WCCI 2012 IEEE World Congress on Computational Intelligence
June, 10-15, 2012 - Brisbane, Australia IJCNN

Tree-based Fitted Q-Iteration for Multi-Objective Markov Decision Problems

Andrea Castelletti, Francesca Pianosi and Marcello Restelli
Department of Electronics and Information
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 - Milan, Italy
Email: {castelle,pianosi,restelli}@elet.polimi.it

***Abstract*—This paper is about solving multi-objective control problems using a model–free batch–mode reinforcement–learning approach. Although many real–world applications have several conflicting objectives, reinforcement–learning (RL) literature has mainly focused on single-objective control problems. As a consequence, in the presence of multiple objectives, the usual approach is to consider many single–objective control problems (resulting from different combinations of the original problem objectives), each one solved using standard RL techniques. The algorithm proposed in this paper is an extension of Fitted Q–iteration (FQI) that enables to learn the control policies for all the linear combinations of preferences (weights) assigned to the objectives in a single training process. The key idea of multi–objective FQI (MOFQI) is to enlarge the continuous approximation of the action–value function, which is performed by single–objective FQI over the state–action space, also to the weight space. The approach is demonstrated on an interesting real–world application for multi–objective RL algorithms: the optimal operation of a multi–purpose water reservoir.**

I. Introduction

Multi–objective control problems are quite common in many application fields, including economic systems, water resources systems, mechatronic and robotic systems. These problems are often modeled as Multi–objective Markov Decision Processes (MOMDPs).The conventional approach to solving MOMDPs is to convert the problems themselves into a single–objective optimization, combining all the objective functions into a single functional form that can be handled by any standard single–objective control design method. A well–known combination is the linear combination of the objectives, known as *weighted–sum method*: many single–objective problems associated with different values of the weights are solved and a subset of the theoretical Pareto–optimal solutions to the multi–objective problem is obtained. With the growth in the number of the objectives, the repetitions of single–objective problems scale exponentially, thus making the approach computationally intensive, if not prohibitive. With the development of bio–inspired optimization methods, a number of alternative algorithms have been designed to directly solve the multi–objective control problem by simultaneously handling all the objectives [1]. The common basic idea to all these methods is to formulate the control problem as a simulation–based optimization of the policy parameters within a given class of functions and to generate a subset of the theoretical Pareto–optimal solutions in one single run. Whilst also these approaches suffer from some computational burden associated with the number of objectives considered, for problems with more than two objectives they are definitely more efficient than any other method based on the resolution of multiple single–objective problems. However, as the number of objectives grows, the number of simulation runs they require to produce an acceptable approximation of the Pareto frontier might be considerably high and computationally intractable.

Multi–objective Reinforcement Learning (MORL) is recently emerging as a potentially interesting alternative to the above two approaches to efficiently design multi–objective control policies [2], [3], [4], [5], [6], [7]. Among the works so far published under the MORL umbrella, the most related to the approach presented in this paper is the one presented in [7], which reduces the time– and space–complexity of the Convex Hull Value Iteration algorithm and extends it to continuous–state problems (using linear function approximation) where the model is unknown and batch data are used. Nonetheless, the complexity of the algorithm becomes overwhelming when the number of objectives is larger than three. For a recent survey of the field, we refer the reader to [8].

In this paper we present a novel MORL algorithm, which is a multi–objective extension of the single–objective Fitted Q-Iteration (FQI) algorithm [9]. The key idea of multi–objective FQI (MOFQI) is to enlarge the continuous approximation of the action–value function, which FQI performs over the state–control space, also to the weight space by including new variables (the weights) within the arguments of the action–value function. As a result, MOFQI is able to approximate, with a single learning process, all the policies associated to any convex linear combination of the different objectives. In this way, it is possible to exploit the benefits (inherited from the FQI algorithm) of working with any kind of regression algorithm. The properties of MOFQI are first evaluated by application to a numerical Test case study of a two–objective reservoir system. Pareto–optimal operating policies designed by tree-based MOFQI are compared with those generated by several runs of tree-based FQI for different linear combinations of the objectives, and the nearly optimal solution provided by Stochastic Dynamic Programming. The potential of the proposed MOFQI approach is subsequently explored by ap-

plication to a real–world case study, the operation of the Hoa Binh reservoir in Northern Vietnam.

II. Multi-Objective Markov Decision Processes

A discrete–time continuous Markov Decision Process (MDP) is described as a tuple $\langle X, U, P, R, \gamma, \mu \rangle$, where $X \subset \mathbb{R}^n$ is the continuous state space, $U \subset \mathbb{R}^m$ is the continuous action space, $P(y|x,u)$ is the transition model that defines the transition density between state x and y under action u, $R(x,u,y)$ is a reward function that specifies the instantaneous reward when state y is reached by taking action u in state x, $\gamma \in [0,1)$ is a discount factor, and μ is the initial–state distribution from which the starting state is drawn. The policy is characterized by a density distribution $\pi(u|x)$ that specifies the probability of taking action u in state x. The *value* of a state x under a policy π is the expected return when starting in x and following π thereafter:

$$V^{\pi}(x) = \int_U \int_X \left(R(x,u,y) + \gamma V^{\pi}(y) \right) P(dy|x,u) \pi(du|x).$$

Solving an MDP means to find a policy that maximizes the above function in each state (called *value function*). The optimal value function is the solution of the Bellman optimality equation:

$$V^*(x) = \max_{u \in U} \int_X \left(R(x,u,y) + \gamma V^*(y) \right) P(dy|x,u).$$

Given the initial–state distribution μ, it is possible to define the maximum expected return:

$$J^*_{\mu} = \int_X V^*(x) \mu(dx). \tag{1}$$

To simplify notation, in the following subscript μ will be omitted where possible. For control purposes, it is better to consider action values, i.e., the value of taking action u in state x and following a policy π thereafter. The optimal action–value function is the solution of the following equation:

$$Q^*(x,u) = \int_X \left(R(x,u,y) + \gamma \max_{u' \in U} Q^*(y,u') \right) P(dy|x,u).$$

Any MDP has at least one deterministic stationary optimal policy that takes in each state the action with the highest value, $\pi^*(x) = \arg\max_{u \in U} Q^*(x,u)$.

Multi-Objective MDPs (MOMDPs) are an extension of the MDP model, where several pairs of reward functions and discount factors are defined, one for each objective. Formally, a MOMDP is described as a tuple $\langle X, U, P, \mathbf{R}, \boldsymbol{\gamma}, \mu \rangle$, where $\mathbf{R} = [R_1, \ldots, R_q]$ and $\boldsymbol{\gamma} = [\gamma_1, \ldots, \gamma_q]$ are q-dimensional vectors of reward functions and discount factors respectively. In MOMPDs, any policy π is associated to q value functions $\mathbf{V}^{\pi} = [V_1^{\pi}, \ldots, V_q^{\pi}]$, where V_i^{π} is defined as

$$V_i^{\pi}(x) = \int_U \int_X \left(R_i(x,u,y) + \gamma_i V_i^{\pi}(y) \right) P(dy|x,u) \pi(du|x).$$

Given the initial–state distribution μ and the vector of value functions $\mathbf{V}^{\pi}$ for policy π, it is possible (as done in Eq. 1) to compute the vector of expected returns $\mathbf{J}^{\pi} = [J_1^{\pi}, \ldots, J_q^{\pi}]$.

Definition 1: Policy π *dominates* policy π' if there exists $1 \leq i \leq q$ such that $J_i^{\pi} > J_i^{\pi'}$ and there does not exist $1 \leq j \leq q$ such that $J_j^{\pi} < J_j^{\pi'}$.

Definition 2: Policy π is *Pareto–optimal* if there is no policy π' that dominates π.

In general, it is not possible to find a single policy which dominates all the others; when conflicting objectives are considered, no policy can simultaneously maximize all the objectives. Solving a MOMDP means to find the set of Pareto–optimal policies $\Pi^* = \{\pi | \nexists \pi' \text{ that dominates } \pi\}$, which maps to the so–called Pareto frontier $\mathcal{J}^* = \{\mathbf{J}^{\pi^*} | \pi^* \in \Pi^*\}$.

The traditional approach to solve a multi–objective problem is to transform it into a series of single–objective problems by combining the different objectives with some *scalarizing function* $\psi : \mathbb{R}^q \to \mathbb{R}$ [10]. The most straightforward choice for ψ is a convex combination of the objectives (weighted–sum method) using weights $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_q] \in \Lambda^{q-1}$, where Λ^{q-1} is the unit $(q-1)$–dimensional simplex (so that $\sum_{i=1}^{q} \lambda_i = 1$ and $\lambda_i \geq 0$ for all i). To simplify notation, in the following we drop the superscript $q-1$ from the simplex symbol when there is no risk of confusion. Each vector of weights $\boldsymbol{\lambda}$ defines a single–objective MDP with the following reward function:

$$R_{\boldsymbol{\lambda}}(x,u,y) = \sum_{i=1}^{q} \lambda_i R_i(x,u,y).$$

By linearity of the mathematical expectation and the weighted sum, the expected return of policy π with weight vector $\boldsymbol{\lambda}$ is: $J_{\boldsymbol{\lambda}}^{\pi} = \boldsymbol{\lambda}' \cdot \mathbf{J}^{\pi}$. Since all optimal policies of such single–objective MDPs ($\Pi_{\Lambda}^* = \{\pi_{\boldsymbol{\lambda}}^*\}_{\boldsymbol{\lambda} \in \Lambda}$) are provably Pareto–optimal solutions of the original MOMDP [11], the Pareto–frontier can be estimated by computing the set of expected–return vectors obtained for all the possible values of $\boldsymbol{\lambda}$: $\mathcal{J}_{\Lambda}^* = \left\{ J^{\pi_{\boldsymbol{\lambda}}^*} \right\}_{\boldsymbol{\lambda} \in \Lambda}$.

However, not all Pareto–optimal points can be obtained in this way; in fact, deterministic policies do not suffice for Pareto optimality ($\Pi_{\Lambda}^* \subset \Pi^*$). The reason is that different non-dominated trade–offs among criteria can be obtained by considering stochastic policies [6]. Nonetheless, since the expected–return vector of any stochastic policy is in the convex hull of the expected-return vectors of deterministic policies, the Pareto frontier can be fully estimated from the set of non-dominated deterministic policies computed using the weighted–sum method[1].

An approximation of the set of Pareto–optimal policies, and the associated Pareto frontier, is obtained by considering a finite number of sample weight combinations $\hat{\Lambda} \subset \Lambda$. The more weights combinations are evaluated the more accurate the approximation, but also the longer the computing time needed. The advantage is that such single–objective MDPs can be solved by many standard single–objective methods like dynamic–programming and linear–programming, or faced with reinforcement–learning techniques.

[1] In episodic tasks, to determine policies for each point on the frontier, instead of stochastic policies, we can consider mixture policies, i.e., stochastic combinations of deterministic policies [6].

III. Multi–Objective Batch Reinforcement Learning

When the state space and the action space are finite, MDPs can be solved using dynamic–programming algorithms (such as value iteration or policy iteration) or linear programming in polynomial time. Nonetheless, they suffer from the Bellman's curse of dimensionality that prevents their application in many real–world problems involving medium–to–high order continuous dynamical systems. To overcome these limits, a plethora of approximate dynamic–programming methods [12] have been devised in the recent past. Furthermore, in several applications, the system dynamics are unknown, thus making such methods useless. Reinforcement Learning (RL) algorithms allow the estimation of $Q^*(x,u)$ directly interacting with the environment through a trial–and–error process. Recently, considerable interest has developed in the study of batch-mode RL algorithms that, on the basis of experience samples previously collected from the system, allow to learn offline solutions to control problems.

A. Fitted Q–iteration

While in online learning the control policy is modified step by step according to the previous experience, the batch approach aims at determining the best control policy given a set of experience samples $D = \{\langle x_i, u_i, y_i, r_i\rangle\}_{1\leq i\leq N}$ previously collected according to a given sampling strategy[2]. Since learning is performed off-line, there is no need for directly experimenting on the real system, which is a key requirement in many real–world systems (e.g. water resources systems) for which experiments would lead to unsustainable costs in terms of time, social, and economic loss [14].

In particular, good results have been achieved by the *fitted Q–iteration* algorithm [9] a model–free approach derived from the *fitted value iteration* approach. The idea of FQI is to reformulate the RL problem as a sequence of supervised learning problems.

Given the dataset D, in the first iteration of the algorithm, for each tuple $\langle x_i, u_i, y_i, r_i\rangle$, the corresponding training pair is set to $(x_i, u_i) \rightarrow r_i$, and the goal is to use a regression algorithm to estimate a function that approximates the expected immediate reward $Q_1(x,u) = E[R(x_t,u_t)|x_t = x, u_t = u]$. The second iteration, based on the approximation $\hat{Q}_1$ of the Q_1–function, extends the optimization horizon one step further, by estimating function $\hat{Q}_2$ through regression on the following training dataset: $T_2 = \left\{\left[(x_i,u_i) \rightarrow r_i + \gamma \max_{u\in U} \hat{Q}_1(y_i,u)\right]\right\}_{1\leq i\leq N}$. By proceeding in the same way, at the k^{th} iteration, using the approximation of the Q_{k-1}–function, we can compute an approximation of the optimal action–value function at horizon k. The procedure iterates until the Q–function converges or a maximum number of iterations is reached (see [9] for a discussion about the stopping condition and the convergence properties of the algorithm).

[2]It is assumed that samples in D are enough to avoid conditioning problems with regression algorithms [13].

B. Multi–Objective Fitted Q–iteration

In this paper we propose to extend the FQI algorithm to multi–objective problems, thus producing the multi–objective FQI (MOFQI). The idea is to enlarge the state space X with the unit $(q-1)$–dimensional simplex Λ^{q-1}, in order to consider different weight combinations of the q objectives. Starting from the original dataset D, a new dataset for MOFQI is created by generating new tuples. For each tuple in D, k new tuples are produced by adding a random weight vector (uniformly drawn from the unit $(q-1)$–dimensional simplex) on which the new reward value is computed:

$$D_{MO} = \{\langle x_i, \boldsymbol{\lambda}_i, u_i, y_i, \boldsymbol{\lambda}_i, \boldsymbol{\lambda}_i' \cdot \mathbf{R}(x_i,u_i,y_i)\rangle\}_{1\leq i\leq N_{MO}}, \quad (2)$$

where $N_{MO} = N \cdot k$ is the number of tuples in the MOFQI dataset[3]. It is worth noting that the weight values are actually constant parameters. The result is that MOFQI approximates an optimal action–value function which is parameterized by $\boldsymbol{\lambda}$: $\hat{Q}^*(x,\boldsymbol{\lambda},u)$. In this way, it is possible to generalize information even over the weight space and, after a single training process, MOFQI learns a continuous approximation of the optimal policy over the weight space:

$$\hat{\pi}^*_{\boldsymbol{\lambda}}(x) = \arg\max_{u\in U} \hat{Q}(x,\boldsymbol{\lambda},u),$$

from which the approximate Pareto frontier $\hat{\mathcal{J}}_\Lambda = \left\{\mathbf{J}^{\hat{\pi}^*_{\boldsymbol{\lambda}}}\right\}_{\boldsymbol{\lambda}\in\Lambda}$.

The state space wherein MOFQI operates has a higher dimension than the one of the corresponding single–objective algorithm. As a consequence, to obtain similar performances, MOFQI requires in general more tuples than FQI. Nonetheless, starting from the dataset D used by FQI, it is possible to generate, by increasing the tuple multiplication factor k, a dataset D_{MO} for MOFQI with an arbitrarily larger number of samples ($N_{MO} \gg N$) without collecting new experience samples from the system. As we will see in the experimental section (see Section IV-E), the use of MOFQI is computationally more efficient than solving multiple single–objective problems as soon as the number of these problems exceeds a few units. Furthermore, the generalization over the weight space may be a key factor when problems with many objectives are considered.

IV. Test case study

To evaluate the proposed MOFQI approach, we consider a typical multi–objective control problem: the optimal operation of a water reservoir [14]. For the sake of clarity and to better describe the MOFQI properties, we initially consider a numerical case study, where a simplified stationary model of the problem is defined. A description of parameters used in the

[3]In the general MOMDP formulation, objectives can be associated to different discount factors. Although in this paper we focus only on MOMDPs with a unique discount factor, MOFQI can be easily adapted to work in the more general setting by generating the training datasets as follows: $T_k = [(x_i,u_i) => \sum_{j=1}^{q} \lambda_j \gamma_j^{L-k} r_{j,i} + \max_u Q_{k-1}(y_i,u)]$, where L is the temporal horizon.

experiments for the different algorithms and the introduction of the criterion used to evaluate their performance complete this section.

A. MOMDP model of a water reservoir

A water reservoir can be modeled as a MOMDP with a continuous state variable representing the water volume stored in the reservoir, a continuous action that controls the water release, a state–transition model that depends also on the stochastic reservoir inflow, and a set of conflicting objectives (e.g., avoiding floods, irrigation supply, energy supply, navigation). Formally, the state–transition function can be described by the mass balance equation:

$$x_{t+1} = x_t + \varepsilon_{t+1} - \max\left(\underline{u}_t, \min(\overline{u}_t, u_t)\right) \tag{3a}$$

where x_t is the reservoir storage at time t; ε_{t+1} is the reservoir inflow from time t to $t+1$, generated by a white noise process with normal distribution $\varepsilon_{t+1} \sim N(40, 100)$; u_t is the release decision; $\underline{u}_t$ and $\overline{u}_t$ are the minimum and maximum releases associated to the storage x_t according to the following relations

$$\overline{u}_t = x_t \quad \text{and} \quad \underline{u}_t = \max(x_t - 100, 0) \tag{3b}$$

In this case study, two objectives are considered: flooding along the lake shores and irrigation supply. The associated immediate reward to the flood objective is the negative of the cost due to the excess level w.r.t. a flooding threshold $\bar{h}$: $R_1(x_t, u_t, x_{t+1}) = -\max(h_{t+1} - \bar{h}, 0)$, where h_{t+1} is the reservoir level, given by the storage x_{t+1} divided by the reservoir surface S (in the following experiments $S = 1$), and the flooding threshold is set to 50. The immediate reward function for the other objective is the negative of the deficit in the water supply w.r.t. the water demand $\bar{\rho}$: $R_2(x_t, u_t, x_{t+1}) = -\max(\bar{\rho} - \rho_t, 0)$, where $\rho_t = \max\left(\underline{u}_t, \min(\overline{u}_t, u_t)\right)$ is the release from the reservoir and the water demand is 50. Given the non–economic nature of the above performance indicators and since the MOMDP can be solved optimizing over a finite–time horizon, we set the discount factor γ to 1 for all the objectives.

B. Algorithm Parameters

To evaluate the effectiveness of the MOFQI algorithm we have analyzed its performance in comparison with the solutions found by FQI and Stochastic Dynamic Programming (SDP) using the weighted-sum method. In the following we describe how we have computed the reference solution using SDP and the parameterizations used for FQI and MOFQI in the experiments of Section IV-E.

1) Stochastic Dynamic Programming: As a reference solution we consider the Pareto solutions computed by Stochastic Dynamic Programming (SDP) at 11 different values of the weight vector, namely $\boldsymbol{\lambda} = [\lambda_1, 1 - \lambda_1]$ with $\lambda_1 \in \hat{\Lambda} = \{0.0, 0.1, 0.2, \ldots, 1.0\}$. For each $\boldsymbol{\lambda}$, SDP consists in determining the optimal value function $V^*(\cdot, \boldsymbol{\lambda})$, which returns the maximum expected reward associated to each state x_t:

$$V^*(x_t, \boldsymbol{\lambda}) = \max_{u_t} \underset{\varepsilon_{t+1}}{E} \left[R_{\boldsymbol{\lambda}}(x_t, u_t, x_{t+1}) + \gamma V^*(x_{t+1}, \boldsymbol{\lambda})\right]. \tag{4}$$

Expectation in (4) requires knowledge of the disturbance probability distribution function (pdf). Since we want a reference solution, a perfect knowledge of such pdf is assumed. The equation is solved numerically, using 156 uniformly spaced storage values (from 0 to 155); 161 values of the control u_t (from 0 to 160); and 81 values of inflow ε_{t+1} (from 0 to 80). The solution algorithm is iterative. At each iteration (say the k-th), the current estimate $\hat{V}^k(\cdot)$ of the Bellman function is used to approximate $V^*(\cdot)$ on the right hand side of (4) and derive the new estimate $\hat{V}^{k+1}(\cdot)$. At the first iteration, the value function is initialized at $\hat{V}^0(x_t) = 0$ for all discretized values of x_t. If $\gamma < 1$, the approximate Bellman function converges and the algorithm terminates when the maximum absolute difference between $\hat{V}^k(\cdot)$ and $\hat{V}^{k+1}(\cdot)$ goes below a given tolerance threshold. Alternatively, the algorithm can be interrupted after a prescribed number of iterations (this is the termination test used when $\gamma = 1$). After the final iteration (say the L^{th}), the approximation of the optimal value function is assumed equal to its last approximation, $\hat{V}^*(x_t) = \hat{V}^L(x_t)$, $\forall x_t$. Once the value function has been computed, the optimal policy is:

$$\hat{\pi}^*_{\boldsymbol{\lambda}}(x_t) = \arg\max_{u_t} \underset{\varepsilon_{t+1}}{E} \left[R_{\boldsymbol{\lambda}}(x_t, u_t, x_{t+1}) + \gamma \hat{V}^*(x_{t+1}, \boldsymbol{\lambda})\right].$$

Numerical results presented in Section IV-E have been obtained by $L = 10$ iterations.

2) FQI and MOFQI: The simple reservoir model considered in this case study, allow us to feed both FQI and MOFQI algorithms with experience samples drawn uniformly random from the state–action space; for each sample the next state and reward values are obtained from the generative model (3). The only difference between the dataset for FQI and the one for MOFQI is that in the multi–objective case the state space has one more dimension which represents the value of weight λ_1. As a result, the dataset takes the shape presented in Eq. 2.

We have chosen to approximate the Q–functions using extremely–randomized trees (Extra–Trees) [15] since they have been successfully combined with the FQI algorithm in many applications [9], [14]. Given a dataset $D = \{\langle i^l, o^l \rangle\}_{1 \le l \le N}$, the extra–tree building algorithm grows an ensemble of M regression trees. Nodes are split using the following rule: K alternative cut-directions (regressor input) are randomly selected and, for each one, a random cut-point is chosen; a score (explained variance) is then associated to each cut-direction and the one maximizing the score is adopted to split the node (for details, see [15]). The algorithm stops partitioning a node if its cardinality is smaller than n_{min} and the node is therefore a leaf. To each leaf a value is assigned, obtained as the average of the regressor outputs o^l associated to the inputs i^l that fall in the leaf. The estimates produced by the M trees are finally aggregated with arithmetic average.

In the following experiments we have used $M = 100$ trees and a number of alternative cut directions equal to the number

of state and actions variables (two for FQI and three for MOFQI). The score function and the termination test of the split process have been slightly modified w.r.t. to the ones proposed in [15]. In [15], the node is split in the direction with the largest variance reduction and the split process is recursively repeated until the number n_{min} of samples in the node falls below a given threshold, which determines how much each regression tree generalizes over the input space. Since choosing such threshold may be difficult and using the same generalization over the whole input space may be not effective, we propose a different score function based on a Student's t-test to estimate which cut is most likely to partition the input space into two subsets with different output means. Exploiting this information, the splitting process is stopped when no cut has a probability higher than $\tau\%$ of producing two regions with different output means (after tuning, in the experiments we used $\tau = 0.98$).

C. Performance Evaluation

Policies produced by SDP and learning algorithms are evaluated using Monte Carlo simulations. Each simulation consists of 100 steps and is repeated 10 times (using independent realizations of inflow trajectories) for each one of the 10 different initial states (chosen uniformly random over the state space X). As a consequence, the performance of each policy π is evaluated on 100 scenarios of 100 steps. The result is a performance vector $\overline{\mathbf{J}}^{\pi}$ whose components are the average rewards per step associated to each objective under π.

D. Comparing two Pareto frontiers

In the recent past, many performance metrics for multi–objective optimization have been proposed [16]. From [16], we adopt a measure that, for any weight $\boldsymbol{\lambda}$, takes into account the difference between the aggregate expected return in the optimal Pareto frontier and the one produced by an approximated algorithm. The measure is normalized by the differences of optimal function values in the Nadir and Utopia points. Given the weighted linear combination of the reward functions, we measure the loss L of an approximation of the Pareto frontier, denoted with $\hat{\mathcal{J}}_{\Lambda} = \left\{ J^{\hat{\pi}^*_{\boldsymbol{\lambda}}} \right\}_{\boldsymbol{\lambda} \in \Lambda}$, w.r.t. the subset of the optimal Pareto frontier obtained considering the optimal solutions for any weight vector $\mathcal{J}^*_{\Lambda} = \left\{ J^{\pi^*_{\boldsymbol{\lambda}}} \right\}_{\boldsymbol{\lambda} \in \Lambda}$ as:

$$L_p(\hat{\mathcal{J}}_{\Lambda}, \mathcal{J}^*_{\Lambda}) = \int_{\boldsymbol{\lambda} \in \Lambda} \frac{(J^{\pi^*_{\boldsymbol{\lambda}}} - J^{\hat{\pi}^*_{\boldsymbol{\lambda}}})}{\Delta J^*_{\boldsymbol{\lambda}}} p(d\boldsymbol{\lambda}), \qquad (5)$$

where $p(\cdot)$ is a probability density over the simplex Λ and $\Delta J^*_{\boldsymbol{\lambda}} = \sum_{i=1}^{q} \lambda_i \left(\max_{\boldsymbol{\lambda}' \in \Lambda} J_i^{\pi^*_{\boldsymbol{\lambda}'}} - \min_{\boldsymbol{\lambda}' \in \Lambda} J_i^{\pi^*_{\boldsymbol{\lambda}'}} \right)$ is the normalization factor. According to this (non–negative) measure, an approximate Pareto frontier $\hat{\mathcal{J}}^A_{\Lambda}$ produced by algorithm A is better than the approximation $\hat{\mathcal{J}}^B_{\Lambda}$ produced by algorithm B when $L_p(\hat{\mathcal{J}}^A_{\Lambda}, \mathcal{J}^*_{\Lambda}) < L_p(\hat{\mathcal{J}}^B_{\Lambda}, \mathcal{J}^*_{\Lambda})$ (i.e., if its loss is smaller). Since, in general, the optimal Pareto frontier $\mathcal{J}^*$ is not available, a reference Pareto frontier can be used instead.

E. Experimental Results

In this section, we will show how the non-dominated policies for the previously–defined water–reservoir control problem are approximated by the MOFQI algorithm. As a reference, we consider the solutions computed by SDP for the set of weight values. Although the solution computed by SDP is still an approximation, the discretization described in Section IV-B1 achieves near–optimal performance. Furthermore, we discuss computational aspects of the proposed approach when compared to frontier approximation by means of multiple executions (using different weights) of the single–objective FQI algorithm.

Figure 1 shows the continuous multi–objective value function computed with MOFQI on a dataset composed of $20,000$ tuples ($10,000$ experience samples and $k = 2$) and compares it with the discrete approximation $\hat{V}^*(\cdot, \boldsymbol{\lambda})$ produced by SDP for the 11 sampled weights values $\hat{\Lambda}$ [4]. As we can notice, the approximation produced by MOFQI is fairly good for mid–range weight values, while it is quite inaccurate at the boundaries. This behavior is not surprising and it is usual when, as in the case of Extra-trees, an averager [17] is used as function approximator. Since averagers use convex combinations of output values associated to neighbor samples, they generally fail to extrapolate properly and approximation errors at the edges of the sampled space are often larger. Nonetheless, as shown in Fig. 2, the infinite set of policies (parameterized in $\boldsymbol{\lambda}$) learned by MOFQI is quite close to the finite set of policies computed by SDP.

For $\lambda_1 = 0$, i.e., only the irrigation objective is considered, the optimal decision is to release the water demand (50) as long as this is possible (storage $x_t \geq 50$), and the maximum possible (x_t) otherwise (Fig. 2). Correspondingly, the optimal value function (Fig. 1) is lowest when the reservoir is empty (critical situation) and increases with the storage. For $\lambda_1 = 1$, i.e., only the flood objective is considered, the optimal decision is to always release the maximum possible, i.e., x_t (Fig. 2). The optimal value function (Fig. 1) is independent of the storage, in fact, if one releases the maximum possible, then the future state x_{t+1} does not depend on the current state x_t but equals the inflow ε_{t+1} (see (3)). Intermediate weight values lead to compromise solutions.

Figure 3 shows the Pareto frontiers obtained by Monte Carlo simulations (as explained in Section IV-C) of the non-dominated solutions produced by the different algorithms. As we can see, the approximation of the Pareto frontier produced by FQI using $10,000$ samples is similar to the one of MOFQI when using the same $10,000$ samples to build a dataset with $20,000$ tuples. Nonetheless, from the computational perspective, it is worth recalling that FQI has to solve as many single–objective learning problems (using $10,000$ tuples for each of them) as the number of points used to approximate the Pareto frontier.

Since MOFQI has been proposed as an alternative approach

[4] In the plot, the marks are less than 11 because, as it often happens in MO problems, some points overlap

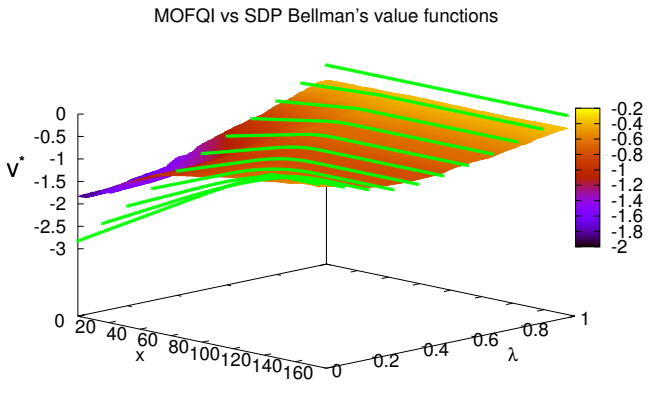

Fig. 1: Comparison between the value function by SDP for 11 weight values and the one estimated by MOFQI over the whole weight space.

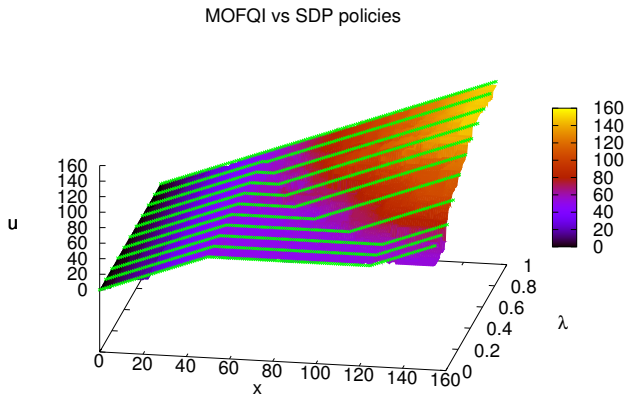

Fig. 2: Comparison between the control law by SDP for 11 weight values and the one estimated by MOFQI over the whole weight space.

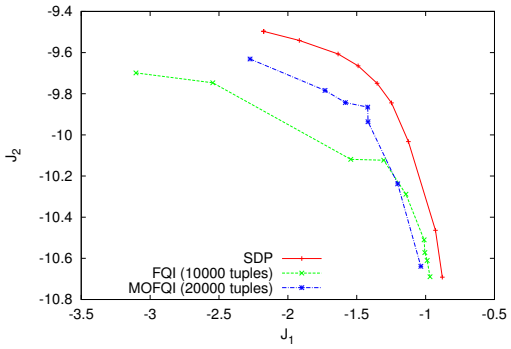

Fig. 3: Comparison among the approximated Pareto frontiers obtained with SDP, FQI, and MOFQI for 11 weight values.

samples	FQI	MOFQI $k=1$	MOFQI $k=2$	MOFQI $k=3$	MOFQI $k=4$	MOFQI $k=5$
1000	0.321	0.654	0.532	0.506	0.389	0.313
2000	0.265	0.527	0.504	0.433	0.329	0.287
3000	0.241	0.480	0.421	0.383	0.282	0.242
4000	0.240	0.458	0.380	0.324	0.241	0.224
5000	0.224	0.444	0.351	0.250	0.239	0.223
10000	0.208	0.348	0.237	0.209	0.199	0.190

TABLE I: Loss of FQI and MOFQI (with different factors k) w.r.t. SDP when different sample size are considered

to the repeated application of single-objective FQI following the weighting method, we will discuss in what conditions MOFQI outperforms repeated FQI. Table I shows the loss L (as defined in Eq. 5) for FQI and MOFQI (using different values for the multiplication factor k) algorithms w.r.t. to SDP, as the number of experience samples varies between $1,000$ and $10,000$. The loss has been computed by considering $p(\cdot)$ as a uniform distribution over the discrete set $\hat{\Lambda}$ of 11 weights defined in the previous section. As expected, the loss reduces for both the approaches as the number of experience samples increases and the number of tuples required for the multi–objective problem is larger than the one needed by FQI for learning single–objective problems, the accuracy being the same. From the data in Table I, we can observe that, starting from the same set of experience samples, MOFQI performs as FQI when its training dataset is about five times larger than the one of FQI. From the computational perspective, while the training time of FQI grows linearly with the number of required points on the Pareto frontier, MOFQI is independent of such number. So, we can conclude that, to have a dense approximation of the Pareto frontier, MOFQI is computationally preferable to repeated FQI (in the proposed test case, this is true as long as more than five points are required).

V. Hoa Binh Reservoir Case Study

The second case study considered in this paper is a real-world system, the Hoabinh reservoir in Northern Vietnam (Figure 4). The reservoir has a surface area of about 198 km^2 and an active storage of 6.056 billion m^3. The main objectives of reservoir operation are hydropower production (the plant has a capacity of 1920 MW and produces more than 7,000 GWh per year) and flood mitigation in the downstream city of Hanoi. The reservoir and the downstream river network are modeled by a combination of conceptual and data-driven models with a daily resolution time step. A detailed description of the system and associated model can be found in [18], here we will provide a brief description of the problem formulation as a MODMP.

The problem can be modeled with two state variables, the reservoir storage and the corresponding day of the year[5], the action variable u is the release decision for the next 24 hours, and the future state is the reservoir storage y on the day after, estimated by a mass balance equation like (3a), and $t+1$. The minimum and maximum feasible release $\underline{u}$ and $\overline{u}$ are computed

[5]Since the system can be described as cyclostationary with period $T = 365$ days, we can obtain a stationary MDP by enlarging the state space with the time variable [19].

based on the storage and inflow values, and taking into account the rating curves of the bottom and intermediate gates and the spillways. The reward associated to the hydropower objective (R_1) is the value (ranging from 0 to 1) of the daily hydropower production, the one of flood control (R_2) is the cost of floods changed in sign, i.e. it ranges from -1 (maximum cost) to 0 (minimum cost), and it is estimated by a non–linear function of the water level in Hanoi.

A. Experimental setup for FQI and MOFQI

Time series of measured flows over the period 1957-1978 together with the simulation model were used to generate the dataset for FQI and MOFQI. For each day in the time series, ten storage and action values were randomly sampled: the storage is drawn uniformly over the range from 3.7151×10^9 to 1.0415×10^{10} m^3, while the action is randomly chosen from a finite set of 20 values of daily average release, ranging from 0 to 13,000 m^3/s. The total number of experience samples is $73,650$. First, FQI is repeatedly applied under 6 different values of the weight λ_1. Then, MOFQI is applied once, using the enriched dataset of Eq. (2). Such dataset is obtained by associating each original experience sample with k random sampled weight values[6]; in the following, results relevant to the case $k = 3$ and $k = 7$ will be compared.
In all the optimization experiments, Extra Trees were used to approximate the state–action value function, with $M = 100$ trees and a number of alternative cut directions K equal to 3 for FQI and 4 for MOFQI. The threshold τ for computing the score function described at the end of Sec. IV-B2 was set to 0.9. The number of algorithm's iterations L was decided based on the analysis of the system functioning. From the definition of the hydropower objective, it follows that the optimal reservoir operation should allocate the hydropower production in the period of maximum energy value, i.e. from April to June. The storage to sustain such production must have been created in the previous flood season (August–September of the previous year) that is around 200 days before. The optimal operation horizon for the flood objective, instead, is much shorter since the time required to empty the reservoir in anticipation of a big flood is around 10 days. Since the number of algorithm iterations should correspond to the maximum length of the operation horizon for the considered objectives, L was set to 200.

B. Experimental Results

The operating policies were simulated under the time series of observed reservoir inflows and Lo and Thao discharges over the time horizon 1957-1978 (calibration dataset) and the horizon 1995-2004 (validation dataset). Performances over the validation dataset can be compared to the historical operation (average reward values: J_1=0.292, J_2=-0.129), as the reservoir construction was completed in 1989 and its filling in 1994.

[6]In this experiment $k-2$ weight values were randomly chosen, while the remaining two were set to $\lambda_1 = 1$ (hydropower only) and $\lambda_1 = 0$ (flood control only).

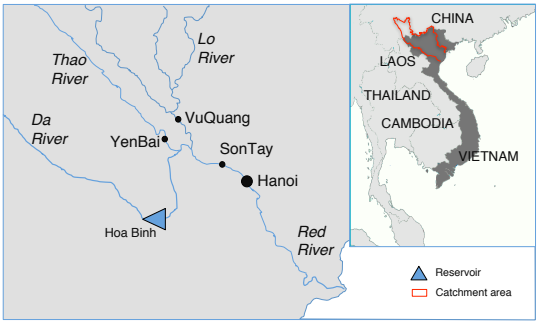

Fig. 4: The Hoa Binh reservoir water system in Northern Vietnam.

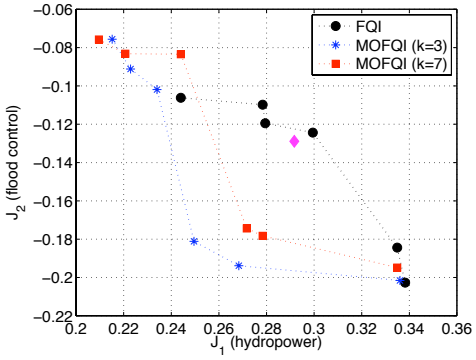

Fig. 5: Hoa Binh case study: comparison among the approximated Pareto frontiers obtained with FQI and MOFQI with different multiplication factors (validation dataset). Magenta diamond is the historical operation performance.

Table II reports the objective values over the calibration dataset, Table III those of the validation dataset. Figure 5 also shows the policies performances over the validation dataset. First, it can be noticed that the performances of MOFQI are obviously higher with larger multiplication factor k. More interestingly, the comparison of FQI and MOFQI (with $k = 7$) for λ_1 between 0 and 0.5 (i.e., flood control is more relevant than hydropower production) shows that while FQI outperforms MOFQI over the calibration dataset, it is outperformed over the validation dataset. The result is consistent with the findings of [20] for multi-task learning algorithm: learning the solutions of multiple tasks simultaneously, as in MOFQI, improves generalization abilities, while learning on a single task, as in reiterate FQI, is more likely to overfit the data.

For $\lambda_1 > 0.5$, it can be noticed that MOFQI performs rather poorly with respect to FQI. The reason is that, as stated above, the maximization of the hydropower objective requires a longer time horizon (around 200 days) than the flood control objective (about 10 days). This implies that the number L of algorithm iterations must be high, thus increasing the approximation error in the action–value function estimate. To reduce such approximation errors, the number of tuples must be increased, i.e. a much larger multiplication factor k must be used for MOFQI.

TABLE II: Hoa Binh case study: objective values (hyd=hydropower; flo=flooding) over the calibration dataset 1957-1978. Best scores in boldface.

	FQI			MOFQI (k=3)			MOFQI (k=7)		
λ_1	hyd	flo	J	hyd	flo	J	hyd	flo	J
0.0	0.244	-0.043	-0.043	0.222	-0.043	-0.043	**0.206**	**-0.041**	**-0.041**
0.1	**0.269**	**-0.046**	**-0.015**	0.225	-0.062	-0.033	0.207	-0.050	-0.024
0.5	**0.284**	**-0.056**	**0.114**	0.232	-0.068	0.082	0.247	-0.057	0.095
0.7	**0.300**	**-0.057**	**0.194**	0.236	-0.075	0.143	0.260	-0.064	0.163
0.9	**0.326**	**-0.076**	**0.286**	0.262	-0.088	0.228	0.283	-0.077	0.247
1.0	**0.328**	**-0.100**	**0.328**	0.326	-0.100	0.326	**0.328**	**-0.093**	**0.328**

TABLE III: Hoa Binh case study: objective values (hyd=hydropower; flo=flooding) over the validation dataset 1995-2004. Best scores in boldface.

	FQI			MOFQI (k=3)			MOFQI (k=7)		
λ_1	hyd	flo	J	hyd	flo	J	hyd	flo	J
0.0	0.244	-0.106	-0.106	**0.215**	**-0.076**	**-0.076**	**0.210**	**-0.076**	**-0.076**
0.1	0.278	-0.110	-0.071	0.223	-0.091	-0.060	**0.221**	**-0.083**	**-0.053**
0.5	0.279	-0.119	0.080	0.234	-0.102	0.066	**0.244**	**-0.083**	**0.081**
0.7	**0.299**	**-0.124**	**0.173**	0.250	-0.181	0.121	0.272	-0.174	0.138
0.9	**0.335**	**-0.184**	**0.284**	0.268	-0.194	0.223	0.278	-0.178	0.233
1.0	**0.338**	**-0.203**	**0.339**	0.336	-0.201	0.337	0.335	-0.195	0.336

VI. Conclusion

In this paper we presented an extension of batch-mode Reinforcement Learning to derive multi–objective optimal control policies. Experience gained from experiments conducted on a synthetic multi–purpose water reservoir shows that, using a relative small number of experience samples, MOFQI provides a good approximation of the Pareto–optimal frontier as computed with several repetitions of Stochastic Dynamic Programming for different weight values. In addition, MOFQI becomes computationally more efficient than the repeated application of its single–objective twin, FQI, when more than five points are used to approximate the Pareto frontier. The Hoa Binh reservoir case study was used to evaluate the benefits from the applicability of MOFQI on a real world sized problem. The approximation of the Pareto frontier produced by FQI using 73,650 tuples dominates the one by MOFQI when using the same 73,650 samples to build a dataset with $3\times$73,650 tuples. By enlarging the data set to $7\times$73,650 tuples, MOFQI provides comparable results, in terms of aggregated objective value, with those by FQI. However, with no additional computation cost the approximation of the Pareto front by MOFQI can be made much more dense, thus allowing for a more accurate exploration of trade-offs and for a better informed decision-making. A generalized analysis of the algorithm's computational requirements will be the subject of future studies.

References

[1] Z. Abidin, M. Arshad, and U. Ngah, "A Survey: Animal-Inspired Metaheuristic Algorithms," in *2nd Postgraduate Colloquium School of Electrical & Electronic USM, EEPC*, 2009.

[2] Z. Gábor, Z. Kalmár, and C. Szepesvári, "Multi-criteria reinforcement learning," in *Proc. of ICML*, 1998, pp. 197–205.

[3] S. Mannor and N. Shimkin, "A geometric approach to multi-criterion reinforcement learning," *JMLR*, vol. 5, pp. 325–360, 2004.

[4] S. Natarajan and P. Tadepalli, "Dynamic preferences in multi-criteria reinforcement learning," in *Proc. of ICML*, 2005, pp. 601–608.

[5] L. Barrett and S. Narayanan, "Learning all optimal policies with multiple criteria," in *Proc. of ICML*, 2008, pp. 41–47.

[6] P. Vamplew, R. Dazeley, E. Barker, and A. Kelarev, "Constructing Stochastic Mixture Policies for Episodic Multiobjective Reinforcement Learning Tasks," *AI 2009: Advances in Artificial Intelligence*, pp. 340–349, 2009.

[7] D. J. Lizotte, M. Bowling, and S. A. Murphy, "Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis," in *Proc. of ICML*, 2010, pp. 695–702.

[8] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker, "Empirical evaluation methods for multiobjective reinforcement learning algorithms," *Machine Learning*, pp. 1–30, 2010.

[9] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, no. 1, pp. 503–556, Apr. 2005.

[10] P. Perny and P. Weng, "On finding compromise solutions in multiobjective markov decision processes," in *ECAI*, 2010, pp. 969–970.

[11] K. Chatterjee, R. Majumdar, and T. Henzinger, "Markov decision processes with multiple objectives," *STACS 2006*, pp. 325–336, 2006.

[12] W. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. Wiley-Interscience, 2007.

[13] R. Munos and C. Szepesvári, "Finite-time bounds for fitted value iteration," *J. of Machine Learning Research*, vol. 9, pp. 815–857, 2008.

[14] A. Castelletti, S. Galelli, M. Restelli, and R. Soncini-Sessa, "Tree-based reinforcement learning for optimal water reservoir operation," *Water Resources Research*, vol. 46, no. 9, p. W09507, 2010.

[15] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[16] M. P. Hansen and A. Jaszkiewicz, "Evaluating the quality of approximations to the non–dominated set," Technical University of Denmark, Tech. Rep. IMM-REP-1998-7, 1998.

[17] G. Gordon, "Approximate solutions to markov decision processes," Ph.D. dissertation, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1999.

[18] A. Castelletti, F. Pianosi, X. Quach, and R. Soncini-Sessa, "Assessing water resources management and development in Northern Vietnam," *Hydrology and Earth System Sciences Discussions*, vol. 8, no. 4, pp. 7177–7206, 2011.

[19] A. Castelletti, S. Galelli, M. Restelli, and R. Soncini-Sessa, "Tree-based reinforcement learning for optimal water reservoir operation," *Water Resources Research*, vol. 46, no. W09507, 2010.

[20] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997.