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 NICOLAS.CARRARA@ORANGE.COM
Orange Labs, Lannion, France
Univ. Lille, CNRS, Centrale Lille, INRIA UMR 9189 - CRIStAL, Lille, France

Romain Laroche ROMAIN.LAROCHE@MICROSOFT.COM
Microsoft Maluuba, Montréal, Canada

Jean-Léon Bouraoui JEANLEON.BOURAOUI@ORANGE.COM
Tanguy Urvoy TANGUY.URVOY@ORANGE.COM
Orange Labs, Lannion, France

Olivier Pietquin OLIVIER.PIETQUIN@UNIV-LILLE1.FR
Univ. Lille, CNRS, Centrale Lille, INRIA UMR 9189 - CRIStAL, Lille, France

Abstract

We address the problem of budgeted reinforcement learning, in continuous state-space, using a batch of transitions. To this extend, we introduce a novel algorithm called Budgeted Fitted-Q (BFTQ). Benchmarks show that BFTQ performs as well as a regular Fitted-Q algorithm in a continuous 2-D world but also allows one to choose the right amount of budget that fits to a given task without the need of engineering the rewards. We believe that the general principles used to design BFTQ can be applied to extend others classical reinforcement learning algorithms for budgeted oriented applications.

Keywords: Reinforcement Learning, Budgeted-MDP, Fitted-Q

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 both optimize its travel time, its fuel consumption, and 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 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 because: one cannot afford to work "on expectation"; 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 constraints. The car should optimize the expected travel time while guaranteeing a crash probability strictly lower than the one of a human driver. In this case, methods using worst percentile scenario criteria like *conditional value-at-risk* (CVaR) (Chow et al. (2015)) may help to design a good strategy.

One can also consider a scenario where a military send drones to a survey mission, and can only afford to lose a known limited percentage. In such scenario, it is difficult to shape the drone casualties/cost in term of negative rewards but there is no need to deploy the whole CVaR machinery. The aim of our work is to handle a relaxation of the risk-sensitive

RL problem where we want to maximize the expected primary reward while keeping a secondary criterion – on expectation – under a given budget. We formalize this problem as a *Budgeted Markov Decision Process* (BDMP) in continuous state-space (Boutilier and Lu (2016)). Solving a BMDP is a harder problem than solving a *Constrained Markov Decision Process* (CDMP) (Altman, 1999) as it requires the budget to be a parameter of the policy.

We introduce Budgeted Fitted-Q (BFTQ), a practical extension of the Fitted-Q (FTQ) algorithm for BMDP. It takes as input a batch of RL transitions and estimates the optimal parametric strategies under budget. We experiment BFTQ on 2-D world with holes and we show that BFTQ performs as well as FTQ and that the obtained policy is not shaped for a specific budget.

2. Background and Related Works

2.1 Markov Decision Process

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

Fitted-Q (FTQ, Ernst et al. (2005)) is a generic algorithm to solve MDPs with continuous state-space and unknown R and P. It implements *Value-Iteration* (V.I, (Bellman (1956)) with a regression model as a proxy for the Q-function. It uses a batch of transitions $(s_i, a_i, r'_i, s'_i)_{i \in [0,N]}$ which covers most of the state space. It trains the Q-function at each iteration of VI using a supervised learning algorithm that regresses the following supervised training batch: $\{(s_i, a_i),\ r'_i + \gamma_r * \max_{a'} Q(s'_i, a')\}_{i \in [0,N]}$.

2.2 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 $\pi^* = \text{argmax}_\pi \mathbb{E}_\pi \sum_t \gamma_r^t R_t$ s.t. $\mathbb{E}_\pi \sum_t \gamma_c^t C_t \leq \beta$ (1) where $R_t = R(s_t, a_t, s_{t+1})$ and $C_t = C(s_t, a_t, s_{t+1})$.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 $\beta > 0$.

A convenient property of MDPs, is that the optimal policy is unique, deterministic, and greedy: $\pi^*(s) = \text{argmax}_a Q(s, a)$. 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

1. We usually write $\pi(s)$, for $s \in \mathcal{S}$, as a random variable in $\mathcal{A}$.

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 picking always the *safe* action is feasible for any $\beta \geq 0$, but if $\beta = 1/2$, the most rewarding feasible policy is to pick randomly 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.

2.3 Related work

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 Fitted-Value-Iteration but it assumes that the environment (R, C and P) is known. Geibel and Wysotzki (2005); Chow et al. (2015); Achiam et al. (2017) propose algorithms for CMDP, but they require interactions with the environment during training (actor-critic, Q-learning and Policy Gradient, Constrainted-Policy-Optimization). Abe et al. (2010) work with constraint-batch 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. Finally, one of the main drawback of most of all these algorithms is the fact that they resolve CMDP and not BMDP. The approaches relying on using Lagrangian/KKT conditions would fall under the same limitation. On the contrary, Boutilier and Lu (2016) resolve a BMDP using a VI-like algorithm. However it only applies to finite and known environments. The next section introduce the contribution of the paper: Budgeted Fitted-Q.

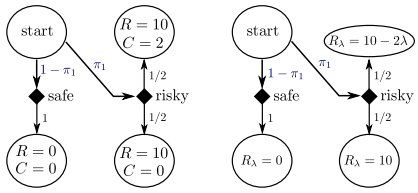


Figure 1: On the left hand side, a simple *risky-vs-safe* BMDP. The probability of picking the risky action is π_1. On the right hand side an attempt to relax the problem with negative rewards.

3. 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.

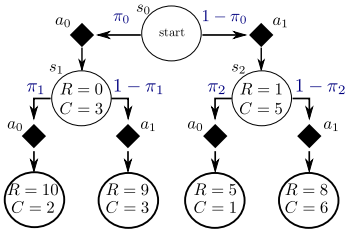


Figure 2: A simple deterministic finite BMDP.

3.1 Intuition

We illustrate the principle of BFTQ with a simple BMDP described on Fig. 2. On this example, the transition, reward, and constraint functions are fully deterministic and finite, and the state and action sets are of respective sizes 7 and 2. Starting from the initial state, the goal is to maximise the expected cumulative reward under constraint that the expected cumulative cost lays 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 $\pi_2 = \mathbb{P}(a_0 \mid s_2)$. On the left hand side of the figure, on state s_1, the action a_0 leads to a better reward $R = 10 > 9$ and a lesser cost $C = 2 < 3$ than action a_1: this action is dominant for any budget. On the right hand side state s_2, we retrieve the same *safe/unsafe* trade-off as in Figure 1: the best policy has to be mixed between actions a_0 and a_1.

Assuming $\gamma_r = \gamma_c = 1$, the optimization problem of Eq. 1 becomes:

$$\begin{aligned} \max_{\pi_0,\pi_1,\pi_2} \quad & \pi_0 \cdot [10\pi_1 + 9(1-\pi_1)] + (1-\pi_0) \cdot [1 + 5\pi_2 + 8(1-\pi_2)] \\ \text{s.t.} \quad & \pi_0 \cdot [3 + 2\pi_1 + 3(1-\pi_1)] + (1-\pi_0) \cdot [5 + \pi_2 + 6(1-\pi_2)] \leq \beta \end{aligned}$$

In this example, one can see that we may easily separate the constrained optimization problem into one independent sub-problem for each sub-tree[2]. For that purpose, a variable substitution is operated as follows. Intuitively speaking, the agent optimises its budget distribution over all possible actions:

$$\max_{\pi_0,\beta_0,\beta_1} \pi_0 Q_r^\pi(a_0,\beta_0) + (1-\pi_0) Q_r^\pi(a_1,\beta_1) \text{ s.t. } \pi_0 Q_c^\pi(a_0,\beta_0) + (1-\pi_0) Q_c^\pi(a_1,\beta_1) \leq \beta$$

$$\text{where} \begin{cases} Q_r^\pi(a_0,\beta_0) = \max_{\pi_1} [10\pi_1 + 9(1-\pi_1)] \text{ s.t. } 2\pi_1 + 3(1-\pi_1) \leq \beta_0 \\ Q_r^\pi(a_1,\beta_1) = \max_{\pi_2} [1 + 5\pi_2 + 8(1-\pi_2)] \text{ s.t. } \pi_2 + 6(1-\pi_2) \leq \beta_1 \\ Q_c^\pi(a_0,\beta_0) = 3 + \beta_0 \text{ and } Q_c^\pi(a_1,\beta_1) = 5 + \beta_1 \end{cases}$$

This simplifies into:

$$\max_{\pi_0,\beta_0,\beta_1} 10\pi_0 + (1-\pi_0) \cdot (27 + 3\beta_1)/5 \text{ s.t. } \pi_0(3+\beta_0) + (1-\pi_0)(5+\beta_1) \leq \beta$$

3.2 Algorithm

While Fig. 2 illustrates the method on a simple BMDP, it can be extended to non-finite environments. If the number of states is very large or even infinite, one needs to use function approximations. In BFTQ, two functions have to be approximated. The regular Q-function estimates the discounted sum of rewards; it is denoted here by Q_r, and is defined on a state-action-budget tuple. The constraint Q-function estimates the discounted sum of constraints, denoted here by Q_c; it is defined on a state-action-budget tuple as well. For both Q-functions, the budget variable contextualises with the remaining current budget when evaluating the function. In the intuition example, budget variables are β_0 and β_1. In Fig. 2, 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 agent's interactions with the environment through a trial and error process. Those interactions are collected as transitions $(s_i, a_i, r'_i, s'_i)_{i\in[0,N]}$. Since BFTQ solves a BMDP, the transitions are extended to $(s_i, a_i, r'_i, s'_i, c'_i)_{i\in[0,N]}$. Because of the budget dependence of the BFTQ functions, the transitions must be further extended with budget values precise enough to capture the budget variations of each trajectory. The

2. To the best of our knowledge, for general BMDP – with loops – this separability assumption is still a conjecture.

new transitions are denoted as $(s_i, a_i, r'_i, s'_i, c'_i, \beta_i)_{i\in[0,N]}$[3]. With this method, we will be able to construct the optimal policy for any input couple (s, β) corresponding to the current state and budget remaining. The following notations are used: n is the current iteration of BFTQ, i the index of the i^{th} transition, $f(x_i) \xleftarrow{reg} y_i$ is the regression of f using database $\{(x_i, y_i)\}_{i\in[1,N]}$ and $\Delta_{\mathcal{A}}$ is the probability simplex over $\mathcal{A}$.

BFTQ relies on two policies. 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). BFTQ algorithm consists in iterating over its two Q-functions: the reward action-value Q-function Q_r^n is evaluated with Eq. 2, and the constraint action-value Q-function Q_c^n is evaluated with Eq. 3. Both may be trained with any supervised learning regression algorithm.

$$Q_r^{n+1}(s_i, a_i, \beta_i) \xleftarrow{reg} r'_i + \gamma_r \sum_{a'\in\mathcal{A}} \pi^n(s'_i, a', \beta_i) Q_r^n(s'_i, a', b^n(s'_i, a', \beta_i)) \tag{2}$$

$$Q_c^{n+1}(s_i, a_i, \beta_i) \xleftarrow{reg} c'_i + \gamma_c \sum_{a'\in\mathcal{A}} \pi^n(s'_i, a', \beta_i) Q_c^n(s'_i, a', b^n(s'_i, a', \beta_i)) \tag{3}$$

$$\Psi^n = \left\{ \begin{array}{l} \pi \in \Delta_{\mathcal{A}}^{\mathcal{S}\times\mathbb{R}}, b \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}\times\mathbb{R}} \text{such that, } \forall s \in \mathcal{S}, \forall \beta \in \mathbb{R}, \\ \sum_{a\in\mathcal{A}} \pi(s, a, \beta) Q_c^n(s, a, b(s, a, \beta)) \leq \beta \end{array} \right\} \tag{4}$$

$$(\pi^{n+1}, b^{n+1}) \leftarrow \underset{(\pi,b)\in\Psi^{n+1}}{\operatorname{argmax}} \sum_{a\in\mathcal{A}} \pi(s, a, \beta) Q_r^{n+1}(s, a, b(s, a, \beta)) \tag{5}$$

Similarly to regular FTQ, BFTQ updates iteratively its greedy policy (π^n, b^n). 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. 4, which denotes the search space for admissible (behavioural-policy, budget-policy) couples. Eq. 5 denotes the optimisation process. The optimisation described in Eq. 5 does not need to be explicitly computed, it can be performed as required. In the BFTQ algorithm, one does only need to compute Eq. 5 for the transitions in the learning base. A complete method to solve of Eq. 5 is described in Appendix A and B. The stop criterion of BFTQ is classically defined as a threshold η on Q-functions's parameters differences.

3.3 Policy Execution

The way an agent must use the resulting behavioural-policy π and budget-policy b over a trajectory is formalised in Eq. 6 and 7. Let β_0 be any initial budget and s_0 be any initial state. At each time step, the agent determines the distribution over the actions and the budget repartition. Then, it samples an action according to the distribution and replaces its current budget according to the budget repartition and the action sampled.

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

4. Experiments

3. For example, if the maximum cumulative constraint of a problem is 1, then a basic way to create those transitions is to duplicate each transition of $(s_i, a_i, r'_i, s'_i, c'_i)_{i\in[0,N]}$ with all β in the set $\{\beta : \exists n \in \mathbb{N}$ such that $\beta = 0.1n$ and $\beta \in [0, 1]\}$.

$$a_t \sim \pi(s_t, \cdot, \beta_t) \quad (6)$$
$$\beta_{t+1} = b(s_t, a_t, \beta_t) \quad (7)$$
$$s_{t+1} \sim P(s_t, a_t, \cdot)$$
$$r_{t+1} \sim R(s_t, a_t, s_{t+1})$$
$$c_{t+1} \sim C(s_t, a_t, s_{t+1})$$

We carry out experiments on randomly generated **continuous** 10×10 2-D worlds. The agent starts in top-left corner $(0, 0)$. The goal is to reach the bottom-right corner, at coordinates $(9, 9)$, which ends the episode with an 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. There are three available actions: *move_right*, *move_bottom* and *dont_move*. The two first actions yield stochastic transitions: a Gaussian noise of mean 0 and standard deviation $\sigma = 0.5$ is applied on the direction followed.

In the following experiments, we use two different RL agents. The first one, 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), details are left in Appendix B). β is the budget used when initiating a new trajectory. Note that all BFTQ(β) agents use Eq. 2 and 3. The second agent: FTQ(λ) uses FTQ with a regression tree. λ is used when penalizing a fall into a hole: $r \leftarrow r - \lambda c$. We evaluate BFTQ($\beta$) on 30 randomly generated 10×10 **continuous** 2-D worlds. In each grid world, 25% of the cells are randomly elected as holes. The training dataset is uniformly distributed over the state-action sets (30 transitions of 3 actions in 400 states -each step of value 0.5 -). The trained policy is evaluated on a total of 10^4 trajectories.

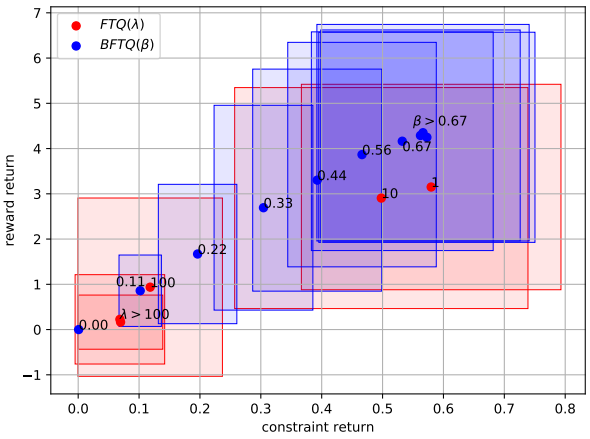


Figure 3: BFTQ(β) and FTQ(λ) reward w.r.t the constraints. Values near the dots are the budgets β allowed for BFTQ and the value λ for FTQ. Each dot is the mean on 30 2-D worlds with 1000 test trajectories on each of them. The standard deviation plotted refer to the variability between the random 2-D worlds.

Since holes are constraints of value 1 and are absorbing, the constraints can be seen as failing probability. The same goes for rewards as success probability. In consequence, we test the policies for each budget β - or maximal failing probability - from 0 to 1. Results are shown in Fig. 3. From 0.0 to 0.67, budgets are not exceeded. Actually, the budget is even almost completely spent. This behaviour is typical of under constraint optimisation where optimal solutions are really close to the constraints. The reward also increases accordingly, but from $\beta = 0.67$ to $\beta = 1.0$, failing probability is capped as well as the reward. The explication is the following: at some point, the agent had to take some risk to get the reward; but taking more risk won't necessary give it more reward, because the path to the reward is not that risky. We can also observe the phenomena where either FTQ computes a super safe policy for $\lambda \in \{100, 10^4\}$ either it computes a really risky policy when $\lambda \in \{10, 1\}$. It seems impossible to find the right λ in order to stay bellow a budget of value 0.3 for example. On the other hand, the BFTQ agent is free to choose any budget value it wants to respect without worrying about the reward design.

On Fig 3, BFTQ reaches the same amount of reward return as FTQ. One can even note that, in this experiment, BFTQ is better than FTQ when budget is unlimited and $/\lambda$ is low. In theory, reward returns should be similar. We believe that some generalization errors occurs for FTQ while BFTQ is more robust to themin this particular context. The impact of the generalization and the approximate method used to solve Eq 5 will be analysed in further work .But if one focus on the BFTQ algorithm, assumes regression algorithms used are perfect and optimisation method find the optimal solution of the non linear program in Eq 5, then BFTQ fits the requirement of the BMDP task.

5. Conclusion

In this paper we introduce BFTQ, a direct extension of FTQ to solve BMDP problem. We test the algorithm on 2-D world and we show that BFTQ performs as well as FTQ in terms of reward/constraint raised couple but also enables to carefully choose the amount of budget the agent can spend.

We believe that classic single reward oriented reinforcement learning algorithms can 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 5, which would speed up considerably the computation time of BFTQ.

Appendix A. Scalability of BFTQ

The downside of Eq. 5 is that one has to solve a non-linear problem under constraints each time both policies are called (Eq. 6, 7, 2 and 3). For Eq. 6 and 7, this process is online: it would be disastrous for time-critic applications. To overcome this issue, we propose to approximate π and b using regressions. For that, instead of Eq. 4, one needs to compute ψ_i (Eq. 8), the set of admissible behavioural/budgets policies couple for a set of samples $(s_i, \beta_i)_{j \in N_i}$. One may design those samples using transitions $(s_i, a_i, r'_i, s'_i, c'_i, \beta_i)_{i \in [0,N]}$ but it's not a requirement.

$$\Psi_i^n = \left\{ \begin{array}{l} \pi_i \in \Delta_{\mathcal{A}}, b_i \in \mathbb{R}^{\mathcal{A}}, \text{ such that:} \\ \sum_{a \in \mathcal{A}} \pi_i(a) Q_c^n(s_i, a, b_i(a)) \leq \beta_i \end{array} \right\} \tag{8}$$

Eq. 9 describes how the budget-policy and behavioural-policy are updated at each iteration of BFTQ.

$$\pi^{n+1}(s_i, \cdot, \beta_i), b^{n+1}(s_i, \cdot, \beta_i) \xleftarrow{reg} \underset{(\pi_i, b_i) \in \Psi_i^{n+1}}{\operatorname{argmax}} \sum_{a \in \mathcal{A}} \pi_i(a) Q_r^{n+1}(s_i, a, b_i(a)) \tag{9}$$

Appendix B. Solving the non-linear problem

The main remaining challenge is solving the non-linear problem in Eq 9. One may use random sampling, SLSQP (Kraft and Schnepper (1989)) or some black box optimisation algorithm such as genetic algorithms to find the optimal budget and action policy for a given transition. But it would be inefficient, because it would disregard some properties of the Q-functions. Let $Q_{s,a}(\beta)$ denote the following parametric function: $Q_{s,a}(\beta) \rightarrow (Q_c(s, a, \beta), Q_r(s, a, \beta))$. Q 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 equals 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 since the space of admissible policies for β' contains the one for β.

We need to solve Eq 5 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 the optimal and admissible solutions is actually the tangent of both $Q_{s,a_\perp}$ and $Q_{s,a_\top}$, lying on top of both functions as shown on Fig 4. This line is unique and 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. Then, the optimal behavioural-policy in s is described in Eq 10 and optimal budget-policy in s is then defined in Eq 12.

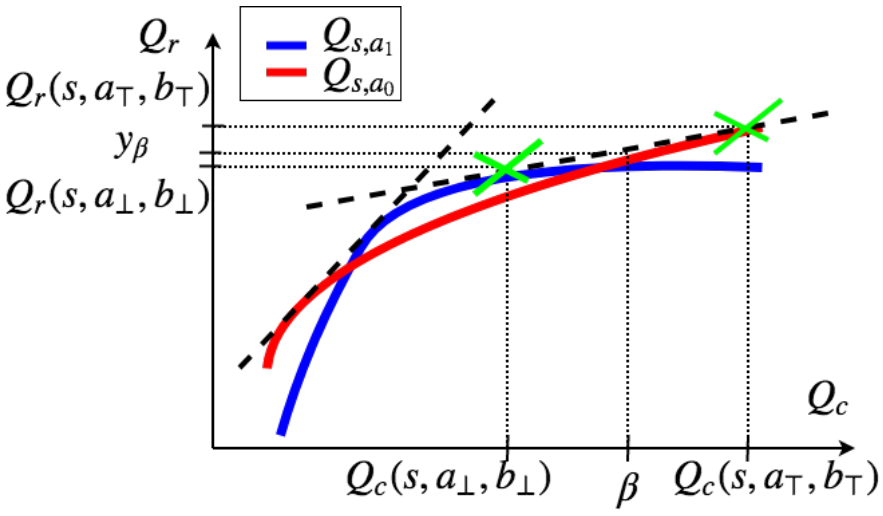


Figure 4: An example of Q-functions. The dot lines are tangent to both curves. The tangent at the right defines the optimal policy when the budget β is defined between $Q_c(s,a_\perp,b_\perp)$ and $Q_c(s,a_\top,b_\top)$. The value y_β is then equal to $\pi(s,a_\perp,\beta)Q_r(s,a_\perp,\beta)+\pi(s,a_\top,\beta)Q_r(s,a_\top,\beta)$.

$$\begin{aligned}
&\pi(s,a_\top,\beta)+\pi(s,a_\top,\beta)=1,\\
&\text{if } \beta \le Q_c(s,a_\perp,b_\perp),\quad \pi(s,a_\top,\beta)=0,\\
&\text{if } \beta \ge Q_c(s,a_\top,b_\top),\quad \pi(s,a_\top,\beta)=1,\\
&\text{otherwise, } \quad \pi(s,a_\top,\beta)=\frac{\beta-Q_c(s,a_\perp,b_\perp)}{Q_c(s,a_\top,b_\top)-Q_c(s,a_\perp,b_\perp)}.
\end{aligned} \tag{10}$$

$$b(s,a_\perp,\beta)=b_\perp \tag{11}$$

$$b(s,a_\top,\beta)=b_\top \tag{12}$$

One can extend this reasoning to cases with more than two actions by considering actions two by two. Finally solving Eq 9 actually amounts to solve the following program:

$$\pi^{n+1}(s_i,\cdot,\beta_i),b^{n+1}(s_i,\cdot,\beta_i) \xleftarrow{reg} \underset{(\pi_i,b_i)\in\Omega_i^{n+1}}{\operatorname{argmax}} \sum_{a\in\mathcal{A}} \pi_i(a)Q_r^{n+1}(s_i,a,b_i(a)) \tag{13}$$

where

$$\Omega_i^n = \left\{ \begin{array}{l} \pi_i, b_i, \text{ such that:} \\ (a_\perp,\ a_\top) \in \mathcal{A}^2, \\ (b_\perp, b_\top) \in \mathbb{R}^2, \\ b_i(a_\top) = b_\top, \\ b_i(a_\perp) = b_\perp, \\ b_i(a) = 0\ \forall a \in \mathcal{A} \setminus \{a_\top, a_\perp\} \\ Q_c^n(s_i, a_\perp, b_\perp) \leq \beta_i < Q_c^n(s_i, a_\top, b_\top), \\ \pi_i(a) = 0\ \forall a \in \mathcal{A} \setminus \{a_\top, a_\perp\} \\ \pi_i(a_\perp) = \frac{\beta_i - Q_c^n(s_i, a_\perp, b_\perp)}{Q_c^n(s_i, a_\top, b_\top) - Q_c^n(s_i, a_\perp, b_\perp)}, \\ \pi_i(a_\top) = 1 - \pi_i(a_\perp), \\ \Delta = \Delta_\perp = \Delta_\top \end{array} \right\} \tag{14}$$

with

$$\begin{aligned} \Delta &= \frac{Q_r^n(s_i, a_\top, b_\top) - Q_r^n(s_i, a_\perp, b_\perp)}{Q_c^n(s_i, a_\top, b_\top) - Q_c^n(s_i, a_\perp, b_\perp)}, \\ \Delta_\perp &= \frac{\partial Q_r(s_i, a_\perp, b_\perp)}{\partial \beta} \Big/ \frac{\partial Q_c(s_i, a_\perp, b_\perp)}{\partial \beta} \\ \Delta_\top &= \frac{\partial Q_r(s_i, a_\top, b_\top)}{\partial \beta} \Big/ \frac{\partial Q_c(s_i, a_\top, b_\top)}{\partial \beta} \end{aligned} \tag{15}$$

One way to solve Eq 13 is to compute the convex hull of the 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 starting with the lowest value in Qc, the corresponding budgets $\mathcal{B}$ and the corresponding actions $\mathcal{A}_\mathcal{P}$[4]. To operate a reasonable discretization of Q-function and reduce the search space, we also need to know the maximal budget the environment could consume, we call it β_{max}.

Finally, Alg. 2 recalls the complete BFTQ algorithm using Eq. 13. The algorithm should stop when b and pi 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.

4. The so-called Graham Scan (Graham (1972)) fits all those criteria.

Algorithm 1 SOLVE

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

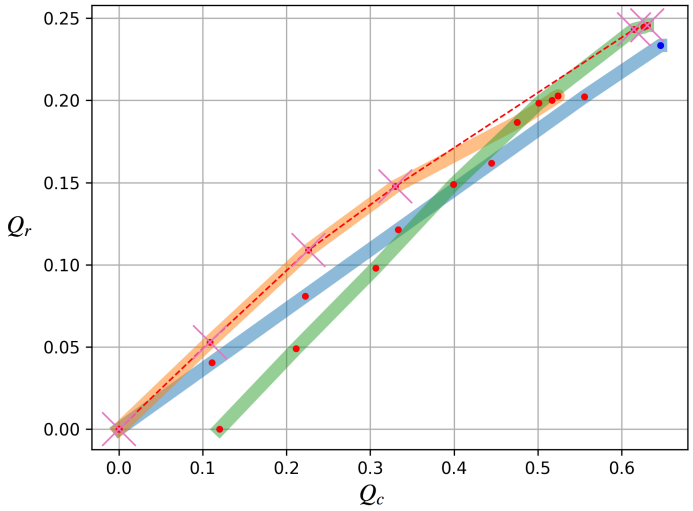


Figure 5: An execution of SOLVE with three actions. Each dot corresponds to the discretization w.r.t β. Red dots are considered to compute the convex hull. Blue dots are removed since they reach a worst value of Q_r for greater Q_c. Finally, pink cross are the dots retained to compute the optimal policy.

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

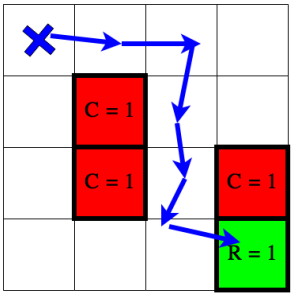
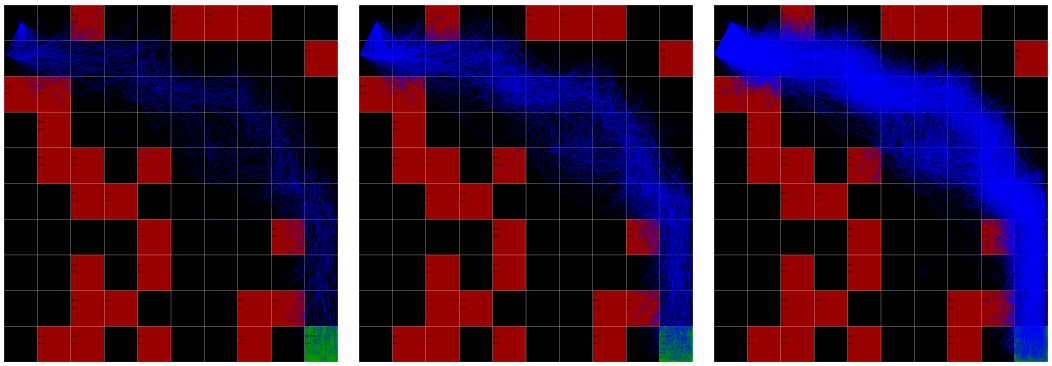


Figure 6: A 4×4 2D world.



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

Figure 7: Execution of a BFTQ policy. Test trajectories are in blue, constraints in red and rewards in green. For a low budget, 0.1, the agent stops its trajectory after 1 or 2 actions. For $\beta = 0.2$, the agent tries to reach the goal with more insistence but still stops some trajectories prematurely. Finally, for $\beta = 0.3$, the agent does not stop until it reaches the goal.

Appendix C. Experiments

Fig 6 describes an example of small 2-D world. Fig 7 describes the trajectories followed by a BFTQ policy for several budgets.

References

Naoki Abe, Prem Melville, Cezar Pendus, Chandan K. Reddy, David L. Jensen, Vince P. Thomas, James J. Bennett, Gary F. Anderson, Brent R. Cooley, Melissa Kowalczyk, Mark Domick, and Timothy Gardinier. Optimizing debt collections using constrained reinforcement learning. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2010.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. *CoRR*, abs/1705.10528, 2017. URL `http://arxiv.org/abs/1705.10528`.

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

Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 1956.

Frederick J. Beutler and Keith W. Ross. Optimal policies for controlled markov chains with a constraint. *Journal of Mathematical Analysis and Applications*, 112(1):236 – 252, 1985. ISSN 0022-247X. doi: https://doi.org/10.1016/0022-247X(85)90288-4. URL `http://www.sciencedirect.com/science/article/pii/0022247X85902884`.

Craig Boutilier and Tyler Lu. Budget allocation using weakly coupled, constrained markov decision processes. In *Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2016.

Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *CoRR*, abs/1512.01629, 2015. URL `http://arxiv.org/abs/1512.01629`.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research*, 2005.

Javier García and Fernando Fernández. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, 2015.

Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence*, 24:81–108, 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 *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

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