A Fitted-Q Algorithm for Budgeted MDPs

Nicolas Carrara, Romain Laroche, Jean-Léon Bouraoui, Tanguy Urvoy, Olivier Pietquin

A Fitted-Q Algorithm for Budgeted MDPs

Nicolas Carrara
Orange Labs
Univ. Lille, CNRS,
Centrale Lille,
INRIA UMR 9189 - CRIStAL
F-59000 Lille, France
nicolas.carrara
@orange.com

Romain Laroche
Microsoft Maluuba
Montréal, Canada
romain.laroche
@microsoft.com

Jean-Léon Bouraoui
Orange Labs
Lannion, France
jeanleon.bouraoui
@orange.com

Tanguy Urvoy
Orange Labs
Lannion, France
tanguy.urvoy
@orange.com

Olivier Pietquin*
Univ. Lille, CNRS, Centrale Lille,
INRIA UMR 9189 - CRIStAL
F-59000 Lille, France
olivier.pietquin
@univ-lille1.fr

Abstract

We address the problem of budgeted/constrained reinforcement learning in continuous state-space using a batch of transitions. For this purpose, we introduce a novel algorithm called Budgeted Fitted-Q (BFTQ). We carry out some preliminary benchmarks on a continuous 2-D world. They show that BFTQ performs as well as a penalized Fitted-Q algorithm while also allowing ones to adapt the trained policy on-the-fly for a given amount of budget and without the need of engineering the reward penalties. We believe that the general principles used to design BFTQ could be used to extend others classical reinforcement learning algorithms to budget-oriented applications.

1 Introduction

Classic reinforcement learning (RL) algorithms focus on the maximization of a unique expected reward signal. But many applications have multiple, possibly conflicting, objectives. For example, an autonomous driving car must optimize its travel time, its fuel consumption, as well as the safety of people. In this example, the first two objectives are of the same nature. Although conflicting on short term, they become coherent on the long term. If the fuel tank is depleted, then the travel time will be potentially infinite. In practice, by replacing this infinite value by the time needed to walk to the next fuel station, one gets a reasonable way to combine these two objectives into a single one. The safety objective is of different nature: it is quite difficult to assess the cost of a dramatic car accident and it's often more intuitive to think in term of probability for a given run to fail the constraint.

*now with Google Brain, Paris

This problem is better formalized as a worst-case *constrained* or *risk-sensitive* RL problem: the car should optimize the expected travel time while guaranteeing a crash probability strictly lower than the one of a human driver. The natural formalism for constrained RL is given by *Constrained Markov Decision Processes* (CMDP Beutler and Ross, 1985; Altman, 1999).

The most direct approach to these kind of problems is to relax the constraint by adding a reward penalty. That is to say, if the cost is C we replace the reward R by $R - \lambda C$ where $\lambda > 0$ is the strength of the penalty. But the calibration of this penalty requires the ability to simulate the policies.

Most of the existing solutions focus on the modelling of one policy for each budget. But in some practical situations, the budget can decrease with time. Let us consider the example of a military swarm of drones in a reconnaissance mission. In that case, the individual drones policies should become more risk-averse as the swarm population decreases. For these kind of problems, the budget is a parameter of the policy; they are formalized as *Budgeted Markov Decision Process* (BMDP Boutilier and Lu, 2016). Solving a BMDP is a harder problem than solving a CMDP.

The aim of our work is to handle budgeted RL problems:

- Directly from batch of transitions (*i.e.* without simulation or interaction with the environment during training);
- In a continuous-state space;
- With the budget as a parameter of the policy.

For this purpose we introduce Budgeted Fitted-Q (BFTQ), an extension of the Fitted-Q (FTQ) algorithm. It takes as input a batch of RL transitions and estimates the optimal parametric strategies under budget. We experiment BFTQ

on 2-D worlds with forbidden areas. We show that BFTQ performs as well as a FTQ with reward penalties.

2 Background and Related Works

2.1 Markov Decision Process

A *Markov Decision Process* (MDP) is formalized as a tuple $\langle \mathcal{S}, \mathcal{A}, R, P, \gamma \rangle$; where $\mathcal{S}$ is the state set, $\mathcal{A}$ is the action set, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the (potentially stochastic) reward function, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function, and γ is the reward discount factor. The behaviour of the agent is defined as a policy $\pi : \mathcal{S} \to \mathcal{A}$, which can either be deterministic or stochastic. Solving an MDP consists in finding a policy π^* that maximises the γ-discounted expected return $\mathbb{E}_{\pi^*}[\sum_t \gamma^t R(s_t, a_t, s_{t+1})]$. The optimal policy π^* satisfies Bellman's optimality equation (Bellman (1956)):

$$\pi^*(s) = \underset{a \in A}{\operatorname{argmax}}\, Q^*(s, a),$$
$$Q^*(s, a) = \mathbb{E}_{s'|s,a}\left[R(s, a, s') + \gamma Q^*(s', \pi^*(s'))\right]$$

If $\gamma < 1$, this equation admits a unique solution. If the MDP structure is known, and if the state set $\mathcal{S}$ is of tractable size, the optimal policy can be computed directly as a fix-point of the Bellman equation. This algorithm is called *Value-Iteration* (VI). When the state set $\mathcal{S}$ is continuous or too large, the Q-function has to be approximated. Fitted Value-iteration fulfils this role.

2.2 RL algorithms for continuous MDP

Fitted Value-iteration (FVI) is a generic planning algorithm. It implements *Value-Iteration* with a regression model as a proxy for the Q-function. At each iteration it fits the model to map the state-action couples (s, a) to their respective expected values in the Bellman equation involving R and P. However, in RL problems, R and P are usually unknown.

Fitted-Q (FTQ, Ernst et al. (2005)) resolves the aforementioned problem. Let $(s_i, a_i, r'_i, s'_i)_{i \in [0,N]}$ be a batch of transitions which covers most of the state space. Given this batch, FTQ trains the Q-function at each iteration of VI using a supervised learning algorithm that regresses the following supervised training batch:

$$\{(s_i, a_i),\ r'_i + \gamma * \max_{a'} Q(s'_i, a')\}_{i \in [0,N]}$$

2.3 Budgeted Markov Decision Processes

A *Constrained Markov Decision Process* (CMDP, Beutler and Ross (1985); Altman (1999)) is an MDP augmented with a cost function $C : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, a cost discount γ_c, and a budget β. The cost function and its budget represent the "hard" constraint of the problem, while the reward function represents the task to complete. The problem of constrained RL consists in finding the policy which maximizes the expected return while keeping the discounted sum of the cost under the given budget β. Formally:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \sum_t \gamma^t R_t, \tag{1}$$
$$\text{subject to} \quad \mathbb{E}_\pi \sum_t \gamma_c^t C_t \leq \beta \tag{2}$$

where $R_t = R(s_t, a_t, s_{t+1})$ and $C_t = C(s_t, a_t, s_{t+1})$.

For the sake of simplicity we consider a weak form of risk in Eq. (2): the budget is only constrained *on average*. For sensitive applications the failure risk should be modelled in term of worst-deviation from the expected cumulative cost or in term of worst-percentile (Chow et al., 2018).

The *Budgeted Markov Decision Process* (BMDP, Boutilier and Lu (2016)) problem is a generalisation of the CMDP problem where the objective is to find a generic policy π^*_β which works for any CMDP of budget $\beta > 0$.

2.4 A remark on deterministic policies

A convenient property of MDPs, is that the optimal policy is unique, deterministic and greedy: $\pi^*(s) = \operatorname{argmax}_a Q(s, a)$. In a CMDP, and *a fortiori* in a BMDP, this is in general not the case. It has been shown indeed that the optimal policy under constraint is a random mixture of two deterministic policies Beutler and Ross (1985, Theorem 4.4).

To illustrate this fact, let us consider the trivial BMDP on the left of Fig. 1. On this example we have $\mathbb{E}_\pi R = 10\pi_1$ and $\mathbb{E}_\pi C = \pi_1$. The deterministic policy consisting in always picking the safe action is feasible for any $\beta \geq 0$. But if $\beta = 1/2$, the most rewarding feasible policy is to randomly pick the safe and risky actions with equal probabilities. If we attempt to cast this BMDP into an MDP by replacing the costs by negative rewards, the policy we will obtain will be deterministic, hence suboptimal.

2.5 Related work

Optimization under constraint algorithms have proven their worth in many real life applications like operational efficiency for businesses or network design problems during the 90s (Bertsekas (1996)). But, these algorithms do not capture the temporal nature of a wide class of RL problems, such as in robotics, dialogue systems, control systems, or trading strategies.

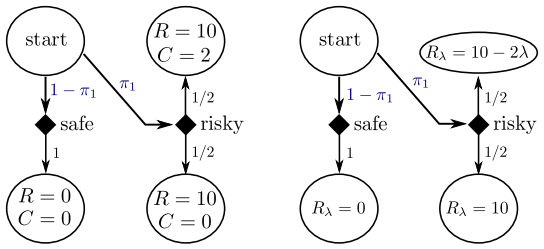


Figure 1: On the left hand side, a simple *risky-vs-safe* BMDP. The probability of picking the risky action is π_1. On the right hand side an attempt to relax the problem with negative rewards.

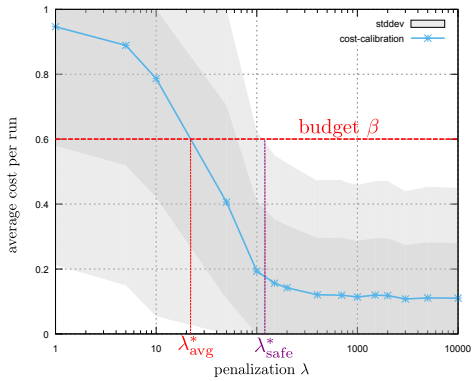


Figure 2: Calibration of a penalty multiplier according to the budget β. The optimal multiplier λ^*_{avg} is the smallest one to satisfy the budget constraint on average. Safer policies can also be selected according to the largest deviation form this mean cost.

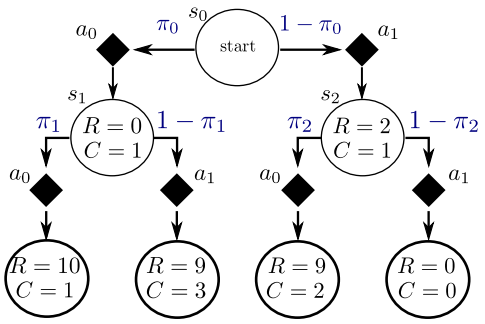


Figure 3: A simple deterministic finite BMDP.

As a consequence, researchers got interested in resolving constrained sequential problems (García and Fernández (2015)), such as MDP under constraints Beutler and Ross (1985); Altman (1999)). Undurti et al. (2010) proposes an algorithm to solve CMDP with a continuous state-space using the same principles as FVI; but they assume that the environment (R, C and P) is known.

Geibel and Wysotzki (2005); Chow et al. (2018); Achiam et al. (2017) propose algorithms for CMDP, but they require interactions with the environment during the training (actor-critic, Q-learning and Policy Gradient, Constrained-Policy-Optimization).

Abe et al. (2010) work with constraint-based RL but their solution is a Q-learning algorithm adapted to batch learning, which is known not to be a sample-efficient batch RL algorithm. Thomas et al. (2015); Petrik et al. (2016); Laroche and Trichelair (2017) introduced batch RL algorithms for risk sensitive problems but these algorithms were not adapted for continuous environments.

It is important to note, that these algorithms are designed to work for CMDP and require some adaptation to fit the general BMDP problem. On the contrary, Boutilier and Lu (2016) resolve a BMDP using a VI-like algorithm; however it only applies to finite and known environments.

3 Penalized Fitted-Q Iteration

As mentioned before, a natural way to relax the constraint of Eq. (2) is to penalize states with high cost. Assuming $\gamma = \gamma = \gamma_c$, the MDP problem to solve becomes:

$$\pi^*_\lambda = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \sum_t \gamma^t \left(R_t - \lambda C_t\right), \qquad (3)$$

As explained on Fig. 2, the optimal deterministic policy can be obtained by a line-search on the Lagrange multiplier values λ. Then, according to Beutler and Ross (1985, Theorem 4.4), the optimal policy is a randomized mixture of two deterministic policies: the safest deterministic policy that violates the constraint $\pi_{\lambda-}$ and the riskier of the feasible ones $\pi_{\lambda+}$.

Fitted-Q can be easily adapted continuous states CMDP and BMDP through this methodology, but, given the high variance, it requires a lot of simulations to get a proper estimate of the calibration curve. Our purpose is to avoid this calibration phase.

4 Budgeted Fitted-Q Iteration

In this section, we introduce **Budgeted-Fitted-Q** (BFTQ), a batch RL under constraints algorithm for BMDP, as an extension of the regular FTQ algorithm.

4.1 Intuition

We illustrate the principle of BFTQ on a simple BMDP described on Fig. 3. This example counts 7 states and 2 actions, its transition, reward, and constraint functions are

fully deterministic. In order to maximise the cumulative reward while keeping the sum of the cost under β, there are only 3 parameters to optimise: $\pi_0 = \mathbb{P}(a_0 \mid \text{start})$, $\pi_1 = \mathbb{P}(a_0 \mid s_2)$, and $\pi_2 = \mathbb{P}(a_0 \mid s_2)$.

First note that on the left hand side of the figure, on state s_1, the action a_0 dominates a_1 for any budget: we can safely set $\pi_1 = 1$ and replace state s_1 by a terminal state of expected reward 10 and cost 5. We write: $Q_r(s_0, a_0) = 10$ and $Q_c(s_0, a_0) = 2$.

Assuming $\gamma = \gamma_c = 1$, the optimization problem of Eq. 1 simplifies into:

$$\begin{aligned} &\max_{\pi_0, \pi_2} 10\pi_0 + (1 - \pi_0) \cdot (2 + 9\pi_2) \\ &\text{s.t. } 2\pi_0 + (1 - \pi_0) \cdot (1 + 2\pi_2) \leq \beta \end{aligned}$$

The best policy depends on the budget: If $\beta < 1$ the problem is infeasible, if $\beta \geq 3$ we maximize reward by playing a_1, then a_0. If $\beta = 3/2$, the best policy is to take $\pi_0 = 1/2$ and $\pi_2 = 0$, if $\beta = 5/2$, we rather take $\pi_0 = 1/2$ and $\pi_2 = 1$.

It is also possible to solve the problem locally on s_2 as long as the budget $\beta \geq 1$ is left as a free parameter:

$$\begin{aligned} \pi_2^*(\beta) &= \operatorname*{argmax}_{\pi_2} 9\pi_2 \text{ s.t. } 2\pi_2 \leq \beta - 1 \\ &= \min\{1, (\beta - 1)/2\} \end{aligned}$$

We then write:

$$\begin{aligned} Q_r(s_0, a_1, \beta) &= 2 + 9\pi_2^*(\beta) \\ Q_c(s_0, a_1, \beta) &= 1 + 2\pi_2^*(\beta) \end{aligned}$$

The final optimization problem becomes:

$$\begin{aligned} &\max_{\pi_0, \beta_1} \pi_0 Q_r^\pi(s_0, a_0) + (1 - \pi_0) Q_r^\pi(s_0, a_1, \beta_1) \qquad (4) \\ &\quad \text{s.t.:} \\ &\pi_0 Q_c^\pi(s_0, a_0) + (1 - \pi_0) Q_c^\pi(s_0, a_1, \beta_1) \leq \beta \end{aligned}$$

If the values Q_r and Q_c are known for each state/action/budget triplet we can drive the agent locally according to this equation.

For tree-shaped BMDP like this one, the global optimization problem can always be separated in independent sub-problems (one for each sub-tree)[1]. Intuitively speaking, the agent optimises its budget distribution over all possible actions.

[1]To the best of our knowledge, this separability assumption is still a conjecture for general BMDPs (with loops).

4.2 Training Algorithm

The preceding example can be extended to continuous-states environments by the use of function approximation. In BFTQ, two functions have to be approximated. The regular Q-function estimating the expected discounted sum of rewards, denoted here by Q_r, is defined on a state-action-budget triplet. The constraint Q-function for the sum of constraints, denoted here by Q_c, is defined on a state-action-budget triplet as well. For both Q-functions, the budget variable contextualises with the current budget remaining when evaluating the function.

Notation	Description
n	Current iteration of BFTQ.
i	Index of the i^{th} transition.
$f(x_i) \xleftarrow{reg} y_i$	Regression of f using database $\{(x_i, y_i)\}_{i \in [1,N]}$.
$\Delta_{\mathcal{A}}$	Probability simplex over $\mathcal{A}$.

Table 1: Notations used in BFTQ

In Fig. 3, the environment transitions, rewards and constraints were assumed to be known. In most situations, the model of the environment is unknown. FTQ, like many others RL algorithms, overcomes this problem by using the agent interactions with the environment through a trials and errors process. Those interactions are collected as a base of transitions. To take the budget into account while constructing the optimal policy in BFTQ, this training base is enriched as a list: $(s_i, a_i, r'_i, s'_i, c', \beta_i)_{i \in [0,N]}$ where r_i and c_i denote respectively the effective rewards and the costs of the transition and the β_i are taken from a discretization of the admissible budgets[2].

The training phase of BFTQ relies on two policies: with the notations summarized on Table 1, the behavioural-policy $\pi(s, a, \beta) : \mathcal{S} \times \mathbb{R} \to \Delta_{\mathcal{A}}$ is the classical stochastic policy mapping states to distributions over actions, augmented with the budget dependency. The budget-policy $b(s, a, \beta) : \mathcal{S} \times \mathcal{A} \times \mathbb{R} \to \mathbb{R}$ allocates constraint budget to each triplet (action, state, budget).

The BFTQ algorithm consists in iterating over its two Q-functions: the reward action-value Q-function Q_r^n is evaluated with Eq. 5, and the constraint action-value Q-function Q_c^n is evaluated with Eq. 6. Both can be trained with any supervised learning regression algorithm.

$$\begin{aligned} &Q_r^{n+1}(s_i, a_i, \beta_i) \xleftarrow{reg} \qquad (5) \\ &\qquad r'_i + \gamma \sum_{a' \in \mathcal{A}} \pi^n(s'_i, a', \beta_i) Q_r^n(s'_i, a', b^n(s'_i, a', \beta_i)) \end{aligned}$$

[2]For example, if the cumulative cost of a problem is in interval $[1, 3]$, we can take β in $\{1 + n/100 \mid 0 \leq n \leq 200\}$.

$$Q_c^{n+1}(s_i, a_i, \beta_i) \xleftarrow{reg} \quad (6)$$
$$c'_i + \gamma_c \sum_{a' \in \mathcal{A}} \pi^n(s'_i, a', \beta_i) Q_c^n(s'_i, a', b^n(s'_i, a', \beta_i))$$

Contrary to regular FTQ where the policy is a simple greedy exploitation of the Q-function, BFTQ has to compute its two policies π^n and b^n at each iteration. Since taking some actions could exceed the budget, both budget-policy and behavioural-policy are jointly optimised under the constraint of belonging to set Ψ, described in Eq. 7. This equation denotes the search space for admissible (behavioural-policy, budget-policy) couples. Eq. 8 describes the optimisation process.

$$\Psi^n = \left\{ \begin{array}{l} \pi \in \Delta_{\mathcal{A}}^{\mathcal{S}\times\mathbb{R}}, b \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}\times\mathbb{R}}, \\ \text{such that, } \forall s \in \mathcal{S}, \forall \beta \in \mathbb{R}, \\ \sum\limits_{a\in\mathcal{A}} \pi(s,a,\beta) Q_c^n(s,a,b(s,a,\beta)) \leq \beta \end{array} \right\} \quad (7)$$

$$(\pi^{n+1}, b^{n+1}) \leftarrow \quad (8)$$
$$\underset{(\pi,b)\in\Psi^{n+1}}{\operatorname{argmax}} \sum_{a\in\mathcal{A}} \pi(s,a,\beta) Q_r^{n+1}(s,a,b(s,a,\beta))$$

The stopping criterion of BFTQ is classically defined as a threshold η on Q-functions parameters differences.

The main result of this training phase is the couple of fitted Q-functions Q_r and Q_c. As a side result we also obtain approximations of the optimal action and budget policies π_i and b_i for each transition of the training base.

4.3 Theoretical Policy Execution

Let β_t and s_t be respectively the remaining budget and state of the agent at time step t. The agent can theoretically use the two Q-functions Q_r and Q_c according to Eq. (8) to determines the optimal distribution $\pi(s_t, \cdot, \beta_t)$ over the actions and the corresponding budget repartition $b(s_t, \cdot, \beta_t)$. It then samples an action according to the distribution and replaces its current budget according to the budget repartition and the action sampled.

$$a_t \sim \pi(s_t, \cdot, \beta_t) \quad (9)$$
$$\beta_{t+1} = b(s_t, a_t, \beta_t) \quad (10)$$
$$s_{t+1} \sim P(s_t, a_t, \cdot)$$

The advantage of this approach is that it only approximates on Q_r and Q_c. Its downside is that one has to solve a non-linear problem under constraints at each time step of the policies execution: for time-critic applications, its use would be disastrous.

4.4 Policies approximations

Having to solve Eq. (8) is painful for both the training and deployment phases. To overcome this issue, we approximate the functions π and b using regressions on the training transitions. The optimisation described in Eq. 8 does not need to be computed exhaustively: it has to be performed only for the transitions in the learning base. For this purpose, we use the sets ψ_i^n, as defined in Eq. (11)), which are the projections of the set ψ^n of Eq. (7) on the training transitions.

$$\Psi_i^n = \left\{ \begin{array}{l} \pi_i \in \Delta_{\mathcal{A}}, b_i \in \mathbb{R}^{\mathcal{A}}, \text{ such that:} \\ \sum\limits_{a\in\mathcal{A}} \pi_i(a) Q_c^n(s_i, a, b_i(a)) \leq \beta_i \end{array} \right\} \quad (11)$$

The projection of Eq. (8) on the training set is formalized in Eq. 12.

$$\pi^{n+1}(s_i, \cdot, \beta_i), b^{n+1}(s_i, \cdot, \beta_i) \xleftarrow{reg} \quad (12)$$
$$\underset{(\pi_i,b_i)\in\Psi_i^{n+1}}{\operatorname{argmax}} \sum_{a\in\mathcal{A}} \pi_i(a) Q_r^{n+1}(s_i, a, b_i(a))$$

At the end of the training process, we can drop the Q-functions and work directly with the fitted policies as in Eq. (9) and Eq. (10).

4.5 Solving the non-linear problem

The main remaining challenge is to solve the non-linear problem Eq 12 during the training phase. One may use some black-box optimization algorithm like random sampling, SLSQP (Kraft and Schnepper (1989)) but it would be inefficient. Hopefully we can exploit some specific properties of the Q-functions to speed-up the optimization. Let $Q_{s,a}(\beta)$ denote the following parametric function:

$$Q_{s,a}(\beta) \rightarrow (Q_c(s,a,\beta), Q_r(s,a,\beta)) \quad (13)$$

This function is concave and strictly increasing in β along the axis of Q_r. Indeed, given a budget β, a state s and an action a, the expected reward and actual budget consumption for a given action, $Q_r(s,a,\beta)$ and $Q_c(s,a,\beta)$ will be lower or equal to the ones yield by a larger budget $\beta' = \beta + \epsilon \; \forall \epsilon \geq 0$. The intuition is as follows: given a budget β', in worst case scenarios (when extra budget ϵ is useless), one can always achieve at least the same performances as with the lower budget β; indeed, the space of admissible policies for β' contains the one for β.

We need to solve Eq 8 for a state s. For the moment, we assume to have 2 possible actions $a_\perp$ and $a_\top$. The functions $Q_{s,a_\perp}$ and $Q_{s,a_\top}$ are plotted on Fig 4. The optimal Q_r value is necessarily obtained by the probabilist combination of those actions. This combination of actions yield values (Q_c, Q_r) on a straight line. Since Q functions are concave and strictly increasing, the line defining

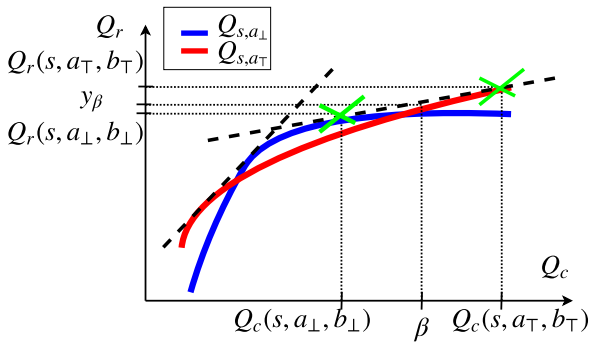


Figure 4: An example of Q-functions as defined in Eq 13. The dot lines are tangent to both curves. The tangent at the right defines the optimal policy when the budget β lies between $Q_c(s,a_\perp,b_\perp)$ and $Q_c(s,a_\top,b_\top)$. The value y_β is then equal to $\pi(s,a_\perp,\beta)Q_r(s,a_\perp,\beta) + \pi(s,a_\top,\beta)Q_r(s,a_\top,\beta)$.

the optimal and admissible solutions is actually a tangent of both $Q_{s,a_\perp}$ and $Q_{s,a_\top}$, lying on top of both functions as shown on Fig 4. We need to find both intersections points between the tangent and the Q functions to define the optimal solution. Let $(Q_c(s,a_\perp,b_\perp), Q_r(s,a_\perp,b_\perp))$ and $(Q_c(s,a_\top,b_\top), Q_r(s,a_\top,b_\top))$ denote the two intersections points. The optimal budget-policy in s is then defined as $b(s,a_\perp,\beta) = b_\perp$ and $b(s,a_\top,\beta) = b_\top$. The optimal behavioural-policy in s is the following:

$$\begin{aligned}
&\pi(s,a_\top,\beta) + \pi(s,a_\top,\beta) = 1,\\
&\text{if } \beta \le Q_c(s,a_\perp,b_\perp), \quad \pi(s,a_\top,\beta) = 0,\\
&\text{if } \beta \ge Q_c(s,a_\top,b_\top), \quad \pi(s,a_\top,\beta) = 1,\\
&\text{otherwise, } \quad \pi(s,a_\top,\beta) = \frac{\beta - Q_c(s,a_\perp,b_\perp)}{Q_c(s,a_\top,b_\top) - Q_c(s,a_\perp,b_\perp)}.
\end{aligned} \tag{14}$$

One can extend this reasoning to cases with more than two actions by considering actions two by two. Finally, solving Eq 12 actually amounts to solve the following non linear program:

$$\pi^{n+1}(s_i,\cdot,\beta_i), b^{n+1}(s_i,\cdot,\beta_i) \xleftarrow{reg} \underset{(\pi_i,b_i)\in\Omega_i^{n+1}}{\operatorname{argmax}} \sum_{a\in\mathcal{A}} \pi_i(a) Q_r^{n+1}(s_i,a,b_i(a)) \tag{15}$$

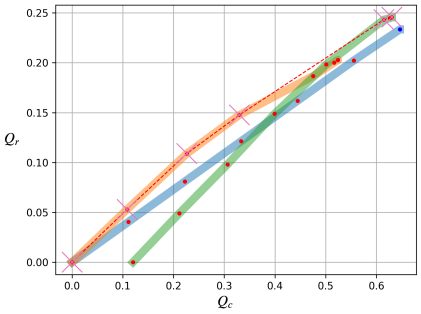


Figure 5: An execution of SOLVE with three actions. Each dots correspond to discretization w.r.t β. Red dots are considered to compute the convex hull. Blue dots are removed since they reach a worst value of Q_r for greater Q_c. Finally, pink cross are the dots retained to compute the optimal policy.

where

$$\Omega_i^n = \left\{ \begin{array}{l} \pi_i, b_i, \text{ such that:} \\ (a_\perp,\ a_\top) \in \mathcal{A}^2, \\ (b_\perp, b_\top) \in \mathbb{R}^2, \\ b_i(a_\top) = b_\top, \\ b_i(a_\perp) = b_\perp, \\ b_i(a) = 0\ \forall a \in \mathcal{A} \setminus \{a_\top, a_\perp\} \\ Q_c^n(s_i,a_\perp,b_\perp) \le \beta_i < Q_c^n(s_i,a_\top,b_\top), \\ \pi_i(a) = 0\ \forall a \in \mathcal{A} \setminus \{a_\top, a_\perp\} \\ \pi_i(a_\perp) = \frac{\beta_i - Q_c^n(s_i,a_\perp,b_\perp)}{Q_c^n(s_i,a_\top,b_\top) - Q_c^n(s_i,a_\perp,b_\perp)}, \\ \pi_i(a_\top) = 1 - \pi_i(a_\perp), \\ \Delta = \Delta_\perp = \Delta_\top \end{array} \right\}$$

$$\text{with } \Delta = \frac{Q_r^n(s_i,a_\top,b_\top) - Q_r^n(s_i,a_\perp,b_\perp)}{Q_c^n(s_i,a_\top,b_\top) - Q_c^n(s_i,a_\perp,b_\perp)}$$

$$\Delta_\perp = \frac{\partial Q_r(s_i,a_\perp,b_\perp)}{\partial\beta} \Big/ \frac{\partial Q_c(s_i,a_\perp,b_\perp)}{\partial\beta}$$

$$\Delta_\top = \frac{\partial Q_r(s_i,a_\top,b_\top)}{\partial\beta} \Big/ \frac{\partial Q_c(s_i,a_\top,b_\top)}{\partial\beta}$$

One way to solve Eq 15 is to compute the convex hull of discrete version of Q functions. We propose an algorithm named SOLVE in Alg 1 to do so and show an example of its execution in Fig 5. We assume that the function $hull$ returns the convex hull $\mathcal{H}$ of a set in clockwise order. It starts with the lowest value in Qc, then the corresponding budgets $\mathcal{B}$ and finally the corresponding actions $\mathcal{A}_\mathcal{P}$[3]. We also need to operate a reasonable discretization of Q-function and reduce the search space. For this reason we need to know the maximal cumulated constraint the environment could return. We call it β_{max}.

[3]The Graham Scan (Graham (1972)) fits all those criteria.

Algorithm 1 SOLVE

In: $s, Q_r, Q_c, K, \beta, \beta_{max}$
Out: π, b
$\mathcal{P} = \emptyset$
for a **in** $\mathcal{A}$ **do**
  for $i = 0$ **to** K **do**
    $step = i/K * \beta_{max}$
    $p = (Q_c(s, a, step), Q_r(s, a, step))$
    $\mathcal{P} = \mathcal{P} \cup p$
  end for
end for
$\mathcal{H}, \mathcal{B}, \mathcal{A}_\mathcal{P} = hull(\mathcal{P})$
$found = False$
$i = 0$
while $\neg\, found$ **do**
  $q_c, q_r = \mathcal{H}[i]$
  $found = \beta \geq q_c$
  $i = i + 1$
end while
$\pi = [0 \dots 0]$
$b = [0 \dots 0]$
if $i < |\mathcal{H}|$ **then**
  $q_c^\top, q_r^\top, q_c^\bot, q_r^\bot = \mathcal{H}[i], \mathcal{H}[i-1]$
  $p_\bot = (\beta - q_c^\bot)/(q_c^\top - q_c^\bot)$
  $b_\bot, b_\top = \mathcal{B}[i-1], \mathcal{B}[i]$
  $a_\bot, a_\top = \mathcal{A}_\mathcal{P}[i-1], \mathcal{A}_\mathcal{P}[i]$
  $\pi[a_\bot], \pi[a_\top] = p_\bot, 1 - p_\bot$
  $b[a_\bot], b[a_\top] = b_\bot, b_\top$
else
  $a_\bot = \mathcal{A}_\mathcal{P}[i-1]$
  $\pi[a_\bot] = 1.$
  $b[a_\bot] = \mathcal{B}[i-1]$
end if

Algorithm 2 BFTQ

In: $\{s_i, a_i, \beta_i, r'_i, c'_i, s'_i\}_{i \in [0,N]}$,
fit_r,fit_c,fit_b,$\gamma_c, \gamma, K, \beta_{max}$
Out: π, b
$stop = false, k = 0$
$Q_r, Q_c = \textbf{lambda} : s, a, \beta \rightarrow 0$
$\pi, b = \textbf{lambda} : s, \beta \rightarrow \textbf{0}$
$\forall i \in [0, N]\ y_i^r, y_i^c, y_i^b, y_i^\pi = 0, \Psi_i = \emptyset$
while $\neg stop$ **do**
  for $i = 0$ **to** N **do**
    $y_i^r = r'_i + \gamma \sum_{a' \in \mathcal{A}} \pi(s'_i, a', \beta_i) Q_r(s'_i, a', b(s'_i, a', \beta_i))$
    $y_i^c = c'_i + \gamma_c \sum_{a' \in \mathcal{A}} \pi(s'_i, a', \beta_i) Q_c(s'_i, a', b(s'_i, a', \beta_i))$
  end for
  $Q_r = fit_r(\{s_i, a_i, \beta_i\}_{i \in [0,N]}, \{y_i^r\}_{i \in [0,N]})$
  $Q_c = fit_c(\{s_i, a_i, \beta_i\}_{i \in [0,N]}, \{y_i^c\}_{i \in [0,N]})$
  for $i = 0$ **to** N **do**
    $y_i^\pi, y_i^b = SOLVE(s_i, Q_r, Q_c, K, \beta, \beta_{max})$
  end for
  $\pi = fit_\pi(\{s_i, \beta_i\}_{i \in [0,N]}, y_i^\pi)$
  $b = fit_b(\{s_i, \beta_i\}_{i \in [0,N]}, y_i^b)$
  $k = k + 1$
  $stop = k > K$ or ("π and b didn't change too much")
end while

Finally, Alg. 2 recalls the complete BFTQ algorithm using Eq. 15. The algorithm should stop when b and π don't change too much between two iterations. The distances may use parameters of π and b approximations. Note that, since each transition has been extended with several values of β, at each iteration, you can save extra computations by pre-computing the convex hulls for each state in the batch of transitions and re-use these hulls for each β.

In the next section, we evaluate BFTQ on a set of 2-D worlds.

5 Experiments

We do experiments on $N \times N$ 2-D worlds with continuous state-space. The agent starts in top-left corner $(0, 0)$. The goal is to reach the bottom-right corner, at coordinates $(N-1, N-1)$, which ends the episode with a immediate reward of 1. On its way to the goal, the agent may fall into randomly placed holes. In that case, it also ends the episode with a constraint of magnitude 1. Consequently, constraints can be seen as failing probability. The same goes for rewards as success probability.There are three actions available: $move\_right$, $move\_bottom$ and $dont\_move$. The two first actions yield stochastic transitions: a Gaussian noise of mean 0 and standard deviation σ is applied. Last action ensures that there always exists a solution for any starting state if β is zero.

In the following experiments, we use two different RL agents: BFTQ(β) uses BFTQ algorithm to learn its policy. Q_r, Q_c, π and b are approximated with regression trees. The algorithm used for computing the convex hull is the Graham scan (Graham (1972)). β is the budget used when initiating a new episode. Note that all BFTQ(β) agents use Eq. 5 and 6. The second agent, FTQ(λ), uses FTQ with a regression tree. The penalty λ is used when rewarding a fall into a hole : $r \leftarrow r - \lambda * c$. The training dataset is uniformly distributed over the state-action sets.

On the first experiment, we compare BFTQ(β) with FTQ(λ) on a single 5×5 world. Results are displayed on Fig. 6. For a given constraint return, both agents achieve a similar reward return. In this setting, BFTQ(β) is slightly optimistic: until $\beta = 0.3$, there is an offset of 0.05 between the given budget and the constraint return. So BFTQ(β) may not fully respect the given budget. But the same phenomena occurs for FTQ(λ): when $\lambda = 10^4$, we may think that the algorithm should return a constraint of value 0, but it is actually 0.07. This is because of the approximations involved in both algorithms. Note that when the constraint return exceeds 0.4, failing probabilities are capped as well as rewards. The explication is as follows: at some point, the agent had to take some risk to get the reward. But taking more risk won't necessary give

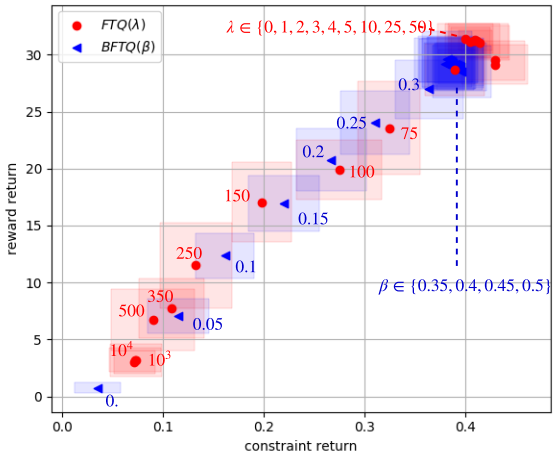


Figure 6: BFTQ(β) and FTQ(λ) reward w.r.t the constraints. Values near the dots are the budgets β allowed for BFTQ and the value λ for FTQ. Each dot is the mean on 9 runs on a 5x5 2-D worlds with 1000 test episodes on each of them. The standard deviation plotted refer to the variability between all test episodes.

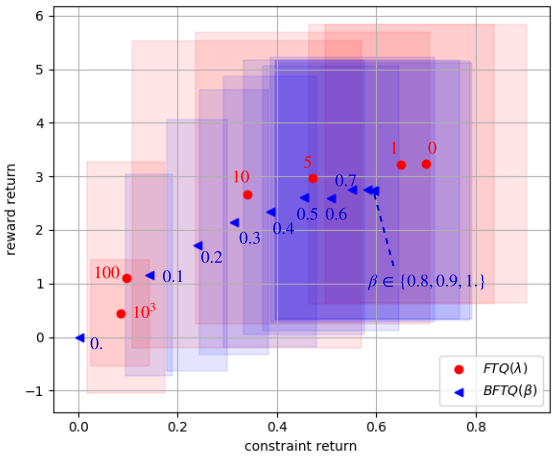
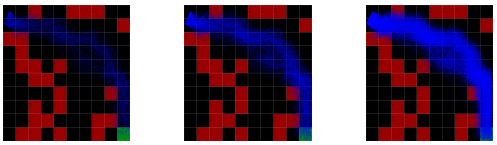


Figure 7: Each dot is the mean on 30 2-D worlds with 1000 test episodes on each of them. The standard deviation plotted refer to the variability between the random 2-D worlds.



(a) $\beta = 0.1$ (b) $\beta = 0.2$ (c) $\beta = 0.6$

Figure 8: Execution of a BFTQ(β) policy on a random 10x10 2-D world. Test episodes are in blue, constraints in red and reward in green.

it more reward, since the path to the reward is not that risky. We can also observe a clear disparity between the λ values and their corresponding budgets. For instance it seems quite difficult to guess the right λ between 100 and 150 in order to stay tightly under a budget of value 0.25 without relaunching simulations. On the contrary, the BFTQ(β) policy allows one to fix any budget bound without worrying about the reward design.

On the second experiment, we compare the two agents on 30 randomly generated 10×10 worlds. Results are displayed on Fig. 7. Once again performances between algorithms are similar, although FTQ(λ) reaches slightly better reward return than BFTQ(β) for the same constraint return. This is mainly explained by the approximation of the behavioural/budget policies and the discretization of Q-functions involved in the computing of the convex hull. We can also notice that, on Fig. 7, BFTQ(β) agent's performances are less noisy than FTQ(λ)'s performances. The explication is as follows: the budget choosing phase is invariable for any world while the λ used in FTQ(λ) to keep the constraint return under a particular value may be different for two worlds. This is why FTQ(λ) needs a calibration phase - which is impossible when environment can not be simulated - while BFTQ(β) only needs to specify a budget.

Finally, some examples of BFTQ(β)'s policies executions are shown in Fig. 8. For a low budget, $\beta = 0.1$, the agent stops its episode after 1 or 2 actions. For $\beta = 0.2$, the agent tries to reach the goal with more insistence but still stops some episodes prematurely (selecting $dont\_move$ action). For $\beta = 0.6$, the agent does not stop until it reaches the goal.

6 Conclusion

We have introduced BFTQ, a direct extension of FTQ to solve BMDP problem. To scale things up, we proposed to learn the policy using supervised learning. We have also shown that in continuous BMDP, Q-functions are concave and increasing. We exploited this property to approximately solve more efficiently non-linear program of BFTQ through a dedicated algorithm that we named SOLVE. Finally, we tested the algorithm on continuous 2-D worlds: these experiments confirm that BFTQ performs as well as a penalized FTQ in terms of reward/constraint raised couple without the need of a simulated calibration phase.

We believe that classic single reward oriented reinforcement learning algorithms could be extended to solve BMDP in the same way. It may also be of interest to find a close form solution for the program in Eq 15. This would speed up considerably the computation time of BFTQ and avoid the need of approximating policies.

References

Naoki Abe et al. Optimizing debt collections using constrained reinforcement learning. In *SIGKDD*, 2010. URL http://doi.acm.org/10.1145/1835804.1835817.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *ICML*, 2017. URL http://proceedings.mlr.press/v70/achiam17a.html.

Eitan Altman. *Constrained Markov Decision Processes*. CRC Press, 1999.

Richard Bellman. Dynamic programming and lagrange multipliers. *National Academy of Sciences of the USA*, 1956.

Dimitri P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods (Optimization and Neural Computation Series)*. Athena Scientific, 1996.

Frederick J. Beutler and Keith W. Ross. Optimal policies for controlled markov chains with a constraint. *Math. An. and App.*, 112:236 – 252, 1985. URL http://www.sciencedirect.com/science/article/pii/0022247X85902884.

Craig Boutilier and Tyler Lu. Budget allocation using weakly coupled, constrained markov decision processes. In *UAI*, 2016. URL http://auai.org/uai2016/proceedings/papers/246.pdf.

Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *JMLR*, 2018. URL http://jmlr.org/papers/v18/15-636.html.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-Based Batch Mode Reinforcement Learning. *JMLR*, 2005.

Javier García and Fernando Fernández. A Comprehensive Survey on Safe Reinforcement Learning. *JMLR*, 2015. URL http://jmlr.org/papers/v16/garcia15a.html.

Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence*, 24, 2005.

Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1972.

Dieter Kraft and Klaus Schnepper. Slsqp, a nonlinear programming method with quadratic programming subproblems. *DLR, Oberpfaffenhofen*, 1989.

Romain Laroche and Paul Trichelair. Safe Policy Improvement with Baseline Bootstrapping. *CoRR*, 2017.

Mohammad Petrik, Marek Ghavamzadeh, , and Yinlam Chow. Safe policy improvement by minimizing robust baseline regret. In *NIPS*, 2016. URL https://papers.nips.cc/.

Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In *ICML*, 2015. URL http://proceedings.mlr.press/v37/thomas15.html.

Aditya Undurti, Alborz Geramifard, Nicholas Roy, and Jonathan P How. Function Approximation for Continuous Constrained MDPs. Technical report, 2010.