Difference of Convex Functions Programming Applied to Control with Expert Data

Bilal Piot, Matthieu Geist, Olivier Pietquin

Noname manuscript No.
(will be inserted by the editor)

Difference of Convex Functions Programming Applied to Control with Expert Data

Bilal Piot[1]
Matthieu Geist[2]
Olivier Pietquin[3]

Received: date / Accepted: date

Abstract This paper reports applications of Difference of Convex functions (DC) programming to Learning from Demonstrations (LfD) and Reinforcement Learning (RL) with expert data. This is made possible because the norm of the Optimal Bellman Residual (OBR), which is at the heart of many RL and LfD algorithms, is DC. Improvement in performance is demonstrated on two specific algorithms, namely Reward-regularized Classification for Apprenticeship Learning (RCAL) and Reinforcement Learning with Expert Demonstrations (RLED), through experiments on generic Markov Decision Processes (MDP), called Garnets.

1 Introduction

The Optimal Bellman Residual (OBR), being a core optimization criterion in Reinforcement Learning (RL), as been recently proven to be a Difference of Convex (DC) functions (Piot et al, 2014c). As a consequence, this paper aims at extending previous results obtained in DC for batch RL to the fields of control with expert data and Learning from Demonstrations (LfD). More precisely, its objective is to leverage the knowledge in DC programming in order to improve the performance of existing methods in control and LfD.

In control theory, there are two canonical ways to make an apprentice agent learn a task from expert demonstrations. The first one consists in directly learning a behaviour (mapping situations to decisions) generalising the expert decisions in states that were unvisited during demonstrations. This is the framework of LfD (Pomerleau, 1989; Atkeson and Schaal, 1997; Schaal, 1997; Argall et al, 2009). The second approach consists in inferring a goal that the apprentice agent should achieve from the demonstrations. The apprentice would than have to interact with the environment and find a strategy to attain it. When this goal is defined trough a reward function representing the local benefit of doing a particular action in a given

[1]Univ. Lille, CRIStAL (UMR CNRS 9189/Lille 1) - SequeL team, bilal.piot@univ-lille1.fr · [2]CentraleSupélec (UMI 2958 GeorgiaTech-CNRS) - IMS-MaLIS team, matthieu.geist@centralesupelec.fr · [3]Univ. Lille, IUF, CRIStAL (UMR CNRS 9189/Lille 1) - SequeL team, olivier.pietquin@univ-lille1.fr, now with Google DeepMind

state, the agent aims at maximising the sum of rewards encountered during its interaction with the environment. This is the framework of Reinforcement Learning (RL) (Sutton and Barto, 1998). From a human perspective, the reward represents a local satisfaction and realising a task consists in maximising the sum of the local satisfactions. RL has the advantage of clearly defining a task through a reward function, providing with an optimisation criterion. However, optimising a sparsely distributed reward is sometimes a tricky task (sparse rewards are mainly encountered in practice because there are easier to define) as the agent has often to explore exhaustively its environment to discover rewards. For this reason, it can be useful to combine RL with LfD (Schaal, 1997; Piot et al, 2014b) in order to avoid learning from scratch and knowing precisely the task to achieve. There is a vast literature on LfD and RL but very few articles on how DC techniques can improve those methods (one exception being (Piot et al, 2014c)).

DC programming (Tao and An, 1997, 2005) can transform a complex non-convex (but DC) problem into a series of simpler convex problems solvable via gradient-descent/ascent methods or Linear Programming (LP). This property is very interesting as it allows leveraging the huge amount of gradient-descent/ascent and LP literature. Thus, DC techniques have been applied to several domains and Machine Learning (ML) is no exception (Le Thi et al, 2014b,a). Indeed, DC methods can be used to address classification tasks (Le Thi et al, 2008; Le et al, 2015) and RL problems (Piot et al, 2014c).

To make use of DC, we place ourselves in the Markov Decision Process (MDP) paradigm, well-suited to study sequential decision making problems in stochastic, discrete-time and finite action-state space environments. In this specific framework, finding an optimal control can be cast into minimising a criterion, namely the OBR, which appears to be a DC function (Piot et al, 2014c). More precisely, we focus on two existing methods called Reward-regularised Classification for Apprenticeship Learning (RCAL) (Piot et al, 2014a) and Reinforcement Learning with Expert Demonstrations (RLED) (Piot et al, 2014b). RCAL is an LfD method which consists in constraining a classification method to using the dynamics of the underlying MDP so as to obtain better generalisation properties. RLED is an RL method using expert demonstrations together with OBR minimisation so as to boost learning speed. Those two algorithms consist in minimising a regularised classification criterion where the regularisation term, which is an empirical version of the OBR, is DC. In their original form, these algorithms where based on standard (sub-)gradient descent, but here we show how using DC techniques can improve the optimisation result.

The remaining of the paper is organised as follows. First, Sec. 2 provides the notations and the background. It introduces the concepts of MDP, RL, IL, RLED and DC functions. Then, in Sec. 3, we show how RCAL and RLED can be decomposed in DC problems. Finally, in Sec .4), we show experimental results.

2 Background

Here, we introduce concepts such as MDP (Sec. 2.1) and RL (Sec. 2.2) which are prerequisites to understanding LfD (Sec.2.3) and RLED (Sec.2.4) frameworks. We also briefly introduce the few basic notions of DC programming (Sec 2.5) required to decompose RCAL and RLED into DC problems.

Let us start with the general notations used throughout this paper. Let $(\mathbb{R}, |.|)$ be the real space with its canonical norm and X a finite set, $\mathbb{R}^X$ is the set of functions from X to $\mathbb{R}$. The set of probability distributions over X is noted Δ_X. Let Y be a finite set, Δ_X^Y is the set of functions from Y to Δ_X. Let $\alpha \in \mathbb{R}^X$, $p \geq 1$ and $\nu \in \Delta_X$, we define the $\mathbf{L}_{p,\nu}$-semi-norm of α, noted $\|\alpha\|_{p,\nu}$, by: $\|\alpha\|_{p,\nu} = (\sum_{x \in X} \nu(x)|\alpha(x)|^p)^{\frac{1}{p}}$. In addition, the infinite norm is noted $\|\alpha\|_\infty$ and defined as $\|\alpha\|_\infty = \max_{x \in X} |\alpha(x)|$. Let v be a random variable which takes its values in X, $v \sim \nu$ means that the probability that $v = x$ is $\nu(x)$.

2.1 Markov Decision Process

The MDP paradigm is a state-of-the-art framework to learn optimal control in a stochastic, discrete-time and finite action-state space environment (Howard, 1960; Bellman and Kalaba, 1965). Via the definition of a reward function representing the local information on the benefit of doing an action in a given state, it allows to find an optimal behaviour w.r.t. a predefined criterion. Here, we consider infinite-horizon γ-discounted MDPs where the chosen criterion is the γ-discounted and expected cumulative reward collected by an agent (a formal definition is given below). An optimal behaviour thus maximises the previous criterion and can be exactly found through Dynamic Programming (DP) (Bertsekas, 1995; Puterman, 1994) techniques such as Value Iteration (VI), Policy Iteration (PI) or Linear Programming (LP).

Here, the agent is supposed to act in a finite MDP represented by a tuple $M = \{S, A, R, P, \gamma\}$ where $S = \{s_i\}_{1 \leq i \leq N_S}$ is the finite state space ($N_S \in \mathbb{N}^*$), $A = \{a_i\}_{1 \leq i \leq N_A}$ is the finite action space ($N_A \in \mathbb{N}^*$), $R \in \mathbb{R}^{S \times A}$ is the reward function ($R(s,a)$ represents the local benefit of doing action a in state s), $\gamma \in]0,1[$ is a discount factor and $P \in \Delta_S^{S \times A}$is the Markovian dynamics which gives the probability, $P(s'|s,a)$, to reach s' by choosing action a in state s. It has been shown (Puterman, 1994) for finite MDPs that it is sufficient to consider deterministic policies in order to obtain an optimal behaviour with respect to the γ-discounted and expected cumulative reward criterion. A deterministic policy π is an element of A^S, it maps each state to a unique action and thus defines the behaviour of an agent. The quality of a policy π is defined by the action-value function. For a given policy π, the action-value function $Q^\pi \in \mathbb{R}^{S \times A}$ is defined as:

$$Q^\pi(s,a) = \mathbb{E}^\pi[\sum_{t=0}^{+\infty} \gamma^t R(s_t, a_t)],$$

where $\mathbb{E}^\pi$ is the expectation over the distribution of the admissible trajectories $(s_0, a_0, s_1, \pi(s_1), \dots)$ obtained by executing the policy π starting from $s_0 = s$ and $a_0 = a$. Therefore, the quantity $Q^\pi(s,a)$ represents the γ-discounted and expected cumulative reward collected by executing the policy π starting from $s_0 = s$ and $a_0 = a$. Often, the concept of value function V^π is used and corresponds to:

$$\forall s \in S, \quad V^\pi(s) = Q^\pi(s, \pi(s)),$$

which represents the γ-discounted and expected cumulative reward collected by executing the policy π starting from $s_0 = s$ and $a_0 = \pi(s)$. So, the aim, when

optimising an MDP, is to find a policy π, called an optimal policy, such that:

$$\forall \pi' \in A^S, \forall s \in S, V^{\pi}(s) \geq V^{\pi'}(s).$$

To do so, an important tool is the optimal action-value function $Q^* \in \mathbb{R}^{S\times A}$ defined as $Q^* = \max_{\pi\in A^S} Q^{\pi}$. It has been shown by Puterman (1994) that a policy π is optimal if and only if $\forall s \in S, \quad V^{\pi}(s) = V^*(s)$, where $\forall s \in S, \quad V^*(s) = \max_{a\in A} Q^*(s,a)$. In addition, an important concept, that we will use throughout the paper, is greediness. A policy π is said greedy with respect to a function $Q \in \mathbb{R}^{S\times A}$ if:

$$\forall s \in S, \pi(s) \in \underset{a\in A}{\operatorname{argmax}}\, Q(s,a).$$

Greedy policies are important because a policy π greedy with respect to Q^* is optimal (Puterman, 1994). Thus, if we manage to find Q^*, we automatically found an optimal policy by taking a greedy policy with respect to Q^*. Moreover, Q^{π} and Q^* are known to be the unique fixed points of the contracting operators T^{π} and T^* (also called Bellman operators) respectively:

$$\forall Q \in \mathbb{R}^{S\times A}, \forall (s,a) \in S\times A, \quad T^{\pi}Q(s,a) = R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a)Q(s,\pi(s')),$$
$$\forall Q \in \mathbb{R}^{S\times A}, \forall (s,a) \in S\times A, \quad T^{*}Q(s,a) = R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a)\max_{b\in A} Q(s,b).$$

This means, by uniqueness of the fixed points Q^{π} and Q^*, that:

$$\begin{aligned} Q^{\pi} &= \underset{Q\in\mathbb{R}^{S\times A}}{\operatorname{argmin}} \|T^{\pi}Q - Q\|_{p,\mu} = \underset{Q\in\mathbb{R}^{S\times A}}{\operatorname{argmin}} \|T^{\pi}Q - Q\|_{\infty}, \\ Q^{*} &= \underset{Q\in\mathbb{R}^{S\times A}}{\operatorname{argmin}} \|T^{*}Q - Q\|_{p,\mu} = \underset{Q\in\mathbb{R}^{S\times A}}{\operatorname{argmin}} \|T^{*}Q - Q\|_{\infty}, \end{aligned} \tag{1}$$

where $\mu \in \Delta_{S\times A}$ is such that $\forall (s,a) \in S\times A, \mu(s,a) > 0$ and $p \geq 1$. Thus, Eq (2.1) shows that optimising an MDP can be seen as the minimisation of the criterion $J_{p,\mu}(Q) = \|T^*Q - Q\|_{p,\mu}$ where $T^*Q - Q$ is the OBR. Moreover, Piot et al (2014c) showed that the function $J_{p,\mu}$ is DC and they provided an explicit decomposition for $p = 1$ and $p = 2$. However, minimising directly $J_{p,\mu}$ via a DC programming technique, such as DC Algorithm (DCA), when the MDP is perfectly known is not useful as there exists DP techniques (VI, PI and LP) which efficiently and exactly compute optimal policies. They rely on nice properties of the Bellman operators such as being a contraction and monotonicity. However, when the state space becomes large, two important problems arise and DP programming is not an option anymore. The first one, called the *representation* problem, relate to the fact that each value $Q^{\pi}(s,a)$ of the action-value functions cannot be stored as the number of values grows too much. So these functions need to be approximated with a moderate number of coefficients. The second problem, called the *sampling* problem, arises because only samples from the Bellman operators are observed (there is thus only a partial knowledge of the dynamics P and the reward function R). One solution for the representation problem is to use a linear approximation of the action-value functions thanks to a basis of $d \in \mathbb{N}^*$ functions $\phi = (\phi_i)_{i=1}^d$ where $\phi_i \in \mathbb{R}^{S\times A}$. In addition, we define for each state-action couple (s,a) the vector

$\phi(s,a) \in \mathbb{R}^d$ such that $\phi(s,a) = (\phi_i(s,a))_{i=1}^d$. Thus, the action-value functions are characterised by a vector $\theta \in \mathbb{R}^d$ and noted Q_θ:

$$\forall \theta \in \mathbb{R}^d, \forall (s,a) \in S \times A, Q_\theta(s,a) = \sum_{i=1}^{d} \theta_i \phi_i(s,a) = \langle \theta, \phi(s,a) \rangle,$$

where $\langle .,. \rangle$ is the canonical dot product of $\mathbb{R}^d$. In order to tackle the *sampling* problem, RL techniques have been proposed and rely principally on Approximate DP (ADP) methods. One notable exception, developed by Piot et al (2014c), consists in directly minimising an empirical norm of the OBR via DCA (see Sec. 2.2).

2.2 Reinforcement Learning

RL is a vast domain with different settings (Sutton and Barto, 1998; Lange et al, 2012) sharing the property that the model (the dynamics P and the reward R) of the MDP is only known through data (*sampling* problem). Data can be collected on-line by an agent acting in the MDP, via a simulator or via previous interactions. Here, we focus on the last setting called the batch setting. More precisely, the machine is provided with a set of traces of interactions with the MDP:

$$D_{RL} = (s_j, a_j, r_j, s'_j)_{j=1}^{N_{RL}},$$

where $s_j \in S$, $a_j \in A$, $r_j = R(s_j, a_j)$, $s'_j \sim P(.|s_j, a_j)$ and $N_{RL} \in \mathbb{N}^*$. Using only that set, a batch RL technique must estimate an optimal policy. Several techniques inspired by ADP such as Fitted-Q (Ernst et al, 2005) (Approximate VI technique), Least squares Policy Iteration (LSPI) (Lagoudakis and Parr, 2003) (Approximate PI technique) and Locally Smoothed Regularised Approximate Linear Programming (LSRALP)- (Taylor and Parr, 2012) exist. They rely on particular properties of the optimal Bellman operator T^*, such as monotonicity and contraction, to estimate the fixed-point Q^*. Here, we are interested in a new breed of RL techniques consisting in directly minimising the empirical OBR:

$$\begin{aligned} J_{RL}(Q) &= \frac{1}{N_{RL}} \sum_{j=1}^{N_{RL}} |T^*Q(s_j,a_j) - Q(s_j,a_j)|, \\ &= \frac{1}{N_{RL}} \sum_{j=1}^{N_{RL}} |R(s_j,a_j) + \gamma \sum_{s' \in S} P(s'|s_j,a_j) \max_{a \in A} Q(s',a) - Q(s_j,a_j)|. \quad (2) \end{aligned}$$

Piot et al (2014c) showed that minimising directly the empirical optimal Bellman residual (J_{RL}) is a legit technique as:

$$\|Q^* - Q^\pi\|_{p,\nu} \leq \frac{C}{1-\gamma} J_{RL}(Q),$$

where C is a constant depending on the dynamics P, $\nu \in \Delta_{S \times A}$ and π is a greedy policy with respect to Q. This bound shows that minimising J_{RL}, leads to learn a function close to the optimal quality function Q^*. A finite-sample analysis version of the bound and a comparison to ADP techniques is also provided by Piot et al

(2014c). In order to minimise J_{RL}, they showed that this criterion is DC, gave an explicit decomposition and proposed to use DCA.

In practice, $T^*Q(s_j, a_j)$ cannot be computed from D_{RL}, as P is unknown, unless the MDP is deterministic. Indeed, in that case, which is the one considered in our experiments (see Sec. 4), we have:

$$\sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} Q(s', a) = \max_{a \in A} Q(s'_j, a).$$

So, $T^*Q(s_j, a_j) = R(s_j, a_j) + \gamma \max_{a \in A} Q(s'_j, a)$ can be obtained from D_{RL}. In the general case, only a unbiased estimate of $T^*Q(S_i, A_i)$ can be computed via:

$$\hat{T}^*Q(s_i, a_i) = R(s_i, a_i) + \gamma \max_{b \in A} Q(s'_i, b).$$

The problem is that $|\hat{T}^*Q(s_j, a_j) - Q(s_j, a_j)|^p$ is a biased estimator of $|T^*Q(s_j, a_j) - Q(s_j, a_j)|^p$ and the bias is uncontrolled (Antos et al, 2008). In order to alleviate this typical problem, several better estimators $|\hat{T}^*Q(s_j, a_j) - Q(s_j, a_j)|^p$ of $|T^*Q(s_j, a_j) - Q(s_j, a_j)|^p$ have been proposed, such as embeddings in Reproducing Kernel Hilbert Spaces (RKHS)(Lever et al, 2012) or locally weighted averager such as Nadaraya-Watson estimators(Taylor and Parr, 2012). In both cases, the unbiased estimate of $T^*Q(s_j, a_j)$ takes the form:

$$\hat{T}^*Q(s_j, a_j) = R(s_j, a_j) + \gamma \frac{1}{N_{RL}} \sum_{j=k}^{N_{RL}} \beta_j(s'_k) \max_{a \in A} Q(s'_k, a),$$

where $\beta_j(s'_k)$ is the weight of samples s'_k in the $T^*Q(s_j, a_j)$ estimate.

2.3 LfD and RCAL

LfD consists in learning an policy π_E from demonstrations of an expert (which can be optimal or near-optimal) and without observing rewards. The aim is to generalise the expert behaviour in states that where not observed in the demonstration set. This setting is easily motivated as, in a lot of practical applications, it is easier to provide expert demonstrations than a reward function. Here, we consider the batch LfD setting where a set of expert demonstrations

$$D_E = (s_i, a_i)_{i=1}^{N_E},$$

with $s_i \in S$, $a_i = \pi_E(s_i)$ and $N_E \in \mathbb{N}^*$, and a set of sampled transitions without rewards

$$D_{NE} = (s_j, a_j, s'_j)_{j=1}^{N_{NE}},$$

with $s_j \in S$, $a_j \in A$, $s'_j \sim P(.|s_j, a_j)$ and $N_{NE} \in \mathbb{N}^*$, are provided. The set D_{NE} gives a useful information on the dynamics of the underlying MDP and the set D_E gives examples of an optimal (or sub-optimal) behaviour. There are several ways to tackle the LfD problem. The most well-known and studied are Inverse Reinforcement Learning (IRL) (Ng et al, 2000; Ziebart et al, 2008; Syed and Schapire, 2008; Klein et al, 2012, 2013) and Imitation Learning (IL) (Ratliff et al, 2007; Syed and Schapire, 2010; Ross and Bagnell, 2010; Ross et al, 2011;

Judah et al, 2012). IRL consists in estimating a reward function that explains the expert behaviour. Once the reward is estimated, the resulting MDP has to be solved to end up with an actual policy. The interested reader can refer to this survey (Neu and Szepesvári, 2009). On the other hand. IL consists in directly learning a mapping from states to actions to imitates the expert. This approach can be cast to a pure Supervised Learning (SL) problem such as Multi-Class Classification (MCC) (Pomerleau, 1989; Ratliff et al, 2007; Ross and Bagnell, 2010; Syed and Schapire, 2010). Indeed, to compare to the standard classification notations, a state-action couple $(s_i, a_i = \pi_E(s_i))$ of the expert set D_E could be seen as an input-label (x_i, y_i) couple of a training set $D = (x_i \in X, y_i \in Y)_{i=1}^N$, where X is a compact set of inputs (in our particular case X is even finite) and Y a finite set of labels. The goal of MCC is, given D, to find a decision rule $g \in \mathfrak{H} \subset Y^X$, where $\mathfrak{H}$ is an hypothesis space, that generalises the relation between inputs and labels by minimising the empirical risk:

$$g = \operatorname*{argmin}_{h \in \mathfrak{H}} \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{y_i = h(x_i)\}}.$$

Properties of g and how well it generalises the data are notably studied by Vapnik (1998). However, minimising directly the empirical risk is unrealistic and practitioners use convex surrogates. Often, another approach, called score-based MCC, where a score function $Q \in \mathbb{R}^{X \times Y}$ is learnt, is used (we intentionally decide to use the same notation as action-value functions as there is a close link between between score functions and action-value functions (Klein et al, 2013)). The score $Q(x, y)$ represents the correspondence between the input x and the label y. The higher the score is, the more likely y will be chosen when x is the input. Thus, the decision rule g corresponding to the score Q is $g(x) = \operatorname{argmax}_{y \in Y} Q(x, y)$. For instance, Ratliff et al (2007) use a large margin approach which is a score-based MCC for solving an IL problem. The large margin approach consists, given the training set D, in minimising the following criterion:

$$J(Q) = \frac{1}{N} \sum_{i=1}^{N} \max_{y \in Y} \{Q(x_i, y) + l(x_i, y_i, y)\} - Q(x_i, y_i),$$

where $l \in \mathbb{R}_+^{X \times Y \times Y}$ is called the margin function. If this function is zero, minimising $J(Q)$ attempts to find a score function for which the example labels are scored higher than all other labels. Choosing a non-zero margin function improves generalisation (Ratliff et al, 2007) and instead of requiring only that the example label is scored higher than all other labels, it requires it to be better than each label y by an amount given by the margin function. In practice, one can use a margin function equals 0 for the inputs $(x_i, y_i, y = y_i)$ and 1 otherwise (this is the margin function we chose in our experiments). Applying the large margin approach to the LfD problem gives the following minimisation criterion:

$$J_E(Q) = \frac{1}{N_E} \sum_{i=1}^{N_E} \max_{a \in A} \{Q(s_i, a) + l(s_i, \pi_E(s_i), a)\} - Q(s_i, \pi_E(s_i)).$$

However this approach does not take into account the underlying dynamics of the MDP represented in the set D_{NE}.

To avoid that drawback, Piot et al (2014a) propose to see the score function Q as an optimal quality function Q^* of an MDP. To do so, they rely on the one-to-one relation between optimal quality functions and rewards functions. Indeed, for each function $Q \in \mathbb{R}^{S\times A}$, there exists a reward function $R_Q \in \mathbb{R}^{S\times A}$ such that $Q = Q^*$ where Q^* is the optimal quality function with respect to the reward R_Q. Moreover, there is an explicit formula for R_Q depending only on Q and P (Piot et al, 2014a):

$$R_Q(s,a) = Q(s,a) - \gamma \sum_{s'\in S} P(s'|s,a) \max_{b\in A} Q(s',b).$$

Knowing that, Piot et al (2014a) propose to regularise the criterion J_E by a term controlling the sparsity of the reward associated to the score function. This regularisation term is:

$$\begin{aligned} J_{NE}(Q) &= \frac{1}{N_{NE}} \sum_{j=1}^{N_{NE}} |R_Q(s_j,a_j)|, \\ &= \frac{1}{N_{NE}} \sum_{j=1}^{N_{NE}} |\gamma \sum_{s'\in S} P(s'|s_j,a_j) \max_{a\in A} Q(s',a) - Q(s_j,a_j)|. \end{aligned} \tag{3}$$

This helps to reduce the variance of the method as it considers as good candidates only Q functions with sparse rewards R_Q. The algorithm RCAL consists in minimising by a gradient descent the following criterion:

$$J_{RCAL}(Q) = J_E(Q) + \lambda_{RCAL} J_{NE}(Q).$$

However, it is easy to see that $J_{NE}(Q)$ (see Eq. (2.3)) corresponds to $J_{RL}(Q)$ (see Eq. (2.2)) when the reward function is null. Thus, $J_{NE}(Q)$ is also DC and as J_E is convex, then J_{RCAL} is DC. So, we propose to use the DCA to minimise J_{RCAL} in Sec. 3.

2.4 Reinforcement Learning with Expert Demonstrations

RLED aims at finding an optimal control in a MDP where some expert data are provided in addition to standard sampled transitions with rewards. Such a paradigm is also easily motivated as in a lot of practical applications a goal (reward function) is provided to an agent (a robot for instance) but it can be quite difficult or risky to optimise from scratch (huge or dangerous environment to explore). Also a good control is often difficult to find as the reward function is very sparse and the agent needs to explore a lot of possibilities before finding a reward and retro-propagate it. Thus, an expert (a human for instance) can provide some demonstrations in order to guide the agent through the good learning path and accelerate the learning process (Clouse, 1996; Gil et al, 2009; Knox and Stone, 2012; Taylor et al, 2011; Griffith et al, 2013). This combination of reward and expert data is somehow what we can experience in our daily life when we set goals to achieve (reward functions) and we observe other human beings (experts) achieving that same goals. Here, we consider the batch setting (Kim et al, 2013;

Piot et al, 2014b). More precisely, the apprentice agent is given a set of expert demonstrations (the same as the one in LfD)

$$D_E = (s_i, a_i)_{i=1}^{N_E},$$

where $s_i \in S$, $a_i = \pi_E(s_i)$ and $N_E \in \mathbb{N}^*$, and a set of sampled transitions (the same as the one in RL)

$$D_{RL} = (s_j, a_j, r_j, s'_j)_{j=1}^{N_{RL}},$$

where $s_j \in S$, $a_j \in A$, $r_j = R(s_j, a_j)$, $s'_j \sim P(.|s_j, a_j)$ and $N_{RL} \in \mathbb{N}^*$. Piot et al (2014b) propose the RLED algorithm, minimising the following criterion:

$$J_{RLED}(Q) = J_E(Q) + \lambda_{RLED} J_{RL}(Q),$$

combining two criteria: J_E and J_{RL} (defined in Eq. (2.2)). The regularisation factor λ_{RLED} weights the importance between the expert and RL data. If one has a high confidence on the quality of the RL data, one will set λ_{RLED} to high value and to a low value otherwise. The criterion J_{RLED} can also be seen as the minimisation of J_{RL} guided by constraints provided by the expert data (Piot et al, 2014b). Another explanation could be that RLED produces a score function Q that is forced to be an action-value function. This accelerates the optimisation of J_{RL} and improves the performance of the method. In the original paper (Piot et al, 2014b), the authors propose to minimise $J_{RLED}(Q)$ by a gradient descent. However, as J_{RL} is DC and J_E is convex, then J_{RLED} is DC. Thus, we propose to use DCA to minimise this criterion in Sec. 3. But before, we give some basics on DC programming which are sufficient to derive DC decompositions for RLED and RCAL.

2.5 Basics on DC and DC programming

DC programming addresses non-convex (but DC) and non-differentiable (but sub-differentiable) optimisation problems by transforming them into a series of intermediary convex (thus simpler) optimisations problems. It allows leveraging the knowledge on convex optimisation and for that reason as been adapted to different domains such as Machine Learning (Le Thi et al, 2008, 2014b,a). It also gives some guarantees when one of the function of the DC decomposition is polyhedral, such as convergence in finite time to local minima. Thus, it seems a better solution than a simple gradient descent when confronted to complex non-convex (but DC) optimisation problems.

Let E be a finite dimensional Hilbert space, $\langle ., . \rangle_E$ and $\|.\|_E$ its dot product and norm respectively. We say that a function $J \in \mathbb{R}^E$ is DC if there exists $f, g \in \mathbb{R}^E$ which are convex and lower semi-continuous such that $J = f - g$ (Tao and An, 2005). The set of DC functions is noted $DC(E)$ and is stable to most of the operations that can be encountered in optimisation, contrary to the set of convex functions. Indeed, let $(J_i)_{i=1}^K$ be a sequence of $K \in \mathbb{N}^*$ DC functions and $(\alpha_i)_{i=1}^K \in \mathbb{R}^K$ then $\sum_{i=1}^K \alpha_i J_i$, $\prod_{i=1}^K J_i$, $\min_{1 \leq i \leq K} J_i$, $\max_{1 \leq i \leq K} J_i$ and $|J_i|$ are DC functions (Hiriart-Urruty, 1985). In order to minimise a DC function $J = f - g$, we need to define a notion of differentiability for convex and lower semi-continuous

functions. Let g be such a function and $e \in E$, we define the sub-gradient $\partial_e g$ of g in e as:

$$\partial_e g = \{\delta_e \in E, \forall e' \in E, g(e') \geq g(e) + \langle e' - e, \delta_e \rangle_E\}.$$

For convenience, we make this little abuse of notations where $\partial_e g$ can refer to any element of $\partial_e g$. For a convex and lower semi-continuous $g \in \mathbb{R}^E$, the sub-gradient $\partial_e g$ is non empty for all $e \in E$ (Hiriart-Urruty, 1985). This observation leads to a minimisation method of a function $J \in DC(E)$ called Difference of Convex functions Algorithm (DCA). Indeed, as J is DC, we have:

$$\forall (e, e') \in E^2, J(e') = f(e') - g(e') \underset{(a)}{\leq} f(e') - g(e) - \langle e' - e, \partial_e g \rangle_E,$$

where inequality (a) is true by definition of the sub-gradient. Thus, for all $e \in E$, the function J is upper bounded by a function $I_e \in \mathbb{R}^E$ defined, $\forall e' \in E$, by

$$I_e(e') = f(e') - g(e) - \langle e' - e, \partial_e g \rangle_E.$$

The function I_e is a convex and lower semi-continuous function (as it is the sum of two convex and lower semi-continuous functions which are f and the linear function $\forall e' \in E, \langle e - e', \partial_e g \rangle_E - g(e)$). In addition, those functions have the particular property that $\forall e \in E, J(e) = I_e(e)$. The set of convex functions $(I_e)_{e \in E}$ that upper-bound the function J plays a key role in DCA.

The algorithm DCA (Tao and An, 2005) consists in constructing a sequence $(e_n)_{n \in \mathbb{N}}$ such that the sequence $(J(e_n))_{n \in \mathbb{N}}$ decreases. The first step is to choose a starting point $e_0 \in E$, then to minimise the convex function I_{e_0} that upper-bounds the function J. We can remark that minimising I_e is equivalent to minimising I'_e defined by $\forall e' \in E$

$$I'_e(e') = f(e') - \langle e', \partial_e g \rangle_E.$$

We note e_1 a minimiser of I_{e_0}, $e_1 \in \operatorname{argmin}_{e \in E} I_{e_0}$. This minimisation can be realised by any convex optimisation solver. As $J(e_0) = I_{e_0}(e_0) \geq I_{e_0}(e_1)$ and $I_{e_0}(e_1) \geq J(e_1)$, then $J(e_0) \geq J(e_1)$. Thus, if we construct the sequence $(e_k)_{k \in \mathbb{N}}$ such that $\forall k \in \mathbb{N}, e_{k+1} \in \operatorname{argmin}_{e \in E} I_{e_k}$ and $e_0 \in E$, then we obtain a decreasing sequence $(J(e_k))_{k \in \mathbb{N}}$. Therefore, the algorithm DCA solves a sequence of convex optimisation problems in order to solve a DC optimisation problem. Three important choices can radically change the DCA performance: the first one is the explicit choice of the decomposition of J, the second one is the choice of the starting point e_0 and finally the choice of the intermediate convex solver. The DCA algorithm hardly guarantees convergence to the global optima, but it usually provides good solutions. Moreover, it has some nice properties when one of the functions f or g is polyhedral. A function g is said polyhedral when $\forall e \in E, g(e) = \max_{1 \leq i \leq K} [\langle \alpha_i, e \rangle_H + \beta_i]$, where $(\alpha_i)_{i=1}^K \in E^K$ and $(\beta_i)_{i=1}^K \in \mathbb{R}^K$. If one of the function f, g is polyhedral, J is under bounded, the DCA sequence $(e_k)_{k \in \mathbb{N}}$ is bounded and the DCA algorithm converges in finite time to one of the local minima. The finite time aspect is important in terms of implementation. More details about DC programming and DCA are given by Tao and An (2005) and even conditions for convergence to the global optima.

To summarise, once a DC decomposition is found, minimising the DC criterion $J = f - g$ via DCA corresponds to minimise the following intermediary convex functions:

$$I'_k(e') = f(e') - \langle e', \partial_{e_k} g \rangle_E,$$

with $e_0 \in E$ and $e_{k+1} \in \operatorname{argmin}_{e' \in E} I'_k(e')$. In practice, DCA stops when $e_{k+1} = e_k$ or when $k = K$ (with K the maximal number of steps for DCA and this is the stopping criterion chosen in our experiments) and the output of the algorithm is e_k. In addition, obtaining e_{k+1} can be done by linear programming (if I'_k can be transformed into a linear program) or by gradient descent. In our experiments, we choose gradient descent to solve the intermediary convex problems with the following updates:

$$e'_0 = e_k, \quad \partial_{e'_p} I'_k = \partial_{e'_p} f - \partial_{e_k} g, \quad e'_{p+1} = e'_p - \alpha_p \frac{\partial_{e'_p} I'_k}{\|\partial_{e'_p} I'_k\|_E}, \tag{4}$$

where $(\alpha_p)_{p \in \mathbb{N}} \in \mathbb{R}_+^{\mathbb{N}}$. Finally, we set $e_{k+1} = e'_{p^*}$ where p^* meets a stopping criterion such as $p^* = N$ (with N is the maximal number of steps of the gradient descent and this is the stopping criterion chosen in our experiments) or $\partial_{e'_p} I'_k = 0$ for instance. Thus, when the intermediary convex problems I'_k are determined, it is necessary to be able to compute their gradient $\partial_{e'_p} I'_k$ in order to apply DCA. In the next section, we give the decompositions of the criteria J_{RCAL} and J_{RLED} and how to compute the different gradients.

3 DC Decompositions for RCAL and RLED

In this section, we derive a DC decomposition for the criteria J_{RLED} (Sec. 3.3) and J_{RCAL} (Sec. 3.2) from the DC decompositions of J_{RL} and J_{NE} (Sec. 3.1). Several decompositions are possible, we describe the one that we actually use in experiments. Here, the DC decompositions is realised as if we could compute $\gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} Q(s', a)$. In practice, in the deterministic case, we replace this quantity by $\gamma P(s'_j|s_j, a_j) \max_{a \in A} Q(s'_j, a)$ which is easily computable and in the general case by $\frac{1}{N} \sum_{k=1}^{N} \beta_j(s'_k) \max_{a \in A} Q(s'_k, a)$ which is obtained using RKHS embedding or Nadaraya-Watson estimators.

3.1 DC decomposition of J_{RL} and J_{NE}

Let us start with the criterion $J_{RL}(Q)$

$$\begin{aligned} J_{RL}(Q) &= \frac{1}{N_{RL}} \sum_{j=1}^{N_{RL}} |T^* Q(s_j, a_j) - Q(s_j, a_j)|, \\ &= \frac{1}{N_{RL}} \sum_{j=1}^{N_{RL}} |R(s_j, a_j) + \gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} Q(s', a) - Q(s_j, a_j)|. \end{aligned}$$

As we have seen previously, in RL, the functions Q are characterised by a vector $\theta \in \mathbb{R}^d$ and noted $Q_\theta(s, a) = \sum_{i=1}^{d} \theta_i \phi_i(s, a) = \langle \theta, \phi(s, a) \rangle$. Thus, we consider the criterion:

$$J_{RL}(\theta) = \frac{1}{N_{RL}} \sum_{j=1}^{N_{RL}} |R(s_j, a_j) + \gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} \langle \phi(s', a), \theta \rangle - \langle \phi(s_j, a_j), \theta \rangle|.$$

Noticing that $\gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} \langle \phi(s', a), \theta \rangle$ is convex in θ as a sum of a max of convex functions, that $\langle \phi(s_j, a_j), \theta \rangle$ is also convex in θ and that $|f - g| = 2\max(f, g) - (f + g)$, we have the following DC decomposition for J_{RL} (a complete proof is given by Piot et al (2014c)):

$$f^j_{RL}(\theta) = 2\max\left(R(s_j, a_j) + \gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} \langle \phi(s', a), \theta \rangle, \langle \phi(s_j, a_j), \theta \rangle\right),$$

$$g^j_{RL}(\theta) = R(s_j, a_j) + \gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} \langle \phi(s', a), \theta \rangle + \langle \phi(s_j, a_j), \theta \rangle,$$

$$f_{RL}(\theta) = \frac{1}{N_{RL}} \sum_{j=1}^{N_{RL}} f^j_{RL}(\theta), \quad g_{RL}(\theta) = \frac{1}{N_{RL}} \sum_{j=1}^{N_{RL}} g^j_{RL}(\theta),$$

$$J_{RL}(\theta) = \frac{1}{N_{RL}} \sum_{j=1}^{N_{RL}} f^j_{RL}(\theta) - g^j_{RL}\theta = f_{RL}(\theta) - g_{RL}(\theta).$$

We can do exactly the same calculus for J_{NE} as it is the same criterion than J_{RL} with a null reward. We have:

$$J_{NE}(Q) = \frac{1}{N_{NE}} \sum_{j=1}^{N_{NE}} |R_Q(s_j, a_j)|,$$

$$J_{NE}(Q) = \frac{1}{N_{NE}} \sum_{j=1}^{N_{NE}} |\gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} Q(s', a) - Q(s_j, a_j)|.$$

With the linear parametrisation, we obtain:

$$J_{NE}(\theta) = \frac{1}{N_{RL}} \sum_{i=j}^{N_{RL}} |\gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} \langle \phi(s', a), \theta \rangle - \langle \phi(s_j, a_j), \theta \rangle|,$$

$$f^j_{NE}(\theta) = 2\max\left(\gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} \langle \phi(s', a), \theta \rangle, \langle \phi(s_j, a_j), \theta \rangle\right),$$

$$g^j_{NE}(\theta) = \gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} \langle \phi(s', a), \theta \rangle + \langle \phi(s_j, a_j), \theta \rangle,$$

$$f_{NE}(\theta) = \frac{1}{N_{NE}} \sum_{j=1}^{N_{NE}} f^j_{NE}(\theta), \quad g_{NE}(\theta) = \frac{1}{N_{NE}} \sum_{j=1}^{N_{NE}} g^j_{NE}(\theta),$$

$$J_{NE}(\theta) = \frac{1}{N_{NE}} \sum_{j=1}^{N_{NE}} f^j_{NE}(\theta) - g^j_{NE}(\theta) = f_{NE}(\theta) - g_{NE}(\theta).$$

Now that we have the DC decompositions, it is sufficient to calculate the intermediary convex problems ($I^k_{RL}(\theta)$ and $I^k_{NE}(\theta)$). To do so, we need the gradients $\partial_{\theta'} g_{RL}$ and $\partial_{\theta'} g_{NE}$:

$$\partial_{\theta'} g^j_{RL} = \gamma \sum_{s' \in S} P(s'|s_j, a_j) \phi(s', a^*_{\theta', s'}) + \phi(s_j, a_j), \quad \partial_{\theta'} g_{RL} = \frac{1}{N_{RL}} \sum_{j=1}^{N_{RL}} \partial_{\theta'} g^j_{RL},$$

$$\partial_{\theta'} g^j_{NE} = \gamma \sum_{s' \in S} P(s'|s_j, a_j)\phi(s', a^*_{\theta', s'}) + \phi(s_j, a_j), \quad \partial_{\theta'} g_{NE} = \frac{1}{N_{NE}} \sum_{j=1}^{N_{NE}} \partial_{\theta'} g^j_{NE}.$$

where $a^*_{\theta',s'} = \text{argmax}_{a \in A} \langle \phi(s', a), \theta' \rangle$. So the intermediary convex problems are:

$$I^k_{RL}(\theta) = f_{RL}(\theta) - \langle \partial_{\theta^k_{RL}} g_{RL}, \theta \rangle, \quad I^k_{NE}(\theta) = f_{NE}(\theta) - \langle \partial_{\theta^k_{NE}} g_{NE}, \theta \rangle,$$

with θ^0_{RL}, θ^0_{RL} in $\mathbb{R}^d$, $\theta^{k+1}_{RL} = \text{argmin}_{\theta \in \mathbb{R}^d} I^k_{RL}(\theta)$ and $\theta^{k+1}_{NE} = \text{argmin}_{\theta \in \mathbb{R}^d} I^k_{NE}(\theta)$. To minimise $I^k_{RL}(\theta)$ and $I^k_{NE}(\theta)$, we have to compute their gradients and do the update as in Eq .(2.5):

$$\partial_{\theta'} f^j_{RL} = \begin{cases} \gamma \sum_{s' \in S} P(s'|s_j, a_j)\phi(s', a^*_{\theta', s'}) \\ \text{ if } R(s_j, a_j) + \gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} \langle \phi(s', a), \theta' \rangle > \langle \phi(s_j, a_j), \theta' \rangle, \\ \phi(s_j, a_j) \text{ else} \end{cases}$$

$$\partial_{\theta'} f_{RL} = \frac{1}{N_{RL}} \sum_{j=1}^{N_{RL}} \partial_{\theta'} f^j_{RL}, \quad \partial_{\theta'} I^k_{RL} = \partial_{\theta'} f_{RL}(\theta) - \partial_{\theta^k_{RL}} g_{RL}.$$

$$\partial_{\theta'} f^j_{NE} = \begin{cases} \gamma \sum_{s' \in S} P(s'|s_j, a_j)\phi(s', a^*_{\theta', s'}) \\ \text{ if } \gamma \sum_{s' \in S} P(s'|s_j, a_j) \max_{a \in A} \langle \phi(s', a), \theta' \rangle > \langle \phi(s_j, a_j), \theta' \rangle, \\ \phi(s_j, a_j) \text{ else }, \end{cases}$$

$$\partial_{\theta'} f_{NE} = \frac{1}{N_{NE}} \sum_{j=1}^{N_{NE}} \partial_{\theta'} f^j_{NE}, \quad \partial_{\theta'} I^k_{NE} = \partial_{\theta'} f_{NE}(\theta) - \partial_{\theta^k_{RL}} g_{NE}.$$

The DC decompositions of J_{RCAL} and J_{RLED} follows directly from the ones of J_{RL} and J_{NE}. In addition, one can easily notice that f_{RL}, g_{RL}, f_{NE} and g_{NE} are polyhedral and that property will be directly transmitted to the decompositions of J_{RCAL} and J_{RLED}.

3.2 DC decomposition of J_{RCAL}

The criterion J_{RCAL} is composed of two criterion J_E and J_{NE}:

$$J_E(Q) = \frac{1}{N_E} \sum_{i=1}^{N_E} \max_{a \in A} [Q(s_i, a) + l(s_i, a_i, a)] - Q(s_i, a_i),$$
$$J_{RCAL}(Q) = J_E(Q) + \lambda_{RCAL} J_{NE}(Q).$$

With a linear parametrisation, we have:

$$J_E(\theta) = \frac{1}{N_E} \sum_{i=1}^{N_E} \max_{a \in A} [\langle \phi(s_i, a), \theta \rangle + l(s_i, a_i, a)] - \langle \phi(s_i, a_i), \theta \rangle,$$
$$J_{RCAL}(\theta) = J_E(\theta) + \lambda_{RCAL} J_{NE}(\theta).$$

Thus, the DC decomposition is quite trivial to obtain as J_E is convex:

$$\begin{aligned}
f_{RCAL}(\theta) &= J_E(\theta) + \lambda_{RCAL} f_{NE}(\theta),\\
g_{RCAL}(\theta) &= \lambda_{RCAL} g_{NE}(\theta),\\
J_{RCAL}(\theta) &= f_{RCAL}(\theta) - g_{RCAL}(\theta).
\end{aligned}$$

To obtain the intermediary convex problems, we need to calculate the gradient of g_{RCAL}:

$$\begin{aligned}
\partial_{\theta'} g_{RCAL} &= \lambda_{RCAL} \partial_{\theta'} g_{NE},\\
&= \frac{\lambda_{RCAL}}{N_{NE}} \sum_{j=1}^{N_{NE}} \partial_{\theta'} g_{NE}^j.
\end{aligned}$$

Thus, the convex intermediary problems have the following form:

$$I_{RCAL}^k(\theta) = f_{RCAL}(\theta) - \langle \partial_{\theta_{RCAL}^k} g_{RCAL}, \theta \rangle.$$

To minimise I_{RCAL}^k, we calculate its gradient:

$$\begin{aligned}
\partial_{\theta'} J_E &= \frac{1}{N_E} \sum_{i=1}^{N_E} \phi(s_i, a_{i,\theta'}^*) - \phi(s_i, a_i),\\
\partial_{\theta'} f_{RCAL} &= \partial_{\theta'} J_E + \lambda_{RCAL} \partial_{\theta'} f_{NE},\\
\partial_{\theta'} I_{RCAL}^k &= \partial_{\theta'} f_{RCAL} - \partial_{\theta_{RCAL}^k} g_{RCAL}.
\end{aligned}$$

where $a_{i,\theta'}^* = \text{argmax}_{a \in A}[\langle \phi(s_i, a), \theta' \rangle + l(s_i, a_i, a)]$.

3.3 DC decomposition of J_{RLED}

The decomposition of J_{RLED} is quite similar as the one of J_{RCAL}. We present it briefly for sake of completeness. J_{RLED} is composed by two terms:

$$J_{RLED}(Q) = J_E(Q) + \lambda_{RLED} J_{RL}(Q).$$

With a linear parametrisation, we have:

$$J_{RLED}(\theta) = J_E(\theta) + \lambda_{RLED} J_{RL}(\theta).$$

As J_E is convex, a DC decomposition is quite trivial to obtain:

$$\begin{aligned}
f_{RLED}(\theta) &= J_E(\theta) + \lambda_{RLED} f_{RL}(\theta),\\
g_{RLED}(\theta) &= \lambda_{RLED} g_{RL}(\theta),\\
J_{RLED}(\theta) &= f_{RLED}(\theta) - g_{RLED}(\theta).
\end{aligned}$$

Now, to minimise J_{RLED}, we calculate the intermediary convex problems by obtaining the gradient of g_{RLED}:

$$\begin{aligned}
\partial_{\theta'} g_{RLED} &= \lambda_{RLED} \partial_{\theta'} g_{RL},\\
&= \frac{\lambda_{RLED}}{N_{RL}} \sum_{j=1}^{N_{RL}} \partial_{\theta'} g_{RL}^j.
\end{aligned}$$

Thus, the intermediary convex problems have the following form:

$$I^k_{RLED}(\theta) = f_{RLED}(\theta) - \langle \partial_{\theta^k_{RLED}} g_{RLED}, \theta \rangle.$$

Finally, in order to minimise I^k_{RLED} by gradient descent, we need to calculate $\partial_{\theta'} I^k_{RLED}$:

$$\partial_{\theta'} f_{RLED} = \partial_{\theta'} J_E + \lambda_{RLED} \partial_{\theta'} f_{RL},$$
$$\partial_{\theta'} I^k_{RLED} = \partial_{\theta'} f_{RLED} - \partial_{\theta^k_{RLED}} g_{RLED}.$$

Now that we have the DC decompositions of J_{RCAL} and J_{RLED}, we can compare the performance of those algorithms when the minimisation is realised via direct gradient descent or via DCA. This comparison is realised, in Sec. 4, on an abstract but representative class of MDPs called Garnets.

4 Experiments

This section is composed of three experiments which aim at showing that using DC programming instead of gradient descent slightly improve the performance of existing algorithms, namely RCAL and RLED. The first experiment consists in showing the performance improvement of the RCAL algorithm when the set D_{NE} is fixed and the set D_E is growing on different MDPs which are randomly generated and called Garnets. The RCAL algorithm which consists in the minimisation of the criterion J_{RCAL} is done by two methods. The first one is by gradient descent and is called RCAL in the remaining. The second one is by DCA and is called RCALDC. We also compare RCAL to a classical classification algorithm (Ratliff et al, 2007), called Classif and which corresponds to the minimisation of J_{RCAL} by gradient descent when $\lambda_{RCAL} = 0$. The second experiments focuses on the performance improvement of RLED when the set D_{RL} is fixed and the set D_E is growing. We compare the algorithm RLED when the minimisation is done by gradient descent, called RLED, and when the minimisation is done by DCA, called RLEDDC. Those algorithms are compared to LSPI which uses only the set D_{RL} as input and Classif which uses only the set D_E as input. Finally, the last experiment focuses on the performance improvement of RLED when the set D_E is fixed and the set D_{RL} is growing. But first, to realise those experiments, we need to introduce the notion of Garnets, explain how we construct the sets D_E, D_{NE} and D_{RL} and give the value of the different parameters of DCA and the gradient descent algorithms.

Garnets (Archibald et al, 1995) are an abstract class of finite MDPs and easy to build. Here, we consider a special case of Garnets specified by three parameters: (N_S, N_A, N_B). Parameters N_S and N_A are respectively the number of states and actions. Thus, $S = (s_i)_{i=1}^{N_S}$ and $A = (a_i)_{i=1}^{N_A}$ are, respectively, the state and action spaces. The parameter N_B ($N_B \leq N_S$), called the branching factor, defines for each state-action couple (s, a) the number of next states. Here, we consider deterministic MDPs with $N_B = 1$. The next state for each (s, a), noted $s'_{s,a}$, is drawn uniformly from the set of states. In addition the discount factor is set to $\gamma = 0.9$ or $\gamma = 0.99$. Finally, we need to define the reward function R. To do so, we draw uniformly and without replacement $\lceil N_S/10 \rfloor$ (where $\lceil x \rfloor$ represents the nearest integer form x) states from S. Then, for those states, the reward $R(s)$ is

drawn randomly and uniformly in $[0, 1]$ and for the other states $R(s) = 0$. Thus, we obtain a sparse reward function depending only on the states which is the kind of rewards encountered in practice. As we choose finite MDPs, a canonical choice of features ϕ is the tabular basis $\phi : S \to \mathbb{R}^{N_S}$ where $\phi(s) \in \mathbb{R}^{N_S}$ is a vector which is null excepted in s where it is equal to 1.

A Garnet is a finite MDP where the dynamics P and the reward R is perfectly known. Thus, an optimal policy (playing the role of the expert) can be easily computed by DP (PI in our case). This policy is a key element to build the expert set D_E that fed RCAL and RLED. In our experiments D_E has the following form:

$$D_E = (\omega_j)_{\{1 \leq j \leq L_E\}},$$

where $\omega_j = (s_{i,j}, a_{i,j})_{\{1 \leq i \leq H_E\}}$ is a trajectory obtained by starting from a random state $s_{1,j}$ (chosen uniformly in S) and applying the expert policy (π_E) H_E times such that $a_{i,j} = \pi_E(s_{i,j})$ and $s_{i+1,j} = s'_{s_{i,j},a_{i,j}}$. So, D_E is composed by L_E trajectories of π_E of length H_E and we have $L_E H_E = N_E$. In addition the data set D_{RL} has the following form:

$$D_{RL} = (\tau_j)_{\{1 \leq j \leq L_{RL}\}},$$

where $\tau_j = (s_{i,j}, a_{i,j}, r_{i,j}, s'_{i,j} = s'_{s_{i,j},a_{i,j}})_{\{1 \leq i \leq H_{RL}\}}$ is a trajectory obtained by starting from a random state $s_{1,j}$ (chosen uniformly) and applying the the random policy ($a_{i,j}$ is chosen uniformly from A) H_{RL} times such that $s'_{i,j} = s_{i+1,j}$ and $r_{i,j} = R(s_{i,j})$. So, D_{RL} is composed by L_{RL} trajectories of π_R of length H_{RL} and we have $L_{RL} H_{RL} = N_{RL}$. The set D_{NE} corresponds to the set D_{RL} where the reward $r_{i,j}$ is dropped:

$$D_{NE} = (\tau_j)_{\{1 \leq j \leq L_{NE}\}},$$

where $\tau_j = (s_{i,j}, a_{i,j}, s'_{i,j} = s'_{s_{i,j},a_{i,j}})_{\{1 \leq i \leq H_{NE}\}}$ is a trajectory obtained by starting from a random state $s_{1,j}$ (chosen uniformly) and applying the the random policy H_{RL} times such that $s'_{i,j} = s_{i+1,j}$.

Finally, as we want to compare gradient descent to DCA for the minimisation of the criteria J_{RLED} and J_{RCAL}, it is important to give the parameters of those methods. First, the two methods start form the same starting point which is specified for each experiment. The updates for the gradient descent have the same form as in Eq. (2.5). And the number of updates is 100. To make DCA comparable to gradient descent, we set the number of intermediary convex problems K to 10 and the number of updates N for the gradient descent of the intermediary problems to 10. Thus, we have a total of $KN = 100$ updates for DCA. In addition, for each gradient descent (it can be the global gradient descent or the one used in the intermediary problems), we set the coefficients $\forall p \in \mathbb{N}^*, \alpha_p = 1$.

4.1 RCAL experiment

Our first experiment shows the performance improvement of RCAL when $\gamma = 0.9$, $\lambda_{RCAL} = 0.1$, H_E is increasing and (L_E, H_{NE}, L_{NE}) are fixed. It aims at showing that the algorithms perform better when more expert information is available and a fixed amount of knowledge of the dynamics known through D_{NE} is given. It consists in generating 10 Garnets with $(N_S = 100, N_A = 5, N_B = 1)$ which gives us the set of Garnet problems $\mathfrak{G} = (G_p)_{p=1}^{10}$. On each problem p of the set $\mathfrak{G}$, we

compute an optimal and expert policy, π_E^p. The parameter L_E takes its values in the set $(L_E^k)_{k=1}^{10} = (2, 4, 6, .., 20)$ and $H_E = 5$, $H_{NE} = 5$, $L_{NE} = 20$. Then, for each set of parameters $(L_E^k, H_E, L_{NE}, H_{NE})$ and each G_p, we compute 20 expert policy sets $(D_E^{i,p,k})_{i=1}^{20}$ and 20 random policy sets $(D_{NE}^{i,p,k})_{i=1}^{20}$ which fed the algorithms RCAL and Classif. Overall, we test RCAL on 2000 sets of data. The starting point of the algorithm RCAL (gradient descent and DCA) and Classif is the null function.

The criterion of performance chosen, for the algorithm A and for each couple $(D_E^{i,p,k}, D_{NE}^{i,p,k})$, is the following:

$$T_A^{i,p,k} = \frac{\mathbb{E}_\rho[V^{\pi_E^p} - V^{\pi_A^{i,p,k}}]}{\mathbb{E}_\rho[V^{\pi_E^p}]},$$

where π_E^p is the expert policy, $\pi_A^{i,p,k}$ is the policy induced by the algorithm A fed by the couple $(D_E^{i,p,k}, D_{NE}^{i,p,k})$ and ρ is the uniform distribution over the state space S. For RCAL and Classif, we have $\pi_A^{i,p,k}(s) \in \text{argmax}_{a \in A} Q_{\theta^*}(s, a)$ where θ^* is the output of those algorithms. This criterion of performance is the normalised absolute difference of value-functions between the expert policy and the one induced by the algorithm. Thus, the lesser this criterion is the better. The mean criterion of performance T_A^k for each set of parameters $(L_E^k, H_E, L_{NE}, H_{NE})$ is:

$$T_A^k = \frac{1}{200} \sum_{p=1}^{10} \sum_{i=1}^{20} T^{i,p,k}.$$

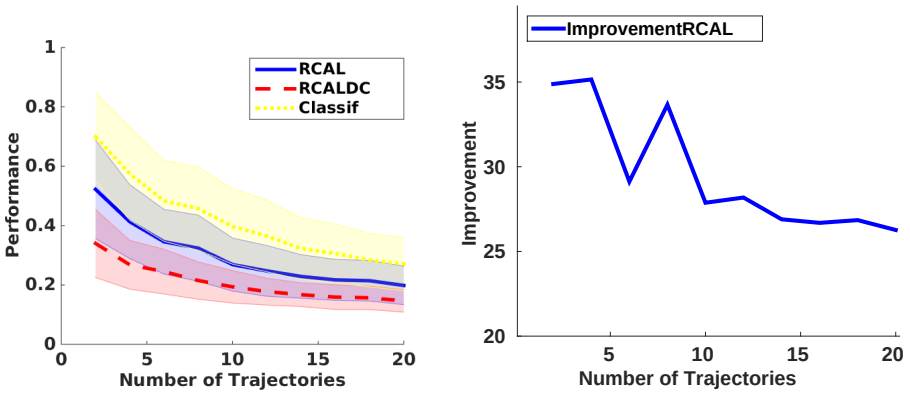
For each algorithm A, we plot $(L_E^k, T_A^k)_{k=1}^{10}$ in Fig. 1(a). The colored shadows on the figures represent the variance of the algorithms. In order to verify that RCALDC has a better performance than RCAL, we calculate the improvement which is the following ratio:

$$Imp^k = 100 \frac{T_{RCAL}^k - T_{RCALDC}^k}{T_{RCAL}^k}.$$

This ratio represents in percentage how much RCALDC is better than RCAL. In Fig. 1(b), we plot $(L_E^k, Imp^k)_{k=1}^{10}$. In Fig. 1(a) and Fig. 1(b), we clearly observe that RCALDC performs in average better than RCAL. In addition, we also compute the number of times that $T_{RCALDC}^{i,p,k}$ is lesser than $T_{RCAL}^{i,p,k}$ over the 2000 runs of this experiment and we obtain 1732. Thus, RCALDC is $100 \times \frac{1732}{2000} = 86.6\%$ of the time better than RCAL. Those different elements tend to prove that using DC programming for RCAL improves clearly the performance.

4.2 RLED experiments

The second experiment is quite similar to the first except that we use the RLED algorithm. Here $\gamma = 0.99$, $\lambda_{RLED} = 0.1$, L_E is increasing and (H_E, H_{RL}, L_{RL}) are fixed. Like the first experiment, it aims at showing that RLED performs better when more expert information is available. It consists in generating a set of Garnets $\mathfrak{G} = (G_p)_{p=1}^{10}$. The parameter L_E takes its values in the set $(L_E^k)_{k=1}^{10} = (1, 2, 3, .., 10)$ and $H_E = 5$, $H_{RL} = 5$, $L_{RL} = 100$. Then, for each set of parameters



(a) Performance. (b) Improvement.

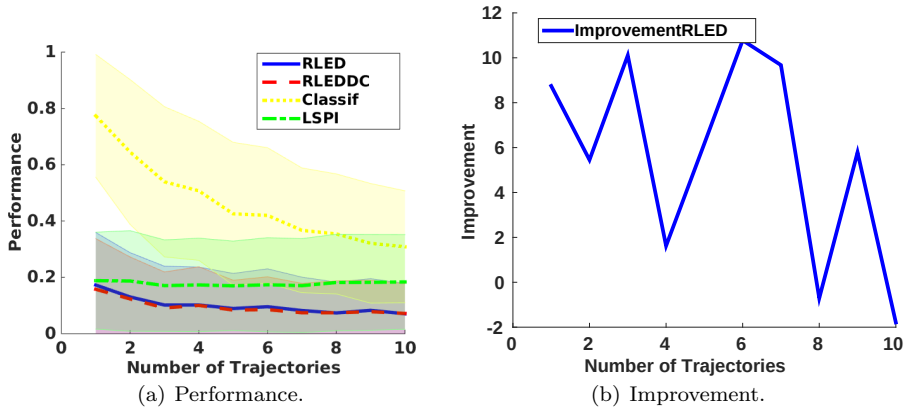
Fig. 1 Garnet Experiment for RCAL.

$(L_E^k, H_E, L_{RL}, H_{RL})$ and each G_p, we compute 20 expert policy sets $(D_E^{i,p,k})_{i=1}^{20}$ and 20 random policy sets $(D_{RL}^{i,p,k})_{i=1}^{20}$ which fed the algorithms RLED, Classif and LSPI. Here, the starting point of RLED is the ouput of the LSPI algorithm.

The criterion of performance for the algorithm A and for each couple $(D_E^{i,p,k}, D_{RL}^{i,p,k})$ is $T_A^{i,p,k}$ which has the same definition as in the first experiment. The mean criterion of performance for each set of parameters $(L_E^k, H_E, L_{RL}, H_{RL})$ is T_A^k. For each algorithm A, we plot $(L_E^k, T_A^k)_{k=1}^{10}$ in Fig. 2(a). In order to verify that RLEDDC has a better performance than RLED, we calculate the improvement which is the following ratio:

$$Imp^k = 100 \frac{T_{RLED}^k - T_{RLEDDC}^k}{T_{RLED}^k}.$$

This ratio represents in percentage how much RLEDDC is better than RLED. In Fig 2(b), we plot $(L_E^k, Imp^k)_{k=1}^{10}$. In Fig. 2(a), we can not distinguish which

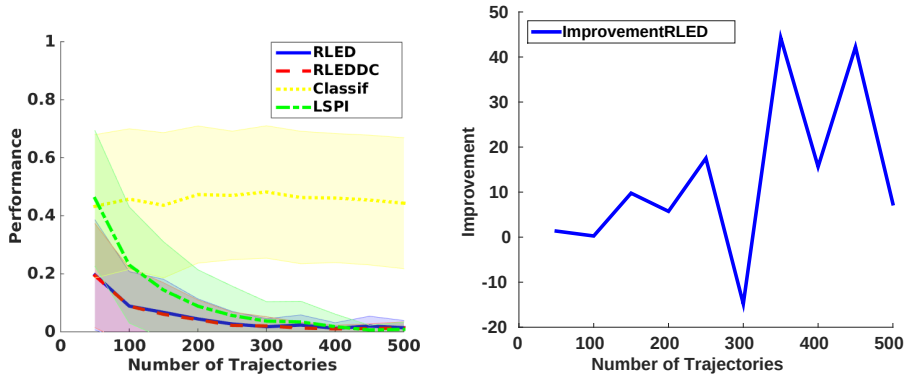


(a) Performance. (b) Improvement.

Fig. 2 Garnet Experiment for RLED with L_E increasing.

algorithm between RLED (with gradient descent) and RLEDDC is better. Thus, we plot a zoom of this curve without the variance to have a better view in Fig. 4(a). Even though, it seems that RLEDDC is slightly better, there is no clear difference between the two algorithms. This is principally due to the fact that RLED performs already well without DCA.

Finally, the third experiment aims at showing that RLED performs better when the information on the model (dynamics and reward) is getting bigger. Here, $\gamma = 0.99$ and $\lambda_{RLED} = 1$. λ_{RLED} is set to a higher value as the set D_{RL} is getting bigger in this experiment and more weight needs to be put on the J_{RL} criterion. The experiment consists in generating a set of Garnets $\mathfrak{G} = (G_p)_{p=1}^{10}$. The parameter L_{RL} takes its values in the set $(L_{RL}^k)_{k=1}^{10} = (50, 100, 150, .., 500)$ and $L_E = 5$, $H_E = 5$, $H_{RL} = 5$. Then, for each set of parameters $(L_E, H_E, L_{RL}^k, H_{RL})$ and each G_p, we compute 20 expert policy sets $(D_E^{i,p,k})_{i=1}^{20}$ and 20 random policy sets $(D_{RL}^{i,p,k})_{i=1}^{20}$ which fed the algorithms RLED, Classif and LSPI. Here, the starting point of RLED is the ouput of the LSPI algorithm. Like the previous experiments, for each algorithm A, we plot the mean performance $(L_{RL}^k, T_A^k)_{k=1}^{10}$ in Fig. 4(a) and the improvement $(L_{RL}^k, Imp^k)_{k=1}^{10}$ in Fig. 4(b). In Fig. 2(b), we cannot distinguish



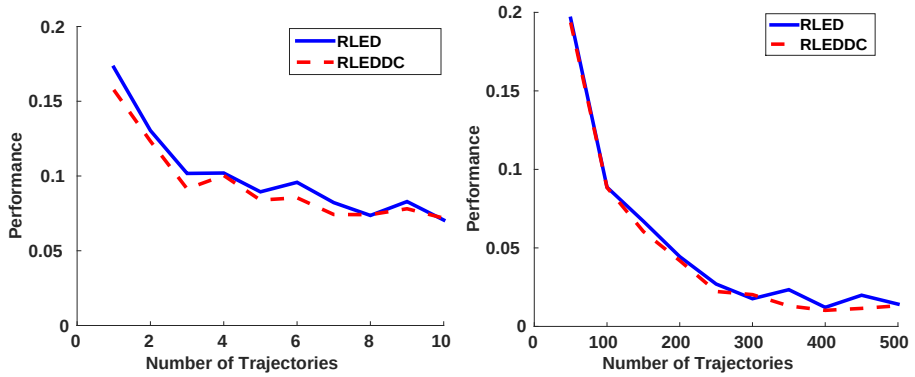
(a) Performance. (b) Improvement.

Fig. 3 Garnet Experiment for RLED with L_{RL} increasing.

between RLED and RLEDDC. A zoom of this plot is proposed in Fig .4(b) where there is a slight advantage for RLEDDC. Thus for RLED and contrary to RCAL, even tough there is a slight improvement using DCA, we can not conclude that there is an advantage to use DC programming.

5 Conclusion and Perspectives

In this paper, we showed the implications of seeing the Optimal Bellman Residual (OBR) as a Difference of Convex (DC) functions in the fields of Reinforcement Learning with Expert Demonstrations (RLED) and Learning from Demonstrations (LfD). More precisely, we gave one of the possible DC decompositions of two



(a) Zoom of RLED performance with L_E increasing

(b) Zoom of RLED performance with L_{RL} increasing

Fig. 4 Zoom for RLED experiments

algorithms, namely RCAL and RLED. In addition, we compared in generic experiments, using randomly constructed Markov Decision Processes (MDPs) called Garnets, the performances of RCAL and RLED using DC programming versus a classical gradient descent. In order to make a fair comparison between the methods, we imposed the same number of updates and the same starting point. Experiments showed a clear advantage of using DC Algorithm (DCA) for the minimisation of the RCAL criterion. However, there was only a slight advantage of DCA for the RLED criterion which can be explained by the fact that RLED with gradient descent already performs well, hence it is quite difficult to improve the method. In conclusion, it seems a promising perspective to use DC programming in fields such as RL and LfD where the goal is to minimise a norm of the OBR which is a DC function. As perspectives, we would like to test several DC decompositions and to start using non-parametric gradient descent in order to solve the intermediary convex problems. Indeed, in RL and more generally in Machine Learning (ML), the choice of features $\phi(s,a) = (\phi_i(s,a))_{i=1}^d$ that represent our hypothesis space is often problem-dependent and need a human expertise. To avoid that step and to automatise even more the algorithm, we want to use non-parametric gradient descent (Grubb and Bagnell, 2011) which automatically learn its own features. Finally, we would like to use those techniques on large scale and real-life applications to prove their ability to scale-up.

References

Antos A, Szepesvári C, Munos R (2008) Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. Machine Learning

Archibald T, McKinnon K, Thomas L (1995) On the generation of Markov decision processes. Journal of the Operational Research Society

Argall B, Chernova S, Veloso M, Browning B (2009) A survey of robot learning from demonstration. Robotics and autonomous systems 57(5):469–483

Atkeson C, Schaal S (1997) Robot learning from demonstration. In: Proc. of ICML
Bellman R, Kalaba R (1965) Dynamic programming and modern control theory. Academic Press New York
Bertsekas D (1995) Dynamic programming and optimal control, vol 1. Athena Scientific, Belmont, MA
Clouse JA (1996) An introspection approach to querying a trainer
Ernst D, Geurts P, Wehenkel L (2005) Tree-based batch mode reinforcement learning. In: Journal of Machine Learning Research
Gil A, Stern H, Edan Y (2009) A cognitive robot collaborative reinforcement learning algorithm. World Academy of Science, Engineering and Technology
Griffith S, Subramanian K, Scholz J, Isbell C, Thomaz AL (2013) Policy shaping: Integrating human feedback with reinforcement learning. In: Advances in Neural Information Processing Systems, pp 2625–2633
Grubb A, Bagnell J (2011) Generalized boosting algorithms for convex optimization. In: Proc. of ICML
Hiriart-Urruty J (1985) Generalized differentiability, duality and optimization for problems dealing with differences of convex functions. In: Convexity and duality in optimization, Springer
Howard RA (1960) Dynamic programming and markov processes
Judah K, Fern A, Dietterich T (2012) Active imitation learning via reduction to iid active learning. In: Proc. of UAI
Kim B, Farahmand A, Pineau J, Precup D (2013) Learning from limited demonstrations. In: Proc. of NIPS
Klein E, Geist M, Piot B, Pietquin O (2012) Inverse reinforcement learning through structured classification. In: Proc. of NIPS
Klein E, Piot B, Geist M, Pietquin O (2013) A cascaded supervised learning approach to inverse reinforcement learning. In: Proc. of ECML, Springer
Knox WB, Stone P (2012) Reinforcement learning from simultaneous human and mdp reward. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, pp 475–482
Lagoudakis M, Parr R (2003) Least-squares policy iteration. Journal of Machine Learning Research
Lange S, Gabel T, Riedmiller M (2012) Batch reinforcement learning. In: Reinforcement Learning, Springer, pp 45–73
Le HM, Le Thi HA, Nguyen MC (2015) Sparse semi-supervised support vector machines by dc programming and dca. Neurocomputing 153:62–76
Le Thi HA, Le HM, Dinh TP, et al (2008) A dc programming approach for feature selection in support vector machines learning. Advances in Data Analysis and Classification 2(3):259–278
Le Thi HA, Le HM, Dinh TP (2014a) Feature selection in machine learning: an exact penalty approach using a difference of convex function algorithm. Machine Learning pp 1–24
Le Thi HA, Nguyen MC, Dinh TP (2014b) A dc programming approach for finding communities in networks. Neural computation
Lever G, Baldassarre L, Gretton A, Pontil M, Grünewälder S (2012) Modelling transition dynamics in MDPs with RKHS embeddings. In: Proc. of ICML
Neu G, Szepesvári C (2009) Training parsers by inverse reinforcement learning. Machine learning 77(2)

Ng A, Russell S, et al (2000) Algorithms for inverse reinforcement learning. In: Proc. of ICML

Piot B, Geist M, Pietquin O (2014a) Boosted and Reward-regularized Classification for Apprenticeship Learning. In: Proc. of AAMAS

Piot B, Geist M, Pietquin O (2014b) Boosted Bellman residual minimization handling expert demonstrations. In: Proc. of ECML, Springer

Piot B, Geist M, Pietquin O (2014c) Difference of Convex Functions Programming for Reinforcement Learning. In: Proc. of NIPS

Pomerleau D (1989) Alvinn: An autonomous land vehicle in a neural network. Tech. rep., DTIC Document

Puterman M (1994) Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons

Ratliff N, Bagnell J, Srinivasa S (2007) Imitation learning for locomotion and manipulation. In: Proc. of IEEE-RAS International Conference on Humanoid Robots

Ross S, Bagnell J (2010) Efficient reductions for imitation learning. In: Proc. of AISTATS

Ross S, Gordon G, Bagnell J (2011) A reduction of imitation learning and structured prediction to no-regret online learning. In: Proc. of AISTATS

Schaal S (1997) Learning from demonstration. In: Proc. of NIPS, pp 1040–1046

Sutton RS, Barto AG (1998) Reinforcement learning: An introduction. Cambridge Univ Press

Syed U, Schapire R (2008) A game-theoretic approach to apprenticeship learning. In: Proc. of NIPS

Syed U, Schapire R (2010) A reduction from apprenticeship learning to classification. In: Proc. of NIPS

Tao P, An L (1997) Convex analysis approach to dc programming: Theory, algorithms and applications. Acta Mathematica Vietnamica 22:289–355

Tao P, An L (2005) The dc (difference of convex functions) programming and dca revisited with dc models of real world nonconvex optimization problems. Annals of Operations Research 133:23–46

Taylor G, Parr R (2012) Value function approximation in noisy environments using locally smoothed regularized approximate linear programs. In: Proc. of UAI

Taylor ME, Suay HB, Chernova S (2011) Integrating reinforcement learning with human demonstrations of varying ability. In: The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, International Foundation for Autonomous Agents and Multiagent Systems, pp 617–624

Vapnik V (1998) Statistical learning theory. Wiley

Ziebart BD, Maas AL, Bagnell JA, Dey AK (2008) Maximum entropy inverse reinforcement learning. In: AAAI, pp 1433–1438