expected number of unknown reward r_i.

Therefore, a value function in a state can be interpreted as a multi-set or a bag of elements of E. Now comparing policies amount to comparing vectors. Interactive value iteration [18] is a procedure inspired from value iteration to find the optimal policy according to the true unknown reward function by querying when needed an expert for comparisons of two bags of elements of E. We present in the next section how this algorithm works.

2.3 Interactive Value Iteration

In order to find an (approximate) optimal policy for an ORMDP with an initially unknown preference relation over vectors, Weng and Zanuttini [18] developed a variant of value iteration, named Interactive Value Iteration (IVI), where the agent may ask a tutor which of two sequences of rewards should be preferred. An example of query is "$r_1 + r_3 \geq 2r_2$?", meaning "is receiving r_1 and r_3 as good as receiving $2r_2$?". As the tutor answers queries, the set of admissible reward functions shrinks and more vectors can be compared without querying the tutor.

At the beginning of the process, the agent only knows the order over rewards, i.e., $r_1 < r_2 < \ldots < r_k$, and knows that she can set without loss of generality r_1 to 0 and r_k to 1 [17]. This initial knowledge is represented by a set $\mathcal{K}$ of linear constraints. When having to choose the best of two value vectors $\overline{v}$ and $\overline{v}'$ in a given state (see Algorithm 2), function $StDom(\overline{v}, \overline{v}')$ compares the two vectors with respect to the following dominance relation (which is analogous to stochastic dominance):

$$\forall j = 1, \ldots, k \quad \sum_{i=j}^{k} \overline{v}_i \geq \sum_{i=j}^{k} \overline{v}'_i \tag{6}$$

In case no dominance is found, the next step is to check whether one of the two vectors is necessarily preferred to the other given the constraints in $\mathcal{K}$. The function $KDom(\overline{v}, \overline{v}')$ checks whether $\overline{v}$ is not less preferred than $\overline{v}'$ (denoted by $\overline{v} \succeq \overline{v}'$) by solving the following linear program:

$$z_K^* = \min \, (\overline{v} - \overline{v}') \cdot r \tag{7}$$

$$\text{s.t.} \quad r \in \mathcal{C}(\mathcal{K}) \tag{8}$$

where the dot in (7) denotes the inner product and $\mathcal{C}(\mathcal{K})$ the set of reward functions $r = (r_1, \ldots, r_k)$ satisfying all constraints in $\mathcal{K}$. We distinguish the following three cases: 1) A non-negative optimal value for the objective function z_K^* implies that for any possible reward function, $\overline{v} \succeq \overline{v}'$; $\overline{v}$ is then said to KDominate $\overline{v}'$. 2) In case of negativity of the optimal value z_K^*,

$KDom(\overline{v}', \overline{v})$ is called to check if $\overline{v}' \succeq \overline{v}$ for all reward function satisfying constraints in $\mathcal{K}$. 3) If the two vectors can still not be compared, the tutor is asked which of $\overline{v}$ or $\overline{v}'$ should be preferred. A query is a 2-subset $\{\overline{v}, \overline{v}'\}$ from which the tutor picks her preferred element (if indifferent, she arbitrarily picks one of them). If she picks $\overline{v}$, then $\overline{v} \succeq \overline{v}'$, otherwise $\overline{v}' \succeq \overline{v}$. If $\overline{v} \succeq \overline{v}'$ (resp. $\overline{v}' \succeq \overline{v}$), then the new constraint $(\overline{v} - \overline{v}').r \geq 0$ (resp. $(\overline{v}' - \overline{v}).r \geq 0$) is added to $\mathcal{K}$.

Algorithm 1 summarizes the IVI procedure. It uses the functions *Init*, that returns the initial set $\mathcal{K}$ of linear constraints induced by $r_1 < \ldots < r_k$, and *getBest* (Algorithm 2) that returns the best of two vectors. The function *getBest* calls first function *StDom*, then function *KDom*, and finally function query (Algorithm 3) if no dominance is found. Note that $StDom(\overline{v}, \overline{v}')$=$KDom(\overline{v}, \overline{v}')$ for initial set $\mathcal{K}$, and that $StDom(\overline{v}, \overline{v}') \Rightarrow KDom(\overline{v}, \overline{v}')$ in all cases. Nevertheless, the IVI procedure takes advantage of the computational efficiency of *StDom* to save some calls to *KDom*.

Weng and Zanuttini [18] showed that the number of queries used by IVI is polynomial in the size of the MDP. However, as the tutor is queried each time two vectors cannot be compared, it seems that a more sophisticated approach could greatly reduce the number of queries needed by the algorithm. In the next section, we propose a modified version of IVI that integrates several techniques in order to reduce the number of queries.

Algorithm 1: IVI [18]

```
   Data: S, A, p, r̂, γ, E, ε
   Result: v̄_t
1  t ← 0
2  compute r̄ from r̂
3  K ← Init(E)
4  for s ∈ S do v̄_0(s) ← (0, ..., 0)
5  repeat
6  |   t ← t + 1
7  |   for s ∈ S do
8  |   |   best ← (0, ..., 0)
9  |   |   for a ∈ A do
10 |   |   |   v̄ ← r̄(s, a) + γ Σ_{s'∈S} p(s, a, s') v̄_{t−1}(s')
11 |   |   |   (best, K) ← getBest(best, v̄, K)
12 |   |   v̄_t(s) ← best
13 until ||v̄_t − v̄_{t−1}|| < ε
14 return v̄_t
```

Algorithm 2: getBest

```
Data: v̄, v̄', K
Result: (v, K)
if StDom(v̄, v̄') then return (v̄, K)
if StDom(v̄', v̄) then return (v̄', K)
if KDom(v̄, v̄', K) then return (v̄, K)
if KDom(v̄', v̄, K) then return (v̄', K)
(best, K) ← query(v̄, v̄', K)
return (best, K)
```

Algorithm 3: query

```
Data: v̄, v̄', K
Result: (v, K)
Ask query {v̄, v̄'}
if v̄ ⪰ v̄' then
|   return (v̄, K ∪ {(v̄ − v̄').r ≥ 0})
else
    return (v̄', K ∪ {(v̄' − v̄).r ≥ 0})
```

3 Modified Interactive Value Iteration

Interactive Value Iteration is appealing for users as answering comparison queries is significantly less cognitively demanding than directly defining the reward function. And yet to be a reasonable alternative, the number of queries needs to be as low as possible. As stated in the introduction, several ideas are investigated to address this problem: 1) delaying queries in order to avoid asking some unnecessary queries; 2) prioritizing queries in order to ask the most informative ones first; 3) allowing small mistakes in the dominance tests in the early stages in order to anticipate the shrinking of the set of admissible reward functions.

3.1 Delaying the queries

In Algorithm 1, the tutor is queried whenever two vectors $\overline{v}$ and $\overline{v}'$ cannot be compared. The intuition behind the improvement that we propose is that by delaying the query phase after all vectors (each vector refers to a possible action in a given state) are generated we might benefit from the fact that new dominance relations might appear later on. Indeed when trying to assess which of three vectors $\overline{v}_1, \overline{v}_2$, and $\overline{v}_3$ is the best, we may be able to find that $\overline{v}_3$ dominates both $\overline{v}_1$ and $\overline{v}_2$; this is enough for us, even if we ignore which is better between $\overline{v}_1$ and $\overline{v}_2$. Therefore querying which of $\overline{v}_1$ and $\overline{v}_2$ is best, as would do Algorithm 1, is unnecessary. Thus, delaying the querying step from the loop over actions will prevent asking some unnecessary queries. For this purpose, we replace Lines 8 to 13 in Algorithm 1 by the following lines:

for $s \in S$ **do**
    $\overline{Q}(s) \leftarrow \emptyset$
    for $a \in A$ **do**
        $\overline{Q}(s,a) \leftarrow \overline{r}(s,a) + \gamma \sum_{s' \in S} p(s,a,s')\overline{v}_{t-1}(s')$
        $\overline{Q}(s) \leftarrow \overline{Q}(s) \cup \{\overline{Q}(s,a)\}$
    $\overline{Q}(s) \leftarrow \text{StDFilter}(\overline{Q}(s))$
    $\overline{Q}(s) \leftarrow \text{KDFilter}(\overline{Q}(s))$
    while $|\overline{Q}(s)| > 1$ **do**
        Let $\{\overline{v}, \overline{v}'\} \subseteq \overline{Q}(s)$
        $(_, \mathcal{K}) \leftarrow \text{query}(\overline{v}, \overline{v}', \mathcal{K})$
        $\overline{Q}(s) \leftarrow \text{KDFilter}(\overline{Q}(s))$
    $\overline{v}_t(s) \leftarrow$ unique vector of $\overline{Q}(s)$

where primitives *StDFilter* and *KDFilter* are given in Algorithms 4 and 5: the former checks the dominance described by Equation 6 for each pair of vectors $\overline{v}, \overline{v}'$ in $\overline{Q}$ and returns the set of undominated vectors; the latter does the same thing for K-dominance.

Basically, $\overline{Q}(s)$ is a set of vectors (discounted collections of unknown rewards) associated to a state s while $\overline{Q}(s,a)$ is related to the value of taking an action a in state s. As in the original IVI, we filter out dominated

and K-dominated vectors before starting the querying phase. Finally, in the **while** loop we query the user about pairs of non-dominated vectors in $\overline{Q}(s)$ until $\overline{Q}(s)$ contains a single element, that is assigned to $\overline{v}_t(s)$ in the last line. This idea of delaying the queries can be pushed further by delaying the querying phase out of the loop over states or by waiting several time steps before asking queries. We will return to this point when discussing future works in Section 5.

Algorithm 4: StDFilter

```
Data: Q̄
Result: Filtered Q̄
for v̄ ∈ Q̄ do
    for v̄' ∈ Q̄ with v ≠ v' do
        if StDom(v̄, v̄') then
            Q̄ ← Q̄ \ {v̄'}
return Q̄
```

Algorithm 5: KDFilter

```
Data: Q̄
Result: Filtered Q̄
for v̄ ∈ Q̄ do
    for v̄' ∈ Q̄ with v ≠ v' do
        if KDom(v̄, v̄') then
            Q̄ ← Q̄ \ {v̄'}
return Q̄
```

3.2 Prioritizing the queries

By delaying the queries out of the loop over actions and even out of the loop over states, one can choose to ask queries in a different order than the sequential one induced by the original IVI procedure. Certainly this order will count as queries are not all equally informative. Thus queries that we presume might solve many others should be asked first. Defining a relevance score to guide the querying process seems to be a promising technique to curb the number of queries necessary to solve the MDP. For this purpose, we now replace Lines 8 to 13 in Algorithm 1 by the following lines:

```
for s ∈ S do
    Q̄(s) ← ∅
    for a ∈ A do
        Q̄(s, a) ← r̄(s, a) + γ Σ_{s'∈S} p(s, a, s') v̄_{t−1}(s')
        Q̄(s) ← Q̄(s) ∪ {Q̄(s, a)}
    Q̄(s) ← StDFilter(Q̄(s))
    Q̄(s) ← KDFilter(Q̄(s))
Queries ← ⋃_s {{v̄, v̄'} ⊆ Q̄(s) : v̄ ≠ v̄'}

while Queries ≠ ∅ do
    {v̄, v̄'} ← arg max{X-score({v̄, v̄'}) : {v̄, v̄'} ⊆ Queries}
    (_, K) ← query(v̄, v̄', K)
    for s ∈ S do Q̄(s) = KDFilter(Q̄(s))
    Queries ← ⋃_s {{v̄, v̄'} ⊆ Q̄(s) : v̄ ≠ v̄'}
```

where function $\mathcal{X}$-score (with $\mathcal{X} \in \{\mathcal{Q}, \mathcal{K}, \mathcal{S}\}$) is one of the priority functions described below, the value of which is intended to reflect how informative a query is, and *Queries* is the set of all unsolved queries over S.

Note that here the best vector returned by query($\overline{v}, \overline{v}', \mathcal{K}$) does not need to be saved.

To define priority functions evaluating the informative value of a query, several methods could be considered. Here we propose two types of heuristics. The first type aims at reducing as much as possible the set of remaining unsolved queries (*i.e.*, focusing on the impact on the cardinality of *Queries*, the set of unsolved queries) while the second type tries to reduce as much as possible the set of admissible reward functions (*i.e.*, focusing on the impact on the polytope $\mathcal{C}(\mathcal{K})$).

Strategies aiming at reducing the cardinality of set Queries. Let $\{\overline{v}, \overline{v}'\}$ be a query. If we know that, for instance, $\overline{v}$ is preferred to $\overline{v}'$, this induces an additional constraint that reduces the polytope $\mathcal{C}(\mathcal{K})$. This new, smaller, polytope may induce new relations of K-dominance — for instance, we might be able to check that for all $r \in \mathcal{C}(\mathcal{K} \cup \{\overline{v} \succeq \overline{v}'\})$ a vector $\overline{v}_1$ is preferred to another vector $\overline{v}_2$ — in other words the information carried in an answer to a query can generalize to other queries. A natural idea to evaluate the information value of a query is then to count the number of queries that can be resolved if $\overline{v} \succeq \overline{v}'$, the number of queries resolved if, on the other hand, $\overline{v}' \succeq \overline{v}$ and to take the minimum between the two values.This relevance score will be called $\mathcal{Q}$-score ($\mathcal{Q}$ for *Queries*). Let $Q_{val}(\overline{v}, \overline{v}')$ be the number of queries decided if $\overline{v} \succeq \overline{v}'$; then:

$$\mathcal{Q}\text{-score}(\{\overline{v}, \overline{v}'\}) = \min\{Q_{val}(\overline{v}, \overline{v}'), Q_{val}(\overline{v}', \overline{v})\}. \tag{9}$$

This idea is simple and natural but comes at the cost of solving $4(N-1)$ linear programs (one for each call to KDom) in the worst case if N is the number of queries.

Strategies aiming at directly reducing polytope $\mathcal{C}(\mathcal{K})$. Another idea is to use the optimal objective values given by procedure KDom. Consider a query $\{\overline{v}, \overline{v}'\} \in$ *Queries*. Let $\mathcal{K}_{val}(\overline{v}, \overline{v}')$ be the optimal value of the linear program described by Equations 7-8, normalized by $||\overline{v} - \overline{v}'||$. Since neither $\overline{v}$ nor $\overline{v}'$ could be filtered, both $\mathcal{K}_{val}(\overline{v}, \overline{v}')$ and $\mathcal{K}_{val}(\overline{v}', \overline{v})$ are necessarily negative. We define the priority of query $\{\overline{v}, \overline{v}'\}$ as

$$\mathcal{K}\text{-score}(\{\overline{v}, \overline{v}'\}) = \min\{|\mathcal{K}_{val}(\overline{v}, \overline{v}')|, |\mathcal{K}_{val}(\overline{v}', \overline{v})|\}. \tag{10}$$

The idea of $\mathcal{K}$-score is that $\mathcal{K}_{val}(\overline{v}, \overline{v}')$ and $\mathcal{K}_{val}(\overline{v}', \overline{v})$ give us an approximation of the volume of the polytope on both sides of the constraint defined by the query. Thus it is likely that a high $\mathcal{K}$-score query will reduce largely the polytope no matter the answer of the tutor. This alternative

has the benefits of its algorithmic simplicity. Indeed the computation of $\mathcal{K}_{val}(\overline{v}, \overline{v}')$ and $\mathcal{K}_{val}(\overline{v}', \overline{v})$ only requires to solve small linear programs.

Alternatively, following the same idea, we sampled rewards in the admissible reward space (using a Gibbs sampler [5]). Let $\mathcal{SR}$ be the set of samples generated and consider a query $\{\overline{v}, \overline{v}'\} \in \mathit{Queries}$. Let $\mathcal{S}_{val}(\overline{v}, \overline{v}') = |\{r \in \mathcal{SR} : (\overline{v} - \overline{v}').r \geq 0\}|$. We define the priority of query $\{\overline{v}, \overline{v}'\}$ as its $\mathcal{S}$-score ($\mathcal{S}$ for sampling):

$$\mathcal{S}\text{-score}(\{\overline{v}, \overline{v}'\}) = \min\{\mathcal{S}_{val}(\overline{v}, \overline{v}'), \mathcal{S}_{val}(\overline{v}', \overline{v})\}. \quad (11)$$

The numbers of samples on each side of the hyperplane defined by the query $\{\overline{v}, \overline{v}'\}$ gives us an approximation of the volumes of the polytope on each side of the query. Hence a query which has roughly 50% of samples on both sides will approximately cut the polytope in two equal volumes and it will be deemed a very informative query according to this strategy. A similar idea was used by Rosenthal and Veloso [14] in a context where rewards are a weighted sum of known subrewards and the elicitation procedure searches for the unknown weights.

3.3 Allowing small mistakes in the early stages

The modification of IVI we propose in this subsection aims at making a compromise between the number of iterations of the procedure and the number of queries issued. The idea is to take the "risk" of slowering convergence by avoiding as much as possible to ask queries in the early stages. For this purpose, the condition $z_K^* \geq 0$ in function *KDom* (we recall that z_K^* corresponds to the optimal value of program (7-8)) is loosened: a vector $\overline{v}'$ is considered to be dominated if $z_K^* \geq -\operatorname{err}(t)$, where $\operatorname{err}(t) \geq 0$ is a function that decreases to 0 with t. Clearly, this trick has the potentiality to avoid many queries during the early stage with the possible drawback of temporarily driving IVI towards a misleading direction. To insure that this modification will not prevent the algorithm to converge towards the optimal value function, we modify the main loop of IVI so that the algorithm will keep running until $\operatorname{err}(t) \leq \delta$ with $\delta << 1$.

3.4 Synthesis

Algorithm 6 synthesizes our modifications to IVI. The initialization of the algorithm (lines 1 to 4) is unchanged. The main loop (lines 5 to 22) iterates until the value function converges to the optimal value function and the $\operatorname{err}(t)$ function converges to 0. From line 7 to line 13, we fill the sets

of possible value vectors for each state by computing and appending the Q-values of the corresponding state-action pairs (lines 7 to 11) and then filter out the vectors (lines 12 and 13) that we already know are dominated by using functions *StDFilter* (Algorithm 4) and *KDFilter* (Algorithm 5). This latter algorithm now takes an extra parameter (i.e., $\text{err}(t)$) for implementing the idea presented in the previous subsection. In the second part of the loop we consider the set of all unsolved queries *Queries* composed of pairs of non-dominated vectors of a same state. While there exists unsolved queries we select and issue the most informative query (lines 16 and 17) using the priority score, $\mathcal{X}$-score ($\mathcal{X} \in \{\mathcal{Q}, \mathcal{K}, \mathcal{S}\}$). Once the tutor answered the query, the acquired information may enable to filter other vectors (lines 18) thus reducing the number of unsolved queries (line 19). Once all the queries are solved, each set $\overline{Q}(s)$ is composed of a single element, corresponding to $\overline{v}_t(s)$ (the value vector of s for the next time step). The optimal value function for $\mathcal{M} = (S,\ A,\ p,\ \hat{\text{r}},\ \gamma)$ is returned. By using standard bookkeeping techniques, we could return the optimal policy as well.

Algorithm 6: Modified IVI

```
   Data: S, A, p, r̂, γ, E, ε, err
   Result: v̄_t
1  t ← 0
2  compute r̄ from r̂
3  K ← Init(E)
4  for s ∈ S do  v̄_0(s) ← (0, ..., 0)
5  repeat
6      t ← t + 1
7      for s ∈ S do
8          Q̄(s) ← ∅
9          for a ∈ A do
10             Q̄(s, a) ← r̄(s, a) + γ Σ_{s'∈S} p(s, a, s') v̄_{t−1}
11             Q̄(s) ← Q̄(s) ∪ {Q̄(s, a)}
12         Q̄(s) ← StDFilter(Q̄(s))
13         Q̄(s) ← KDFilter(Q̄(s), err(t))
14     Queries ← ⋃_s {{v̄, v̄'} ⊆ Q̄(s) : v̄ ≠ v̄'}
15     while Queries ≠ ∅ do
16         {v̄, v̄'} ← arg max{X-score({v̄, v̄'}) : {v̄, v̄'} ⊆ Queries}
17         (_, K) ← query(v̄, v̄', K)
18         for s ∈ S do  Q̄(s) = KDFilter(Q̄(s), err(t))
19         Queries ← ⋃_s {{v̄, v̄'} ⊆ Q̄(s) : v̄ ≠ v̄'}
20     /*each Q̄(s) is now a singleton*/
21     for s ∈ S do  v̄_t = Q̄(s)
22 until ||v̄_t − v̄_{t−1}|| < ε and err(t) < δ
23 return v̄_t
```

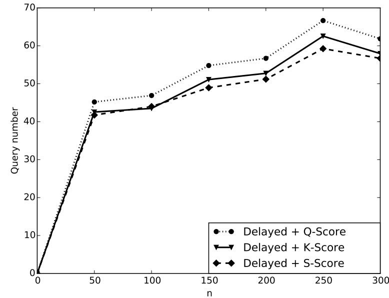

Fig. 1: Number of queries vs number of states for each priority scores.

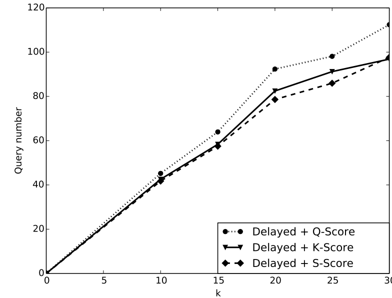

Fig. 2: Number of queries vs number of rewards for each priority scores.

4 Numerical Tests

We tested our approach on three different domains: randomly generated MDPs, autonomic computing [4] and a simulated setting of personalized assistance to impaired people (Coach domain [3])[1]. The original IVI and the improved version described in Section 3 were coded in python using Gurobi 6.0 as an LP solver. The discount factor γ is set to 0.95, ϵ to 10^{-3}, δ to 10^{-7} and $\mathrm{err}(t) = e^{-t}$. The number of samples used by the $\mathcal{S}$-score is 5000. All numeric results are averaged over 20 runs.

Random MDPs. We first compared IVI and different variations of our improved IVI on randomly generated MDPs; Given fixed n, m, k (the numbers of states, of actions and of different types of rewards), we randomly generate the transition function assuming that each pair (s, a) has $\lfloor \log_2(n) \rfloor$ successors (chosen uniformly from the set of states) and transition probabilities are obtained by sampling between 0 and 1 and then normalizing. The type of reward of each pair (s, a) is picked from the uniform categorical distribution $r_1, \ldots, r_k$; the numerical values are randomly generated in interval $[0, 1]$ and reordered in order to be consistent.

In Figures 1 (the number of queries asked as a function of n; $k = |E|$ is fixed to 10 and m to 5) and 2 (the number of queries asked as a function of the number $|E|$ of different ordinal rewards; n is fixed to 50 and m to 5) we compare the different priority scores that can be used to choose the next query to ask; the graph shows that all three techniques ($\mathcal{Q}$-score, $\mathcal{K}$-score and $\mathcal{S}$-score) are similarly effective with $\mathcal{S}$-score performing best.

[1] In both the autonomic computing domain and in Coach, we randomly generated the transition values and the rewards in such a way to satisfy the constraints imposed by the problem domain.

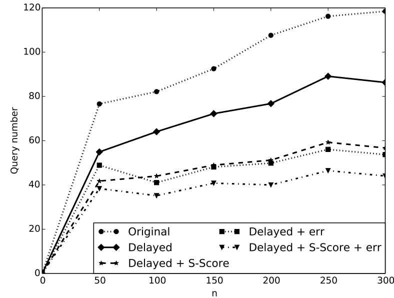

Fig. 3: Number of queries vs number of states for different query strategies.

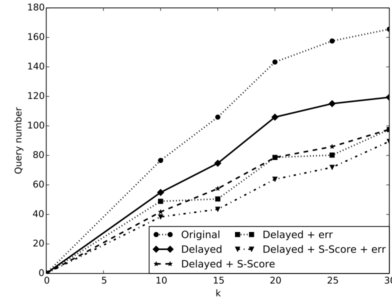

Fig. 4: Number of queries vs number of rewards for different query strategies.

In Figure 3 we compare the impact of the different improvements that we described in Subsections 3.1-3.3 with the performance of the original IVI ($k = |E|$ is fixed to 10 and m to 5). For the queries' priority we focus on the $\mathcal{S}$-score since this seemed to be the best performing strategy. As expected, IVI asks the highest number of queries (around 80 with 100 states; 120 with 300 states); delaying the moment of asking queries already gives a very significant advantage (around 60 and 85 queries for 100, 300 states). If we additionally ask queries according to the priority induced by the $\mathcal{S}$-score, the number of queries reduces even more (around 45 and 55 queries for 100, 300 states); but surprisingly this improvement is less than the improvement obtained by the combination of delaying the queries and the heuristic of allowing small errors in the initial iterations.The best results are obtained by combining the $\mathcal{S}$-score with the "error" heuristic (about 40 queries with 300 states). Interestingly, with our improvements, the number of queries asked by modified IVI grows very slowly with respect to the number of states.

Figure 4 compares the same strategies but with different numbers of rewards (i.e., $k = |E|$), the number of states n being fixed to 50 and m to 5. By comparing Figure 3 with Figure 4 we can see that parameter k impacts much more the number of queries than parameter n; especially when considering the version of IVI including all our improvements (denoted as "Delayed + $\mathcal{S}$-score + err" in the plots).

Autonomic Computing. We also applied our algorithm on the domain of autonomic computing [4]. In this domain, we assume there are κ application server elements on which N available resources have to be assigned. A feasible allocation is an integer vector $\overline{a} = (a_1, \ldots, a_\kappa)$ with $\sum_{i=1}^{\kappa} a_i \leq N$. The client's demand (changing over time) is an integer vector $\overline{d} = (d_1, \ldots, d_\kappa)$ representing κ levels of demand in $\{1, \ldots, D\}$. A state

of the MDP is a vector $(\overline{a}, \overline{d})$ defining current allocation and demand. An action is the adoption of a new allocation $\overline{m} = (m_1, \ldots, m_\kappa)$. The reward of taking action $\overline{m}$ in state $(\overline{a}, \overline{d})$ is $r((\overline{a}, \overline{d}), \overline{m}) = u(\overline{a}, \overline{d}) - c(\overline{a}, \overline{d}, \overline{m})$ where $u(\overline{a}, \overline{d}) = \sum_{i=1}^{\kappa} u(n_i, d_i)$ is the sum of non-decreasing utility-functions $u(a_i, d_i)$ and $c(\overline{a}, \overline{d}, \overline{m})$ is the sum of the costs for removing a resource unit from the server. An action deterministically sets the next allocation, while the uncertainty about demands is stochastic and exogenous.

We ran IVI and modified IVIs on instances with $\kappa = 2$, $N = 3$ and $D = 3$ (90 states and 10 actions). While the original IVI needs 188.7 queries to converge, by delaying the queries we reduce this number to 108.9. Prioritizing the order in which the queries are asked further reduces the number of queries to 86.9. Finally by using the heuristic that allows small mistakes in the first iterations of the algorithm, we only need to ask 66.3 queries.

Coach. Finally, we present our experimental results on the "Coach" domain [3]. In this problem, we provide assistance to a person with dementia accomplishing a daily-life activity (e.g., handwashing) that is decomposed into $T = \{0, \ldots, l\}$ phases. Different types of aids are available, modeled by actions $A = \{0, \ldots, m\}$; $a \in A$ is a form of assistance, each associated with a different level of intrusiveness between 0 and m; 0 represents no prompt (no aid is given), $m - 1$ represents the most intrusive prompt and m means that a caregiver has to be called. The goal is to aid the person in completing the task, with enough aid but avoiding being too intrusive.

A state in the MDP is described as a tuple (t, d, f) where $t \in T$ is the current timestep, $d \in D = \{0, \ldots, 5\}$ is the delay (time already spent in the current phase of the task) and $f \in A$ is the last prompt used. Transitions model the chance of "success", i.e., the probability that the person moves to the next phase. To model the effectiveness of the aid, at each phase $t < l$ of the task, the probability of success is increasing with the level of intrusiveness of the action a; however the probability is decreasing with d. The reward associated to taking action a in state (t, d, f) is defined by $r((t, d, f), a) = r_{goal}(t) + r_{progress}(d) + r_{delay}(d) + r_{prompt}(a)$ where $r_{goal}(t)$ gives a large reward when the final phase is reached and 0 otherwise, $r_{progress}(d)$ is a small reward when passing to the next phase with no delay and 0 otherwise, $r_{delay}(d)$ and $r_{prompt}(a)$ are increasing cost functions.

We ran IVI and our improved versions of IVI on instances with $l = 14$ and $m = 6$ (630 states and 7 actions). The original version of IVI needed 169.9 queries to converge. By delaying the queries we reduced this number to 114.7. Prioritizing the order in which the queries where issued curbed

this score to 77.9. Lastly by allowing small mistakes at the beginning of the algorithm the number of queries issued decreased to 71.2.

5 Conclusion and Future Works

IVI is an appealing procedure that mitigates the burden of defining the reward function of an MDP by interweaving the elicitation and resolution phases. In order to find an optimal policy for an MDP, this procedure queries the tutor about comparisons of multisets of rewards when needed. This paper presents modifications to the original algorithm that are shown to reduce substantially the number of queries issued. The main ideas of the paper are that we can avoid unnecessary queries by delaying the querying phase, and reasoning about the order in which we ask the query.

A natural extension of our work would be to explore new priority scores to guide the querying process. For instance, a strategy alternative to the ones proposed in Section 3.1 would be to work in the space of differences of value vectors $\overline{v} - \overline{v}'$. Points $\overline{v} - \overline{v}'$ for which $\overline{v} \succeq \overline{v}'$ (resp. $\overline{v}' \succeq \overline{v}$) would be labelled $+$ (resp. $-$) and an SVM method would be used to find the hyperplane (going through point $\overline{0}$) best separating $+$ and $-$ labels. The vector orthogonal to this hyperplane can be interpreted as the most likely reward function given $\mathcal{K}$. The next query would then be the unsolved query closest to this hyperplane.

Additionally, we intend to delay even more the querying phase by waiting several time steps before asking queries. In this setting (similar to the one of multiobjective MDPs [19]), sets $\overline{Q}(s)$ would not reduce to a singleton at the end of each iteration. Our preliminary results in this direction (where we delay over 3 time steps) are promising and lead to an even more important reduction of the number of queries. Indeed only ≈ 25 queries are needed to solve a random MDP with 50 states, 5 actions and 10 ordinal rewards (results averaged on 20 runs). However, the number of possible value vector for each state can easily explode in this setting and we need to adapt our algorithm to prevent this from happening.

Acknowledgments. Work supported by the French National Research Agency through the Idex Sorbonne Universites, ELICIT project under grant ANR-11-IDEX-0004-02.

References

1. Abbeel, P., Ng, A.: Apprenticeship Learning via Inverse Reinforcement Learning. In: Proc. Twenty-first Inter. Conf. on Machine Learning. ICML '04, ACM, New York, NY, USA (2004)

2. Bagnell, J., Ng, A., Schneider, J.: Solving uncertain Markov Decision Processes. Tech. rep., CMU (2001)
3. Boger, J., Hoey, J., Poupart, P., Boutilier, C., Fernie, G., Mihailidis, A.: A planning system based on Markov decision processes to guide people with dementia through activities of daily living. IEEE Transactions on Information Technology in Biomedicine p. 2006
4. Boutilier, C., Das, R., Kephart, J.O., Tesauro, G., Walsh, W.E.: Cooperative Negotiation in Autonomic Systems Using Incremental Utility Elicitation. In: In Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence. pp. 89–97 (2003)
5. Casella, G., George, E.I.: Explaining the Gibbs sampler. The American Statistician 46, 167–174 (1992)
6. Delage, E., Mannor, S.: Percentile optimization in uncertain Markov decision processes with application to efficient exploration. In: ICML. pp. 225–232 (2007)
7. Givan, R., Leach, S., Dean, T.: Bounded-parameter Markov decision process. Artif. Intell. 122(1-2), 71–109 (2000)
8. Piot, B., Geist, M., Pietquin, O.: Boosted and Reward-regularized Classification for Apprenticeship Learning. In: 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014). Paris, France (2014), (accepted, to appear)
9. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, USA, 1st edn. (1994)
10. Regan, K., Boutilier, C.: Regret-based Reward Elicitation for Markov Decision Processes. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence. pp. 444–451. UAI '09, AUAI Press, Arlington, Virginia, United States (2009)
11. Regan, K., Boutilier, C.: Robust Policy Computation in Reward-Uncertain MDPs Using Nondominated Policies. In: Fox, M., Poole, D. (eds.) AAAI. AAAI Press (2010)
12. Regan, K., Boutilier, C.: Eliciting Additive Reward Functions for Markov Decision Processes. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three. pp. 2159–2164. IJCAI'11, AAAI Press (2011)
13. Regan, K., Boutilier, C.: Robust Online Optimization of Reward-uncertain MDPs. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three. pp. 2165–2171. IJCAI'11, AAAI Press (2011)
14. Rosenthal, S., Veloso, M.M.: Monte Carlo preference elicitation for learning additive reward functions. In: RO-MAN. pp. 886–891. IEEE (2012)
15. Thomaz, A., Hoffman, G., Breazeal, C.: Real-Time Interactive Reinforcement Learning for Robots. In: AAAI Workshop Human Comprehensible Machine Learning. pp. 9–13 (2005)
16. Weng, P.: Markov Decision Processes with Ordinal Rewards: Reference Point-Based Preferences. In: Proc. of the 21st Inter. Conf. on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011 (2011)
17. Weng, P.: Ordinal Decision Models for Markov Decision Processes. In: ECAI 2012 - 20th Eur. Conf. on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012. pp. 828–833 (2012)
18. Weng, P., Zanuttini, B.: Interactive Value Iteration for Markov Decision Processes with Unknown Rewards. In: Rossi, F. (ed.) IJCAI. IJCAI/AAAI (2013)

19. White, D.J.: Multi-objective infinite-horizon discounted Markov decision processes. Journal of Mathematical Analysis and Applications 89(2) (1982)
20. Xu, H., Mannor, S.: Parametric regret in uncertain Markov decision processes. In: CDC. pp. 3606–3613. IEEE (2009)