Advanced machine learning techniques based on DC programming and DCA

Vinh Thanh Ho

THÈSE

en vue de l'obtention du titre de

DOCTEUR DE L'UNIVERSITÉ DE LORRAINE

(arrêté ministériel du 7 Août 2006)

Spécialité INFORMATIQUE

présentée par

HO VINH THANH

Titre de la thèse :

TECHNIQUES AVANCÉES D'APPRENTISSAGE AUTOMATIQUE BASÉES SUR LA PROGRAMMATION DC ET DCA

—

ADVANCED MACHINE LEARNING TECHNIQUES BASED ON DC PROGRAMMING AND DCA

soutenue le 08 décembre 2017

Composition du Jury :

Rapporteurs	Akiko TAKEDA	*Professeur, The Institute of Statistical Mathematics*
	Emilio CARRIZOSA	*Professeur, University of Seville*
Examinateurs	Tao PHAM DINH	*Professeur, INSA de Rouen*
	Jérôme DARMONT	*Professeur, Université Lumière Lyon 2*
	Yann GUERMEUR	*Directeur de Recherche, LORIA*
Invité	Jean-Michel VANPEPERSTRAETE	*Expert Scientifique, Naval groupe*
Directrice de thèse	Hoai An LE THI	*Professeur, Université de Lorraine*
Co-encadrant	Ahmed ZIDNA	*MCF, Université de Lorraine*

THÈSE PRÉPARÉE AU SEIN DE LABORATOIRE
D'INFORMATIQUE THÉORIQUE ET APPLIQUÉE (LITA)
UNIVERSITÉ DE LORRAINE, METZ, FRANCE

Remerciements

Je souhaite ici adresser un grand merci à toutes les personnes qui ont rendu cette thèse possible. Cette thèse a été réalisée au sein du Laboratoire d'Informatique Théorique et Appliquée (LITA) de l'Université de Lorraine.

Avant tout, je souhaite exprimer ma profonde gratitude à ma directrice de thèse, Madame Hoai An LE THI, Professeur des Universités à l'Université de Lorraine, pour m'avoir accordé la grande opportunité de démarrer ma carrière scientifique au sein du LITA, tout en m'offrant les conditions nécessaires au bon déroulement de cette thèse durant ces années. Sous sa direction j'ai reçu un soutien permanent, des encouragements considérables et pu explorer de nouveaux axes de recherche grâce à ses immenses connaissances dans le domaine de l'optimisation et ses applications. Madame LE THI a bien voulu encadrer mes travaux, corriger mes articles et cette thèse ; elle m'a aidé à surmonter les difficultés avec beaucoup de patience, d'enthousiasme et de générosité. J'ai eu beaucoup de chance de pouvoir bénéficier de ses compétences pédagogiques et de sa grande expérience pour envisager devenir un bon enseignant chercheur scientifique dans le futur. Je la remercie très sincèrement pour tous ses conseils qui sont des atouts importants et utiles dans mon travail de recherche scientifique ainsi que dans ma vie personnelle. Je lui suis extrêmement reconnaissant pour tout le temps qu'elle m'a accordé.

Je souhaite remercier mon co-encadrant, Monsieur Ahmed ZIDNA, Maître de Conférences à l'Université de Lorraine, pour son attention et pour m'avoir encouragé durant ces années des études.

Je souhaite exprimer ma sincère gratitude à Monsieur Tao PHAM DINH, Professeur des Universités à l'INSA de Rouen pour ses conseils pertinents et son suivi à long terme au cours de mes travaux de recherche. Je voudrais le remercier pour nos échanges enrichissants, pour m'avoir suggéré de nouvelles voies de recherche et fait l'honneur de siéger au jury de ma thèse.

Je souhaite remercier Madame Akiko TAKEDA, Professeur à l'Institut de mathématiques statistiques, et Monsieur Emilio CARRIZOSA, Professeur à l'Université de Seville, pour avoir accepté d'être rapporteurs de ma thèse et pour le temps précieux consacré.

Je souhaite remercier également Monsieur Jérôme DARMONT, Professeur à l'Université Lumière Lyon 2, Monsieur Yann GUERMEUR, Directeur de Recherche au

LORIA, et Monsieur Jean-Michel VANPEPERSTRAETE, Expert scientifique dans le Naval groupe, d'avoir bien voulu accepter d'être membres du jury de cette thèse.

Je remercie mes collègues du LITA et mes amis de Metz : Manh Cuong, Minh Thuy, Bich Thuy, Anh Vu, Tran Thuy, Minh Tam, Hoai Minh, Duy Nhat, Phuong Anh, Xuan Thanh, Tran Bach, Viet Anh, Dinh Chien ... pour leur soutien, et leurs encouragements, ainsi que pour les agréables moments passés ensemble lors de mon séjour en France. Je voudrais remercier particulièrement Docteur Xuan Thanh VO pour son aide et son grand soutien durant ces années. Je remercie également Madame Annie HETET, Secrétaire du LITA, pour sa grande disponibilité et son aide très spontanée.

Enfin et surtout, je souhaite exprimer ma grande gratitude, mon amour et mon affection à ma famille qui m'a encouragé, aimé et a beaucoup pensé à moi tout au long de ce travail de recherche.

HO Vinh Thanh

Né le 02 mars, 1991 (Viet Nam)

Tél: 07 82 67 88 96

E-mail: vinh-thanh.ho@univ-lorraine.fr

Adresse personnelle: 1 rue Châtillon, 57000 Metz, France

Adresse professionnelle: Bureau UM-AN1-027, LITA – Université de Lorraine,
3 rue Augustin Fresnel, BP 45112, 57073 Metz, France

Situation Actuelle

Depuis octobre 2014	Doctorant au Laboratoire d'Informatique Théorique et Appliquée (LITA EA 3097) de l'Université de Lorraine. Encadré par Prof. Hoai An Le Thi et MCF. Ahmed Zidna Sujet de thèse : **"Techniques avancées d'apprentissage automatique basées sur la programmation DC et DCA"**

Experience Professionnelle

04/2014–06/2014	Stagiaire au laboratoire LITA, UFR MIM, Université de Lorraine, Metz, France. Responsable de stage: Prof. Hoai An Le Thi. Mémoire: **On solving Markov decision processes in reinforcement learning by DC programming and DCA**.

Diplôme et Formation

2014 au present	Doctorant en Informatique. LITA, UFR MIM, Université de Lorraine, Metz, France.
2013–2014	Master 2 en Mathématique, Université de Tours, France.
2009–2013	Diplôme universitaire en Mathématique et Informatique, Ecole Normale Supérieure de Ho Chi Minh Ville, Viet Nam.

Publications

Refereed international journal papers

[1] Tao Pham Dinh, Vinh Thanh Ho, Hoai An Le Thi. DC programming and DCA for Brugnano-Casulli Piecewise Linear Systems. Computers & Operations Research, 87: 196–204 (2017).

[2] Hoai An Le Thi, Vinh Thanh Ho, Tao Pham Dinh. A unified DC Programming Framework and Efficient DCA based Approaches for Large Scale Batch Reinforcement Learning. *Submitted* to the Journal of Global Optimization.

[3] Vinh Thanh Ho, Hoai An Le Thi. Online DCA for Reinforcement Learning. *Submitted*.

[4] Hoai An Le Thi, Vinh Thanh Ho. Online Learning based on Online DCA and Applications to Online Binary Classification. *In preparation*.

[5] Hoai An Le Thi, Vinh Thanh Ho. Online DCA for Prediction with Expert Advice. *Submitted*.

Refereed papers in books / Refereed international conference papers

[1] Vinh Thanh Ho, Zied Hajej, Hoai An Le Thi, Nidhal Rezg. Solving the Production and Maintenance Optimization Problem by a Global Approach. In: Le Thi et al. (eds) Modelling, Computation and Optimization in Information Systems and Management Sciences. MCO 2015. Advances in Intelligent Systems and Computing, vol 360, pp. 307–318, Springer, 2015.

[2] Vinh Thanh Ho, Hoai An Le Thi, Dinh Chien Bui. Online DC Optimization for Online Binary Linear Classification. In: Nguyen et al. (eds) Intelligent Information and Database Systems. ACIIDS 2016. Lecture Notes in Computer Science, vol 9622, pp. 661–670, Springer, 2016.

[3] Vinh Thanh Ho, Hoai An Le Thi. Solving an Infinite-Horizon Discounted Markov Decision Process by DC Programming and DCA. In: Nguyen et al. (eds) Advanced Computational Methods for Knowledge Engineering. ICCSAMA 2016. Advances in Intelligent Systems and Computing, vol 453, pp. 43-55, Springer, 2016.

[4] Vinh Thanh Ho, Hoai An Le Thi, Ahmed Zidna. A DCA Approach for the Stochastic Shortest Path Problem in Vehicle Routing. *Accepted* by IESM 2017: 7th International Conference on Industrial Engineering and Systems Management.

[5] Vinh Thanh Ho, Hoai An Le Thi. Online DCA for Reinforcement Learning. *Submitted* to ALT 2018: 29th International Conference on Algorithmic Learning Theory.

Communications in national / International conferences

[1] Vinh Thanh Ho, Hoai An Le Thi, Tao Pham Dinh. DC Programming and DCA for Brugnano-Casulli Piecewise Linear Systems. Presentation in the 27th European Conference on Operational Research, Glasgow, UK, July 12 - 15, 2015.

Contents

List of Figures

List of Tables

Abbreviations and Notations

Throughout the dissertation, we use uppercase letters to denote matrices, and lowercase letters for vectors or scalars. Vectors are also regarded as matrices with one column. Some of the abbreviations and notations used in the dissertation are summarized as follows.

DC	Difference of Convex functions
DCA	DC Algorithm
OBLC	Online Binary Linear Classification
RL	Reinforcement Learning
MDP	Markov Decision Process
DP	Dynamic Programming
Batch RL	Batch Reinforcement Learning
OBR	Optimal Bellman Residual
SSP	Stochastic Shortest Path

$\mathbb{R}$	set of real numbers
$\mathbb{R}^n$	set of real column vectors of size n
$\overline{\mathbb{R}}$	set of extended real numbers, $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$
$\|\cdot\|_p$	ℓ_p-norm ($0 < p < \infty$), $\|x\|_p = (\sum_{i=1}^n \|x_i\|^p)^{1/p}$, $x \in \mathbb{R}^n$
$\|\cdot\|$	Euclidean norm (or ℓ_2-norm), $\|x\| = (\sum_{i=1}^n \|x_i\|^2)^{1/2}$, $x \in \mathbb{R}^n$
$\|\cdot\|_\infty$	ℓ_∞-norm, $\|x\|_\infty = \max_{i=1,\ldots,n} \|x_i\|$, $x \in \mathbb{R}^n$
$\langle\cdot,\cdot\rangle$	scalar product, $\langle x, y\rangle = \sum_{i=1}^n x_i.y_i$, $x, y \in \mathbb{R}^n$
$\chi_C(\cdot)$	indicator function of a set C, $\chi_C(x) = 0$ if $x \in C$, $+\infty$ otherwise
$\text{co}\{C\}$	convex hull of a set of points C
$\text{Proj}_C(x)$	projection of a vector x onto a set C
$\text{dom}\, f$	effective domain of a function f
$\nabla f(x)$	gradient of a function f at x
$\partial f(x)$	subdifferential of a function f at x
$\mathbb{P}(X)$	probability of a random variable X

Résumé

Dans cette thèse, nous développons certaines techniques avancées d'apprentissage automatique dans le cadre de l'apprentissage en ligne et de l'apprentissage par renforcement ("reinforcement learning" en anglais – RL). L'épine dorsale de nos approches est la programmation DC (Difference of Convex functions) et DCA (DC Algorithm), et leur version en ligne, qui sont reconnues comme de outils puissants d'optimisation non convexe, non différentiable.

Cette thèse se compose de deux parties : la première partie étudie certaines techniques d'apprentissage automatique en mode en ligne et la deuxième partie concerne le RL en mode batch et mode en ligne. La première partie comprend deux chapitres correspondant à la classification en ligne (chapitre 2) et la prédiction avec des conseils d'experts (chapitre 3). Ces deux chapitres mentionnent une approche unifiée d'approximation DC pour différents problèmes d'optimisation en ligne dont les fonctions objectives sont des fonctions de perte 0-1. Nous étudions comment développer des algorithmes DCA en ligne efficaces en termes d'aspects théoriques et computationnels.

La deuxième partie se compose de quatre chapitres (chapitres 4, 5, 6, 7). Après une brève introduction du RL et ses travaux connexes au chapitre 4, le chapitre 5 vise à fournir des techniques efficaces du RL en mode batch basées sur la programmation DC et DCA. Nous considérons quatre différentes formulations d'optimisation DC en RL pour lesquelles des algorithmes correspondants basés sur DCA sont développés. Nous traitons les problèmes clés de DCA et montrons l'efficacité de ces algorithmes au moyen de diverses expériences. En poursuivant cette étude, au chapitre 6, nous développons les techniques du RL basées sur DCA en mode en ligne et proposons leurs versions alternatives. Comme application, nous abordons le problème du plus court chemin stochastique ("stochastic shortest path" en anglais – SSP) au chapitre 7. Nous étudions une classe particulière de problèmes de SSP qui peut être reformulée comme une formulation de minimisation de cardinalité et une formulation du RL. La première formulation implique la norme zéro et les variables binaires. Nous proposons un algorithme basé sur DCA en exploitant une approche d'approximation DC de la norme zéro et une technique de pénalité exacte pour les variables binaires. Pour la deuxième formulation, nous utilisons un algorithme batch RL basé sur DCA. Tous les algorithmes proposés sont testés sur des réseaux routiers artificiels.

Abstract

In this dissertation, we develop some advanced machine learning techniques in the framework of online learning and reinforcement learning (RL). The backbones of our approaches are DC (Difference of Convex functions) programming and DCA (DC Algorithm), and their online version that are best known as powerful nonsmooth, nonconvex optimization tools.

This dissertation is composed of two parts: the first part studies some online machine learning techniques and the second part concerns RL in both batch and online modes. The first part includes two chapters corresponding to online classification (Chapter 2) and prediction with expert advice (Chapter 3). These two chapters mention a unified DC approximation approach to different online learning algorithms where the observed objective functions are 0-1 loss functions. We thoroughly study how to develop efficient online DCA algorithms in terms of theoretical and computational aspects.

The second part consists of four chapters (Chapters 4, 5, 6, 7). After a brief introduction of RL and its related works in Chapter 4, Chapter 5 aims to provide effective RL techniques in batch mode based on DC programming and DCA. In particular, we first consider four different DC optimization formulations for which corresponding attractive DCA-based algorithms are developed, then carefully address the key issues of DCA, and finally, show the computational efficiency of these algorithms through various experiments. Continuing this study, in Chapter 6 we develop DCA-based RL techniques in online mode and propose their alternating versions. As an application, we tackle the stochastic shortest path (SSP) problem in Chapter 7. Especially, a particular class of SSP problems can be reformulated in two directions as a cardinality minimization formulation and an RL formulation. Firstly, the cardinality formulation involves the zero-norm in objective and the binary variables. We propose a DCA-based algorithm by exploiting a DC approximation approach for the zero-norm and an exact penalty technique for the binary variables. Secondly, we make use of the aforementioned DCA-based batch RL algorithm. All proposed algorithms are tested on some artificial road networks.

Introduction générale

Cadre général et motivations

Au cours de ces dernières années, l'explosion quantitative des données a obligé les chercheurs à avoir de nouvelles visions pour analyser et explorer des données, ce qui est généralement mentionné comme un sujet de Big data. Dans ce contexte, l'inconvénient de l'utilisation d'approches classiques a été mis en évidence. Ainsi, il est nécessaire de recourir aux techniques avancées adaptées au Big data. À la question de "quelles sont les tendances clé des techniques dans Big data?", le développement des techniques innovantes d'apprentissage automatique ("machine learning" en anglais) est une réponse.

Dans cette thèse, nous nous concentrons sur deux challenges en apprentissage automatique dans le contexte du Big data: l'apprentissage automatique pour une grande quantité de données en mode en ligne et en mode batch. En fonction de la disponibilité des données, les techniques d'apprentissage automatique peuvent être considérées en mode batch ou en mode en ligne. En particulier, pour le mode batch, les techniques génèrent des modèles en apprenant sur l'ensemble des données d'apprentissage en une fois. L'apprentissage automatique en mode en ligne (appelé apprentissage en ligne, "online learning" en anglais) met à jour le modèle au fur et à mesure en fonction des nouvelles données. Pour une nouvelle donnée d'entrée à chaque itération, l'apprentissage en ligne effectue une prédiction en utilisant le modèle actuel, puis vérifie la qualité de sa prédiction qui est utilisée pour mettre à jour le modèle pour l'itération suivante. L'apprentissage en ligne joue un rôle important dans des multiples contextes : quand les données sont disponibles progressivement, ou les prédictions doivent être données en temps réel, ou l'apprenant doit s'adapter dynamiquement aux nouveaux types de données, ou il est irréalisable d'apprendre sur l'ensemble des données. Jusqu'alors, la conception d'algorithmes efficaces d'apprentissage en ligne a été influencée par l'optimisation convexe en ligne. Cependant, dans la plupart d'applications, la fonction de perte utilisée pour évaluer les prédictions ou le domaine des prédictions est non convexe. Le désavantage des approches d'optimisation convexe en ligne a été indiqué dans la littérature. Ainsi, il est indispensable de recourir à l'optimisation non convexe en ligne pour développer des algorithmes d'apprentissage en ligne efficaces. La difficulté de cette fonction de perte peut être surmontée en utilisant son approximation DC (Difference of Convex functions). Le problème résultant est encore difficile en raison de sa non convexité. Mais il peut être surmonté par des techniques

basées sur la version en ligne de l'optimisation DC. Comme une contribution de cette thèse, nous développons les versions "en ligne" de DCA (DC Algorithm) standard pour les méthodes d'apprentissage (classification, prédiction) en ligne. La convergence des méthodes en ligne a été rigoureusement étudiée.

Parmi les principales catégories de techniques d'apprentissage automatique, nous nous intéressons à l'apprentissage par renforcement ("reinforcement learning" en anglais) en mode en ligne et en mode batch. L'apprentissage par renforcement est concerné par une classe de problèmes dans lesquels un agent doit apprendre un comportement décisionnel basé sur la rétroaction de récompense par les interactions avec un environnement dynamique. Les problèmes d'optimisation considérés en apprentissage par renforcement deviennent difficiles parce que leurs fonctions objectives sont des fonctions non convexes et non différentiables, plus précisément elles sont DC. Cependant, en exploitant les propriétés intéressantes de ces problèmes, ils peuvent être résolus efficacement par des techniques en ligne/batch basées sur DCA.

Sur le plan algorithmique, la thèse a proposé une approche unifiée, fondée sur la programmation DC et DCA, et leurs versions en ligne, des outils puissants d'optimisation non convexe qui connaissent un grand succès, au cours de trois dernières décennies, dans la modélisation et la résolution de nombreux problèmes d'application dans divers domaines de sciences appliquées, en particulier en apprentissage automatique et fouille de données ("data mining" en anglais) (voir par example [28, 54, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 71, 73, 74, 75, 79, 94, 95, 96, 97, 130] et la liste des références dans [56]). De nombreuses expérimentations numériques sur différents types de données dans divers domaines (biologie, transport, physique ...) réalisées dans cette thèse ont prouvé l'efficacité, la scalabilité, la rapidité des algorithmes proposés et leur supériorité par rapport aux méthodes standards.

La programmation DC et DCA considèrent le problème DC de la forme

$$\alpha = \inf\{f(x) := g(x) - h(x) \ : \ x \in \mathbb{R}^n\} \quad (P_{dc}),$$

où g et h sont des fonctions convexes définies sur $\mathbb{R}^n$ et à valeurs dans $\mathbb{R} \cup \{+\infty\}$, semi-continues inférieurement et propres. La fonction f est appelée fonction DC avec les composantes DC g et h, et $g-h$ est une décomposition DC de f. DCA est basé sur la dualité DC et des conditions d'optimalité locale. La construction de DCA implique les composantes DC g et h et non la fonction DC f elle-même. Chaque fonction DC admet une infinité des décompositions DC qui influencent considérablement sur la qualité (la rapidité, l'efficacité, la globalité de la solution obtenue ...) de DCA. Ainsi, au point de vue algorithmique, la recherche d'une "bonne" décomposition DC et d'un "bon" point initial est très importante dans le développement de DCA pour la résolution d'un programme DC.

L'utilisation de la programmation DC et DCA dans cette thèse est justifiée par de multiple arguments [97]:

— La programmation DC et DCA fournissent un cadre très riche pour les problèmes d'apprentissage automatique et fouille de données: l'apprentissage

automatique et fouille de données constituent *une mine des programmes DC* dont la résolution appropriée devrait recourir à la programmation DC et DCA. En effet la liste indicative (non exhaustive) des références dans [56] témoigne de la vitalité la puissance et la percée de cette approche dans la communauté d'apprentissage automatique et fouille de données.

— DCA est une philosophie plutôt qu'un algorithme. Pour chaque problème, nous pouvons concevoir une famille d'algorithmes basés sur DCA. La flexibilité de DCA sur le choix des décomposition DC peut offrir des schémas DCA plus performants que des méthodes standard.

— L'analyse convexe fournit des outils puissants pour prouver la convergence de DCA dans un cadre général. Ainsi tous les algorithmes basés sur DCA bénéficient (au moins) des propriétés de convergence générales du schéma DCA générique qui ont été démontrées.

— DCA est une méthode efficace, rapide et scalable pour la programmation non convexe. A notre connaissance, DCA est l'un des rares algorithmes de la programmation non convexe, non différentiable qui peut résoudre des programmes DC de très grande dimension. La programmation DC et DCA ont été appliqués avec succès pour la modélisation DC et la résolution de nombreux et divers problèmes d'optimisation non convexes dans différents domaines des sciences appliquées, en particulier en apprentissage automatique et fouille de données (voir la liste des références dans [56]).

Il est important de noter qu'avec les techniques de reformulation en programmation DC et les décompositions DC appropriées, on peut retrouver la plupart des algorithmes existants en programmation convexe/non convexe comme cas particuliers de DCA. En particulier, pour la communauté d'apprentissage automatique et fouille de données, les méthodes très connus comme Expectation–Maximisation (EM) [30], Succesive Linear Approximation (SLA) [16], ConCave–Convex Procedure (CCCP) [131], Iterative Shrinkage–Thresholding Algorithms (ISTA) [25] sont des versions spéciales de DCA.

Nos contributions

Les principales contributions de la thèse consistent à développer les techniques avancées d'apprentissage automatique en utilisant l'approche d'optimisation. Nous étudions une approche basée sur la programmation DC et DCA, et leurs versions en ligne pour résoudre les problèmes d'apprentissage en ligne et, en particulier, d'apprentissage par renforcement. Tout au long de la thèse, les questions clés de DCA ont été étudiées : quelle est la bonne décomposition DC, quelle est la méthode de solution efficace pour les sous-problèmes convexes dans le schéma DCA et quel est le bon point initial. Aborder ces questions dépend fortement des structures spéciales de chaque problème d'optimisation considéré, ce qui est exploité dans la plupart de nos travaux dans les chapitres 2, 3, 5, 6. De plus, les chapitres 2 et 3 incluent les contributions à l'étude d'une version en ligne de DCA en termes d'aspects théoriques et computationnels. Les principales réalisations de la thèse sont décrites en détail comme suit.

Tout d'abord, nous nous concentrons dans le chapitre 2 sur le développement de techniques d'apprentissage en ligne. L'apprentissage en ligne peut être décrit comme le processus consistant à prévoir une séquence d'échantillons basée sur la connaissance de la correction aux échantillons précédents et autres informations disponibles. La qualité des prédictions est évaluée par une fonction de perte, qui est souvent non convexe et/ou non différentiable. Dans nos travaux, nous étudions l'approche DC et nous proposons une version en ligne de DCA, appelée Online DCA, pour résoudre les problèmes en ligne correspondants. Au cas où chaque sous-problème convexe d'Online DCA ne peut pas être résolu explicitement ou n'est pas facile à résoudre, nous proposons un schéma particulier basé sur Online DCA, nommé ODCA, où chaque sous-problème est résolu en approximant par une itération de la méthode de sous-gradient classique. Nous analysons les propriétés de l'ODCA en termes de regret (c'est-à-dire la différence entre la perte cumulative subie et la plus petite perte cumulative tout au long du processus d'apprentissage). Nous indiquons également que les variantes de l'algorithme de gradient en ligne sont une version spéciale d'Online DCA. Comme application, nous considérons la classification en ligne où, à chaque itération, l'apprenant doit donner le classificateur à prédire l'étiquette correspondant à l'instance à venir basé sur l'étiquette correcte révélée plus tard et les classificateurs précédents. Dans ce cas, la qualité du classificateur est souvent mesurée par la fonction de perte 0-1 qui renvoie 1 si l'étiquette prédite est la même que l'étiquette correcte et 0 autrement. En effet, cette fonction est non convexe et non différentiable, mais elle peut être approximée par des fonctions DC. Dans nos travaux, nous proposons trois différentes fonctions d'approximation DC, y compris deux formes de fonction polyhédrale et une forme de fonction sigmoïde. En utilisant l'ODCA pour résoudre les problèmes d'optimisation résultant, nous proposons trois algorithmes en ligne correspondants et analysons le regret de chaque algorithme basé sur principaux résultats de l'analyse d'ODCA mentionnée ci-dessus. Les résultats de l'expérience sur une variété de ensembles de données de classification montrent l'efficacité de nos algorithmes proposés par rapport aux algorithmes de classification en ligne existants.

Le chapitre 3 concerne une autre classe de techniques d'apprentissage en ligne, à savoir la prédiction avec des conseils d'experts ("prediction with expert advice" en anglais). Le paradigme de la prédiction avec les conseils d'experts est introduit comme modèle d'apprentissage en ligne. Il se caractérise par la prise d'une prédiction repose sur la base des prédictions des experts via le vecteur de poids attribué aux experts. De même que le chapitre 2, quand la fonction de perte évaluant la qualité de la prédiction est non convexe et/ou non différentiable, nous pouvons étudier les approches d'approximation DC. En fait, le vecteur de poids appartient souvent à un ensemble particulier (par exemple, simplexe des probabilités), et donc nous proposons deux schémas en ligne en particulier basés sur ODCA, nommés ODCA-SG et DCA-ESG, où chaque sous-problème est résolu en approximant par une itération de la méthode de sous-gradient projeté et la méthode de sous-gradient exponentié, respectivement. Nous analysons également les deux schémas en termes de regret. Nous développons les techniques de prédiction avec des conseils d'experts pour résoudre les problèmes de classification en ligne où les experts sont représentés par les algorithmes de classification en ligne bien connus. Avec une fonction d'approximation DC, nous obtenons deux algorithmes

en ligne basés sur ODCA-SG, DCA-ESG, et analysons la borne de regret pour ces algorithmes. La performance des algorithmes proposés est vérifiée en comparant de trois algorithmes standards existants sur différents ensembles de données benchmark.

Les chapitres 4, 5, 6 se concentrent sur l'étude des techniques d'apprentissage par renforcement en mode batch et en mode en ligne. Le chapitre 4 présente brièvement l'apprentissage par renforcement et ses travaux connexes. L'apprentissage par renforcement vise à estimer la politique optimale dans un environnement dynamique qui est généralement formulé comme un processus de décision markovien ("Markov decision process" en anglais) avec un modèle incomplet. Il est bien connu que nous pouvons aborder cette tâche par le problème de trouver le zéro du soi-disant résidu optimal de Bellman ("optimal Bellman residual" en anglais), un concept classique de programmation dynamique. Il existe quelques travaux dans la littérature suivant cette direction, sachant qu'il résulte à un problème d'optimisation non convexe qui est très difficile à résoudre exactement. Dans le chapitre 5, nous considérons quatre formulations d'optimisation de ce problème qui minimisent le ℓ_p-norm du résidu optimal de Bellman avec $p \in \{1, 2, +\infty\}$ pour développer les techniques d'apprentissage par renforcement en mode batch (c'est-à-dire un ensemble fixe d'expériences d'apprentissage est donné a priori) en utilisant l'approximation linéaire de la fonction de valeur. Ils sont formulés comme programmes DC pour lesquels quatre schémas DCAs sont développés. En exploitant la structure spéciale du résidu optimal de Bellman empirique avec approximation linéaire, nous abordons les questions clés de DCA, en particulier la effet des décompositions DC, l'efficacité des méthodes de solution pour résoudre le sous-problème convexe résultant, et la recherche de bons points initiaux, lors de la conception des quatre algorithmes basés sur DCA. Expériences numériques sur deux benchmarks des problèmes de processus de décision markovien – le problème de Garnet et Gridworld – montrent l'efficacité de nos approche en comparaison avec deux algorithmes existants basés sur DCA et deux algorithmes d'apprentissage par renforcement. En poursuivant ces travaux, le chapitre 6 vise à développer des techniques d'apprentissage par renforcement en mode en ligne via une formule d'optimisation de ℓ_p-norm ($p = 2$). Nous proposons un algorithme en ligne basé sur DCA (ODCA) qui a la propriété de stabilité en ligne. Nous indiquons qu'une classe d'algorithmes de gradient en RL est un cas particulier de notre schéma ODCA. Nous suggérons également une version alternative d'ODCA pour exploiter la connaissance des échantillons. Les résultats numériques sur deux problèmes benchmark – le problème de Mountain car et Pole balancing – indiquent l'efficacité de nos approches en comparaison avec deux algorithmes standard d'apprentissage par renforcement.

Dans le chapitre 7, nous utilisons la technique d'apprentissage par renforcement basée sur DCA pour aborder un des problèmes classiques dans le domaine de tournées de véhicules qui est le problème du plus court chemin stochastique ("stochastic shortest path" en anglais). Nos travaux concernent ce problème pour un seul véhicule indépendant sur un réseau routier en utilisant le critère basé sur le modèle de probabilité à queue. En particulier, ce problème du plus court chemin stochastique vise à rechercher un chemin optimal qui maximise la probabilité d'arriver à la destination avant un délai donné. Il existe deux approches qui formulent ce problème, respectivement, comme une formulation de minimisation de cardinalité (la cardinalité

d'un vecteur est le nombre d'éléments non nuls dans ce vecteur) et une formulation de l'apprentissage par renforcement. Pour la première approche, il existe une double difficulté: le terme de cardinalité et les variables binaires. Certains algorithmes ont récemment été proposés en approximant le terme de cardinalité, cependant, sans traiter les variables binaires. Ainsi, dans ce travail, nous étudions une approche d'approximation DC pour le terme de cardinalité, et employons une technique de pénalité exacte pour les variables binaires. La formulation d'optimisation résultante peut être exprimée sous la forme d'un programme DC pour lequel l'algorithme basé sur DCA, nommé Card-DCA, est proposé. Les résultats de l'expérience montrent l'efficacité de Card-DCA en termes de qualité et rapidité par rapport aux algorithmes existants. Au regard de la formulation de l'apprentissage par renforcement, nous tenons compte du problème d'optimisation de ℓ_1-norm dans lequel l'ensemble d'échantillons donné est défini en fonction des données de temps de déplacement sur le réseau routier. Nous donc proposons un algorithme basé sur DCA, nommé RL-DCA, pour les problèmes du plus court chemin stochastique. Plusieurs expériences numériques sur les réseaux routiers artificiels sont menées afin de comparer deux approches DCA pour ces problèmes, en particulier les algorithmes Card-DCA et RL-DCA.

Organisation de la Thèse

La thèse est composée de huit chapitres. Le premier chapitre décrit brièvement les concepts fondamentaux et les principaux résultats de l'analyse convexe, la programmation DC et DCA, et sa version en ligne, ce qui fournit la base théorique et algorithmique pour les chapitres suivants. Les six chapitres suivants sont divisés en deux parties. La première partie, y compris les chapitres 2 et 3, concerne les techniques d'apprentissage en ligne. Plus précisément, dans le chapitre 2, nous présentons l'approche basée sur Online DCA pour les problèmes d'apprentissage en ligne et développons des techniques en ligne correspondantes pour une classe de problèmes de classification en ligne. En poursuivant cette direction, le chapitre 3 se concentre sur la conception d'une autre technique d'apprentissage en ligne, à savoir la prédiction avec des conseils d'experts. La deuxième partie (chapitres 4, 5, 6, 7) concerne en particulier les techniques d'apprentissage par renforcement. Sa brève introduction et ses travaux connexes sont présentés au chapitre 4. Les techniques d'apprentissage par renforcement basées sur DCA en mode batch et en mode en ligne sont développées respectivement au chapitre 5 et au chapitre 6. Dans le chapitre 7, nous considérons une classe de problèmes du plus court chemin stochastique via des approches DCA. Les conclusions et les perspectives de nos travaux sont données au chapitre 8.

Chapter 1

Preliminary

This chapter summarizes some basic concepts and results that will be the groundwork of the dissertation.

1.1 DC programming and DCA

DC programming and DCA, which constitute the backbone of nonconvex programming and global optimization, were introduced by Pham Dinh Tao in their preliminary form in 1985 [93]. Important developments and improvements on both theoretical and computational aspects have been completed since 1994 throughout the joint works of Le Thi Hoai An and Pham Dinh Tao. In this section, we present some basic properties of convex analysis and DC optimization and DC Algorithm that computational methods of this dissertation are based on. The materials of this section are extracted from [55, 65, 66, 95].

Throughout this section, X denotes the Euclidean space $\mathbb{R}^n$ and $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$ is the set of extended real numbers.

1.1.1 Fundamental convex analysis

First, let us recall briefly some notions and results in convex analysis related to the dissertation (refer to the references [15, 95, 101] for more details).

A subset C of X is said to be *convex* if $(1-\lambda)x + \lambda y \in C$ whenever $x,\ y \in C$ and $\lambda \in [0,1]$.

Let f be a function whose values are in $\overline{\mathbb{R}}$ and whose domain is a subset S of X. The set

$$\{(x,t) : x \in S,\ t \in \mathbb{R}, f(x) \leq t\}$$

is called the *epigraph* of f and is denoted by epi$f$.

We define f to be a *convex function* on S if epi$f$ is convex set in $X \times \mathbb{R}$. This is equivalent to that S is convex and

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y), \quad \forall x, y \in S,\ \forall \lambda \in [0,1].$$

The function f is *strictly convex* if the inequality above holds strictly whenever x and y are distinct in S and $0 < \lambda < 1$.

The *effective domain* of a convex function f on S, denoted by dom$f$, is the projection on X of the epigraph of f

$$\text{dom}f = \{x \in X : \exists t \in \mathbb{R}, (x,t) \in \text{epi}f\} = \{x \in X : f(x) < +\infty\}$$

and obviously, it is convex.

The convex function f is called *proper* if dom$f \neq \emptyset$ and $f(x) > -\infty$ for all $x \in S$.

The function f is said to be *lower semi-continuous* at a point x of S if

$$f(x) \leq \liminf_{y \to x} f(y).$$

Denote by $\Gamma_0(X)$ the set of all proper lower semi-continuous convex functions on X.

Let ρ be a nonnegative number and C be a convex subset of X. One says that a function $\theta : C \to \mathbb{R} \cup \{+\infty\}$ is ρ–convex if

$$\theta[\lambda x + (1-\lambda)y] \leq \lambda\theta(x) + (1-\lambda)\theta(y) - \frac{\lambda(1-\lambda)}{2}\rho\|x-y\|^2$$

for all $x, y \in C$ and $\lambda \in (0,1)$. It amounts to say that $\theta - (\rho/2)\|\cdot\|^2$ is convex on C. The modulus of strong convexity of θ on C, denoted by $\rho(\theta, C)$ or $\rho(\theta)$ if $C = X$, is given by

$$\rho(\theta, C) = \sup\{\rho \geq 0 : \theta - (\rho/2)\|\cdot\|^2 \text{ is convex on } C\}.$$

One says that θ is *strongly convex* on C if $\rho(\theta, C) > 0$.

A vector y is said to be a *subgradient* of a convex function f at a point x^0 if

$$f(x) \geq f(x^0) + \langle x - x^0, y\rangle, \qquad \forall x \in X.$$

The set of all subgradients of f at x^0 is called the *subdifferential* of f at x^0 and is denoted by $\partial f(x^0)$. If $\partial f(x)$ is not empty, f is said to be *subdifferentiable* at x.

For $\varepsilon > 0$, a vector y is said to be an *ε–subgradient* of a convex function f at a point x^0 if

$$f(x) \geq (f(x^0) - \varepsilon) + \langle x - x^0, y\rangle, \qquad \forall x \in X.$$

The set of all ε–subgradients of f at x^0 is called the *ε–subdifferential* of f at x^0 and is denoted by $\partial_\varepsilon f(x^0)$.

Let us describe two basic notations as follows.

$$\text{dom } \partial f = \{x \in X : \partial f(x) \neq \emptyset\} \ \text{ and } \ \text{range } \partial f(x) = \cup\{\partial f(x) : x \in \text{dom } \partial f\}.$$

Proposition 1.1. *Let f be a proper convex function. Then*

1. *$\partial_\varepsilon f(x)$ is a closed convex set, for any $x \in X$ and $\varepsilon \geq 0$.*
2. $\mathrm{ri}(\mathrm{dom} f) \subset \mathrm{dom}\ \partial f \subset \mathrm{dom} f$
   where $\mathrm{ri}(\mathrm{dom} f)$ *stands for the* relative interior *of* $\mathrm{dom} f$.
3. *If f has a unique subgradient at x, then f is differentiable at x, and $\partial f(x) = \{\nabla f(x)\}$.*
4. *$x_0 \in \mathrm{argmin}\{f(x) : x \in X\}$ if and only if $0 \in \partial f(x_0)$.*

Conjugates of convex functions

The *conjugate* of a function $f : X \to \overline{\mathbb{R}}$ is the function $f^* : X \to \overline{\mathbb{R}}$, defined by

$$f^*(y) = \sup_{x \in X}\{\langle x, y\rangle - f(x)\}.$$

Proposition 1.2. *Let $f \in \Gamma_0(X)$. Then we have*

1. *$f^* \in \Gamma_0(X)$ and $f^{**} = f$.*
2. *$f(x) + f^*(y) \geq \langle x, y\rangle$, for any $x, y \in X$.*
   Equality holds if and only if $y \in \partial f(x) \Leftrightarrow x \in \partial f^*(y)$.
3. *$y \in \partial_\varepsilon f(x) \Longleftrightarrow x \in \partial_\varepsilon f^*(y) \Longleftrightarrow f(x) + f^*(y) \leq \langle x, y\rangle + \varepsilon$, for all $\varepsilon > 0$.*

Polyhedral Functions

A *polyhedral* set is a closed convex set that is of form

$$C = \{x \in X : \langle x, b_i\rangle \leq \beta_i,\ \forall i = 1, \ldots, m\}$$

where $b_i \in X$ and $\beta_i \in \mathbb{R}$ for all $i = 1, \ldots, m$.

A function $f \in \Gamma_0(X)$ is said to be *polyhedral* if

$$f(x) = \max\{\langle a_i, x\rangle - \alpha_i : i = 1, \ldots, k\} + \chi_C(x), \quad \forall x \in X \tag{1.1}$$

where $a_i \in X$, $\alpha_i \in \mathbb{R}$ for all $i = 1, \ldots, k$ and C is a nonempty polyhedral set. It is clear that $\mathrm{dom}\, f = C$.

Proposition 1.3. *Let f be a polyhedral convex function, and $x \in \mathrm{dom} f$. Then we have*

1. *f is subdifferentiable at x, and $\partial f(x)$ is a polyhedral convex set. In particular, if f is defined by* (1.1) *with $C = X$ then*

$$\partial f(x) = \mathrm{co}\{a_i : i \in I(x)\}$$

   where $I(x) = \{i \in \{1, \ldots, k\} : \langle a_i, x\rangle - \alpha_i = f(x)\}$.

2. The conjugate f^* is a polyhedral convex function. Moreover, if $C = X$ then

$$\mathrm{dom} f^* = \mathrm{co}\{a_i : i = 1, \ldots, k\},$$
$$f^*(y) = \inf\left\{\sum_{i=1}^{k} \lambda_i \alpha_i \Big| \sum_{i=1}^{k} \lambda_i a_i = y, \sum_{i=1}^{k} \lambda_i = 1, \lambda_i \geq 0, \forall i = 1, \ldots, k\right\}.$$

In particular,

$$f^*(a_i) = \alpha_i, \qquad \forall i = 1, \ldots, k.$$

Difference of Convex (DC) functions

A function f is called DC function on X if it is of the form

$$f(x) = g(x) - h(x), \quad x \in X$$

where g and h belong to $\Gamma_0(X)$. One says that $g - h$ is a *DC decomposition* of f and the functions g, h are its *DC components*. If g and h are in addition finite on all points of X then one says that $f = g - h$ is finite DC function on X. The set of DC functions (resp. finite DC functions) on X is denoted by $\mathcal{DC}(X)$ (resp. $\mathcal{DC}_f(X)$).

Remark 1.1. *Give a DC function f whose DC decomposition is $f = g - h$. Then for every $\theta \in \Gamma_0(X)$ finite on the whole X, $f = (g + \theta) - (h + \theta)$ is another DC decomposition of f. Thus, a DC function f has finitely many DC decompositions.*

1.1.2 DC optimization

DC program

In the sequel, we use the convention $+\infty - (+\infty) = +\infty$.

For $g, h \in \Gamma_0(X)$, a standard *DC program* is of the form

$$(P) \qquad \alpha = \inf\{f(x) = g(x) - h(x) : x \in X\}$$

and its dual counterpart

$$(D) \qquad \alpha^* = \inf\{h^*(y) - g^*(y) : y \in X\}.$$

There is a perfect symmetry between primal and dual programs (P) and (D): the dual program to (D) is exactly (P), moreover, $\alpha = \alpha^*$.

Remark 1.2. *Let C be a nonempty closed convex set. Then, the constrained problem*

$$\inf\{f(x) = g(x) - h(x) : x \in C\}$$

can be transformed into an unconstrained DC program by using the indicator function χ_C, i.e.,

$$\inf\{f(x) = \phi(x) - h(x) : x \in X\}$$

where $\phi := g + \chi_C$ belongs to $\Gamma_0(X)$.

We will always keep the following assumption that is deduced from the finiteness of α

$$\operatorname{dom} g \subset \operatorname{dom} h \quad \text{and} \quad \operatorname{dom} h^* \subset \operatorname{dom} g^*. \tag{1.2}$$

Polyhedral DC program

In the problem (P), if one of the DC components g and h is polyhedral function, we call (P) a *polyhedral DC program*. This is an important class of DC optimization. It is often encountered in practice and has worthy properties.

Consider the problem (P) where h is a polyhedral convex function given by

$$h(x) = \max\{\langle a_i, x\rangle - \alpha_i : i = 1, \ldots, k\}.$$

By Proposition 1.3, the dual problem (D) can be expressed as follows.

$$\begin{aligned} \alpha^* &= \inf\{h^*(y) - g^*(y) : y \in X\} \\ &= \inf\{h^*(y) - g^*(y) : y \in \operatorname{co}\{a_i : i = 1, \ldots, k\}\} \\ &= \inf\{\alpha_i - g^*(a_i) : i = 1, \ldots, k\}. \end{aligned}$$

Note that, if g is polyhedral convex and h is not, then by considering the dual problem (D), we have the similar formulation as above since g^* is polyhedral.

Optimality conditions for DC optimization

A point x^* is said to be a *local minimizer* of $g-h$ if $x^* \in \operatorname{dom} g \cap \operatorname{dom} h$ (so, $(g-h)(x^*)$ is finite) and there is a neighborhood U of x^* such that

$$g(x) - h(x) \geq g(x^*) - h(x^*), \qquad \forall x \in U. \tag{1.3}$$

A point x^* is said to be a *critical point* of $g-h$ if it verifies the generalized Kuhn–Tucker condition

$$\partial g(x^*) \cap \partial h(x^*) \neq \emptyset. \tag{1.4}$$

Let $\mathcal{P}$ and $\mathcal{D}$ denote the solution sets of problems (P) and (D) respectively, and let

$$\mathcal{P}_\ell = \{x^* \in X : \partial h(x^*) \subset \partial g(x^*)\}, \quad \mathcal{D}_\ell = \{y^* \in X : \partial g^*(y^*) \subset \partial h^*(y^*)\}.$$

In the following, we present some fundamental results on DC programming [65].

Theorem 1.1. **i)** *Global optimality condition: $x \in \mathcal{P}$ if and only if*

$$\partial_\varepsilon h(x) \subset \partial_\varepsilon g(x), \ \forall \varepsilon > 0.$$

ii) *Transportation of global minimizers: $\cup\{\partial h(x) : x \in \mathcal{P}\} \subset \mathcal{D} \subset \operatorname{dom} h^*$.*
The first inclusion becomes equality if g^* is subdifferentiable in $\mathcal{D}$. In this case, $\mathcal{D} \subset (\operatorname{dom} \partial g^* \cap \operatorname{dom} \partial h^*)$.

iii) *Necessary local optimality: if x^* is a local minimizer of $g-h$, then $x^* \in \mathcal{P}_\ell$.*

iv) *Sufficient local optimality: Let x^* be a critical point of $g-h$ and $y^* \in \partial g(x^*) \cap \partial h(x^*)$. Let U be a neighborhood of x^* such that $(U \cap \operatorname{dom} g) \subset \operatorname{dom} \partial h$. If for any $x \in U \cap \operatorname{dom} g$, there is $y \in \partial h(x)$ such that $h^*(y) - g^*(y) \geq h^*(y^*) - g^*(y^*)$, then x^* is a local minimizer of $g-h$. More precisely,*

$$g(x) - h(x) \geq g(x^*) - h(x^*), \qquad \forall x \in U \cap \operatorname{dom} g.$$

v) *Transportation of local minimizers: Let $x^* \in \operatorname{dom} \partial h$ be a local minimizer of $g-h$. Let $y^* \in \partial h(x^*)$ and a neighborhood U of x^* such that $g(x) - h(x) \geq g(x^*) - h(x^*),\ \forall x \in U \cap \operatorname{dom} g$. If*

$$y^* \in \operatorname{int}(\operatorname{dom} g^*) \quad \text{and} \quad \partial g^*(y^*) \subset U$$

then y^* is a local minimizer of $h^* - g^*$.

Remark 1.3. *a) By the symmetry of the DC duality, these results have their corresponding dual part. For example, if y is a local minimizer of $h^* - g^*$, then $y \in \mathcal{D}_\ell$.*

b) The properties ii), v) and their dual parts indicate that there is no gap between the problems (P) and (D). They show that globally/locally solving the primal problem (P) implies globally/locally solving the dual problem (D) and vice–versa. Thus, it is useful if one of them is easier to solve than the other.

c) The necessary local optimality condition $\partial h^*(x^*) \subset \partial g^*(x^*)$ is also sufficient for many important classes programs, for example [66], if h is polyhedral convex, or when f is locally convex at x^*, i.e. there exists a convex neighborhood U of x^* such that f is finite and convex on U. We know that a polyhedral convex function is almost everywhere differentiable, that is to say, it is differentiable everywhere except on a set of measure zero. Thus, if h is a polyhedral convex function, then a critical point of $g-h$ is almost always a local solution to (P).

d) If f is actually convex on X, we call (P) a "false" DC program. *In addition, if $\operatorname{ri}(\operatorname{dom} g) \cap \operatorname{ri}(\operatorname{dom} h) \neq \emptyset$ and $x^0 \in \operatorname{dom} g$ such that g is continuous at x^0, then $0 \in \partial f(x^0) \Leftrightarrow \partial h(x^0) \subset \partial g(x^0)$ [66]. Thus, in this case, the local optimality is also sufficient for the global optimality. Consequently, if in addition h is differentiable, a critical point is also a global solution.*

1.1.3 DC Algorithm (DCA)

The DCA consists in the construction of the two sequences $\{x^k\}$ and $\{y^k\}$ (candidates for being primal and dual solutions, respectively) which are easy to calculate and satisfy the following properties:

i) The sequences $(g-h)(x^k)$ and $(h^* - g^*)(y^k)$ are decreasing.

ii) Their corresponding limits x^∞ and y^∞ either satisfy the local optimality condition $(x^\infty, y^\infty) \in \mathcal{P}_\ell \times \mathcal{D}_\ell$ or are critical points of $g-h$ and $h^* - g^*$, respectively.

From a given initial point $x^0 \in \operatorname{dom} g$, the DCA generates these sequences by the scheme

$$y^k \in \partial h(x^k) = \arg\min\{h^*(y) - \langle y, x^k\rangle : y \in X\}, \tag{1.5a}$$

$$x^{k+1} \in \partial g^*(y^k) = \arg\min\{g(x) - \langle x, y^k\rangle : x \in X\}. \tag{1.5b}$$

The interpretation of the above scheme is simple. At iteration k of DCA, one replaces the second component h in the primal DC program by its affine minorant

$$h_k(x) = h(x^k) + \langle x - x^k, y^k\rangle, \tag{1.6}$$

where $y^k \in \partial h(x^k)$. Then the original DC program is reduced to the *convex program*

$$(P_k) \qquad \alpha_k = \inf\{f_k(x) := g(x) - h_k(x) : x \in X\}$$

that is equivalent to (1.5b). It is easy to see that f_k is a majorant of f which is exact at x^k i.e. $f_k(x^k) = f(x^k)$. Similarly, by replacing g^* with its affine minorant

$$g_k^*(y) = g^*(y^{k-1}) + \langle y - y^{k-1}, x^k\rangle \tag{1.7}$$

where $x^k \in \partial g^*(y^{k-1})$, it leads to the convex program

$$(D_k) \qquad \inf\{h^*(y) - g_k^*(y) : y \in X\}$$

whose solution set is $\partial h(x^k)$.

Well definiteness of DCA

DCA is well defined if one can construct two sequences $\{x^k\}$ and $\{y^k\}$ as described above from an arbitrary initial point. The following lemma is the necessary and sufficient condition for this property.

Lemma 1.1 ([65])**.** *The sequences $\{x^k\}$ and $\{y^k\}$ in DCA are well defined if and only if*

$$\operatorname{dom}\partial g \subset \operatorname{dom}\partial h \quad \text{and} \quad \operatorname{dom}\partial h^* \subset \operatorname{dom}\partial g^*.$$

Since for $\varphi \in \Gamma_0(X)$ one has $\operatorname{ri}(\operatorname{dom}\varphi) \subset \operatorname{dom}\partial\varphi \subset \operatorname{dom}\varphi$ (Proposition 1.1). Moreover, under the assumptions $\operatorname{dom} g \subset \operatorname{dom} h$, $\operatorname{dom} h^* \subset \operatorname{dom} g^*$, one can say that DCA in general is well defined.

Convergence properties of DCA

Complete convergence of DCA is given in the following results [65].

Theorem 1.2. *Suppose that the sequences $\{x^k\}$ and $\{y^k\}$ are generated by DCA. Then we have*

i) *The sequences $\{g(x^k) - h(x^k)\}$ and $\{h^*(y^k) - g^*(y^k)\}$ are decreasing and*

- *$g(x^{k+1}) - h(x^{k+1}) = g(x^k) - h(x^k)$ if and only if $\{x^k, x^{k+1}\} \subset \partial g^*(y^k) \cap \partial h^*(y^k)$ and $[\rho(h) + \rho(g)]\|x^{k+1} - x^k\| = 0$.*

- $h^*(y^{k+1}) - g^*(y^{k+1}) = h^*(y^k) - g^*(y^k)$ *if and only if* $\{y^k, y^{k+1}\} \subset \partial g(x^k) \cap \partial h(x^k)$ *and* $[\rho(h^*) + \rho(g^*)]\|y^{k+1} - y^k\| = 0$.

DCA terminates at the kth iteration if either of the above equalities holds.

ii) *If* $\rho(h) + \rho(g) > 0$ *(resp.* $\rho(h^*) + \rho(g^*) > 0$*), then the sequence* $\{\|x^{k+1} - x^k\|^2\}$ *(resp.* $\{\|y^{k+1} - y^k\|^2\}$*) converges.*

iii) *If the optimal value* α *is finite and the sequences* $\{x^k\}$ *and* $\{y^k\}$ *are bounded, then every limit point* x^∞ *(resp.* y^∞*) of the sequence* $\{x^k\}$ *(resp.* $\{y^k\}$*) is a critical point of* $g - h$ *(resp.* $h^* - g^*$*).*

iv) *DCA has a linear convergence for general DC program.*

v) *In polyhedral DC programs, the sequences* $\{x^k\}$ *and* $\{y^k\}$ *contain finitely many elements and DCA has a finite convergence.*

vi) *If DCA converges to a point* x^* *that admits a convex neighborhood in which the objective function* f *is finite and convex (i.e. the function* f *is locally convex at* x^**) and if the second DC component* h *is differentiable at* x^**, then* x^* *is a local minimizer to the problem* (P)*.*

Remark 1.4. *a) Finding* y^k*,* x^{k+1} *based on the scheme 1.5 amounts to solving the problems* (D_k) *and* (P_k)*. Thus, DCA works by reducing a DC program to a sequence of convex programs which can be solved efficiently.*

b) In practice, the calculation of the subgradient of the function h *at a point* x *is usually easy if we know its explicit expression. But, the explicit expression of the conjugate of a given function* g *is unknown, so calculating* x^{k+1} *is done by solving the convex problem* (P_k)*. For the large-scale setting, the solutions to the problem* (P_k) *should be either in an explicit form or achieved by efficient algorithms with inexpensive computations.*

c) When h *is a polyhedral function, the calculation of the subdifferential* $\partial h(x^k)$ *is explicit by Proposition 1.3. With a fixed choice of subgradients of* h*, the sequence* $\{y^k\}$ *is discrete i.e. it has only finitely many different elements. This leads to finite convergence of DCA.*

d) DCA's distinctive feature relies upon the fact that DCA deals with the convex DC components g *and* h *but not with the DC function* f *itself. Moreover, a DC function* f *has infinitely many DC decompositions which have crucial implications for the qualities (e.g. convergence speed, robustness, efficiency, globality of computed solutions) of DCA. For a given DC program, the choice of optimal DC decompositions is still open. Of course, this depends strongly on the very specific structure of the problem being considered.*

e) Similarly to the effect of DC decompositions on DCA, searching the good initial points for DCA is also an open question to be studied.

1.2 Online DC programming and Online DCA

Online DC programming and Online DCA is seen as an online version of DC programming and DCA. While DC programming aims to search a point to minimize a DC function, in Online DC programming, the DC programming problems are solved iteratively as *an online process*. In recent years, the more and more emergence of the online process of nonconvex, nonsmooth optimization problems in various domains of applied sciences, specially in machine learning, motivates us to develop a novel approach based on Online DC programming and Online DCA. In this section, we present some notations and fundamental definitions relating to the Online DC programming and Online DCA framework.

Throughout this section, T denotes the total number of steps in online process, t denotes the step index and X denotes the Euclidean space $\mathbb{R}^n$.

1.2.1 Online DC optimization

Online DC programming

Online DC programming consists of a feasible set X and a sequence of the function $\{f_t\}_{t=1,\ldots,T}$ where each $f_t : X \to \overline{\mathbb{R}}$ is a DC function whose DC decomposition is $g_t - h_t$ where g_t and $h_t \in \Gamma_0(X)$. For simplicity, instead of X we consider the feasible set is a nonempty, bounded, closed, convex subset of X, denoted by $\mathcal{S}$. In this case, an Online DC programming problem can be described as follows. At the step t, one tries to find the *best* feasible point in $\mathcal{S}$, denoted w^t, by solving a DC program of form

$$\inf\{F_t(w) := G_t(w) - H_t(w) : w \in \mathcal{S}\} \tag{1.8}$$

where the functions F_t, G_t and H_t depend on previous DC functions f_i, g_i and h_i, $i = 1, \ldots, t-1$, respectively. Then, one observes a new DC function f_t for that step.

The ultimate goal is to generate the sequence $\{w^t\}_{t=1,\ldots,T}$ that minimizes the cumulative objective value until step T, defined as

$$\sum_{t=1}^{T} f_t(w^t). \tag{1.9}$$

Minimizing (1.9) depends on the update rule to generate the sequence of $\{w^t\}_{t=1,\ldots,T}$, more precisely what DC optimization problem is considered at each step. In this work, we propose the following update rule

$$w^{t+1} \in \operatorname*{arg\,min}_{w \in \mathcal{S}} \; F_t(w), \; t = 1, \ldots, T, \tag{1.10}$$

where the objective function

$$F_t(w) := \sum_{i=t_0}^{t} f_i(w) + R(w),$$

the function $R : \mathcal{S} \to \mathbb{R}$ is a convex regularization function, and $t_0 \in \{1, t\}$.

Obviously, F_t is a DC function. Provided that at the step t, a DC decomposition of F_t is defined as

$$F_t = G_t - H_t,$$

where the function $G_t(w) = \sum_{i=t_0}^{t} g_i(w) + R(w)$ and $H_t(w) = \sum_{i=t_0}^{t} h_i(w)$.

1.2.2 Online DCA

Recall that for solving a DC program, DCA has the quite simple principle: at each iteration k, DCA approximates the concave part $-h$ by its affine majorization (by choosing a subgradient of h at x^k) and minimizes the resulting convex function. Founded on the idea of DCA, we derive an online version of standard DCA, which is called Online DCA. In particular, at the step t, one replaces the second DC component h_t of DC loss function f_t by its affine minorization (corresponding to taking $z^t \in \partial h_t(w^t)$) and then updates the point w^{t+1} for the next step $t+1$ by solving the convex subproblem (1.11).

Online DCA

Initialization: let w^1 be an initial point in $\mathcal{S}$
for $t = 1, 2, \ldots, T$ **do**
Compute $z^t \in \partial h_t(w^t)$.
Compute w^{t+1}, an optimal solution to the convex program

$$\min \left\{ G_t(w) - \sum_{i=t_0}^{t} \langle z^i, w \rangle : w \in \mathcal{S} \right\}. \tag{1.11}$$

end for

1.2.3 ODCA: a proposed Online DCA based scheme

In Online DCA scheme, at each step, we need to solve the convex subproblem (1.11). However, in some practice problems, (1.11) cannot be solved explicitly or is solved with expensive computaions. Thus, we propose a particular Online DCA based scheme where each convex subproblem (1.11) is solved by approximating by one iteration of subgradient methods (see e.g. [111] for more details about subgradient methods). Particularly, at the step t, considering $t_0 = t$ and $R(w) = 0$, the subproblem (1.11) becomes

$$\min \left\{ g_t(w) - \langle z^t, w \rangle : w \in \mathcal{S} \right\}, \tag{1.12}$$

where $z^t \in \partial h_t(w^t)$.

Approximating (1.11) by one iteration of the subgradient methods at the point w^t with the step size of η_t, we derive the following update rule

$$w^{t+1} = \Psi(\eta_t, w^t, s^t), \tag{1.13}$$

where $s^t \in (\partial g_t(w^t) - z^t)$ and Ψ is the appropriate operator of subgradient methods (for instance the projection operator $\text{Proj}_{\mathcal{S}}(\cdot)$ of the point $w^t - \eta_t s^t$ corresponding to the projected subgradient method).

Thus, a particular Online DCA based scheme (ODCA) applied to the Online DC programming problem (1.10) can be described as follows.

ODCA: Online DCA-Subgradient scheme

Initialization: let w^1 be an initial point in $\mathcal{S}$, $\{\eta_t\}$ be the sequence of step sizes.

for $t = 1, 2, \ldots, T$ **do**

1. Compute $z^t \in \partial h_t(w^t)$.
2. Compute $s^t \in (\partial g_t(w^t) - z^t)$.
3. Compute w^{t+1} using (1.13).

end for

1.2.4 Analysis of ODCA

In this section, we will present the fundamental concepts that are used to study the analysis of the ODCA based algorithms in the next chapters. First of all, we restate the ultimate goal mentioned previously as a performance metric of the series of points obtained by the algorithm in comparison with the best fixed point in terms of the cumulative objective value in hindsight, which is called a *regret* of the algorithm.

Definition 1.1 ([45, 134])**.** *Assume that the sequence of the points* $\{w^1, w^2, \ldots, w^T\}$ *is generated by an algorithm* A*. Formally, the regret of* A *until step* T *is defined as*

$$\mathsf{Regret}_{\mathrm{A}}^T := \sum_{t=1}^{T} f_t(w^t) - \min_{w \in \mathcal{S}} \sum_{t=1}^{T} f_t(w). \tag{1.14}$$

We aim to show that the average regret of ODCA tends to zeros when T tends to infinity, that is to say, the performance of ODCA (1.9) is as good as that of the best fixed point in hindsight on the average. To attain that interesting property, we first indicate the regret bound of ODCA as a function of T. In fact, the analysis of ODCA depends largely on the update rule (1.13). Therefore, the different ODCA based algorithms lead to the different regret bounds, which will be stated in more detail in next chapters.

Remark 1.5. *It would be interesting to note that if the sequence of points* $\{w^t\}_{t=1,\ldots,T}$ *is generated by the* ODCA *algorithms, then it is easy to obtain the following property:*

$$\sum_{t=1}^{T} f_t(w) \leq \sum_{t=1}^{T} f_t^C(w), \ \forall w \in \mathcal{S}, \tag{1.15}$$

and, specially,

$$\sum_{t=1}^{T} f_t(w^t) = \sum_{t=1}^{T} f_t^C(w^t), \tag{1.16}$$

where $f_t^C(w) = g_t(w) - (h_t(w^t) + \langle z^t, w - w^t \rangle), t = 1, \ldots, T$, *is a convex majorization of* $f_t(w)$. *From this fact, we can observe that by exploiting online convex algorithms to give the upper bound for the cumulative objective value corresponding to the convex loss function* f_t^C, *we will derive the upper bound for* (1.9) *with respect to the DC loss functions* $\{f_t\}_{t=1,\ldots,T}$, *which is described in [46].*

Another definition, namely online stability, is one of many quantities for any online algorithm [103, 104]. The online stability condition means that on the average, the difference between consecutive points generated by the online algorithm is eventually small.

Definition 1.2 ([104])**.** *Assume that the sequence of the points* $\{w^1, w^2, \ldots, w^T, w^{T+1}\}$ *is generated by an algorithm* A. *The (cumulative) online stability of* A *is defined as*

$$\mathsf{S}_{\mathrm{A}}^T := \sum_{t=1}^{T} \|w^t - w^{t+1}\|. \tag{1.17}$$

An algorithm A *is called to be online stable if* $\mathsf{S}_{\mathrm{A}}^T = o(T)$ – *that is,*

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \|w^t - w^{t+1}\| = 0. \tag{1.18}$$

Part I

Online learning

Chapter 2

Online Learning and Applications to Online Binary Linear Classification[1]

Abstract: *This chapter concerns an approach based on an online version of DC (Difference of Convex functions) programming and DCA (DC Algorithm) for developing machine learning techniques in online mode (called online learning for short). Online learning is a class of techniques at the interface between machine learning and online optimization since its main goal is to minimize the cumulative loss between the predicted answer and the correct one along its learning process. There exist many efficient online learning techniques developed based on online convex optimization approaches. However, their disadvantages have recently been highlighted when facing the nonconvex answer domain or/and the nonconvex, nonsmooth loss function in most applications. In this chapter, we consider Online DC programming for online learning, propose an online version of DCA, called Online DCA, and indicate that Online DCA covers some variants of online gradient descent. In addition, we design a particular Online DCA based scheme, named ODCA, and study its analysis in terms of regret bound. As an application, we intensively investigate an Online DCA approach for the problems in the topic of online binary linear classification. Exploiting different DC approximation functions, we develop three corresponding ODCA based algorithms and analyze their regret bounds. Numerical experiments on a variety of benchmark datasets show the efficiency of our proposed algorithms in comparison with the state-of-the-art online classification algorithms.*

1. The material of this chapter is developed from the following work:
[1]. Vinh Thanh Ho, Hoai An Le Thi, Dinh Chien Bui. Online DC Optimization for Online Binary Linear Classification. In: Nguyen et al. (eds) Intelligent Information and Database Systems. ACIIDS 2016. Lecture Notes in Computer Science, vol 9622, pp. 661–670, Springer, 2016.

2.1 Introduction

2.1.1 Background and related works

In recent years, it is neccessary to study many innovative techniques for treating the problems with an immense amount of data. Among certain techniques, developing machine learning techniques in online mode (online learning for short (OL)) occupies an important place. Online learning can be seen as the process of predicting an answer to the sequential arrival of questions based on the knowledge of the correct answers corresponding to previous questions and possibly other available information [109]. Formally, at the step t, the learner receives a question relevant to an input data, described as a vector x_t. Then he makes a prediction in order to answer that question. In fact, his prediction is constructed based on a hypothesis, $h^t : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the space of questions and possible answers respectively. After that, he gets the correct answer to the question, cast as a vector $y_t \in \mathcal{Y}$. To assess the quality of his prediction, he suffers a loss between the predicted answer and the correct one, defined by $\ell_t(h^t, (x_t, y_t))$, where the loss function ℓ_t is used to update the hypothesis for the answer to the new question at the next step. The main goal for the learner is to generate the sequence of hypotheses which minimizes the cumulative suffered loss along its learning process.

Online learning plays a significant role in multiple contexts: when the data samples are available over time, the predictions must be made in real time, the learner is required to dynamically adapt to new data patterns, or even learning over the entire data at once is impossible in the computational aspect. Online learning has a wide range of applications such as online advertisement placement, online web ranking, online email categorization, prediction of stock prices and currency exchange, real-time recommendation [108, 109].

Due to its goal, online learning is seen as a general class of techniques at the interface between machine learning and online optimization. Up until now, most of the effective online learning algorithms have been derived from online convex optimization (see e.g. [108, 109, 134] for more details). The paragidm of online convex optimization was introduced by Zinkevich [110, 134], where the prediction domain and the loss function are convex. In online convex optimization, there exists a common update rule to select the hypothesis at each step: Follow-the-Leader (FTL) [49] or/and its regularization form (RFTL) [108, 110]. In FTL, the learner chooses the hypothesis that has the minimum cumulative loss function over all past steps, whereas RFTL makes a stability for the predictions of online learning algorithms. From these, many effective online convex algorithms were proposed such as online gradient descent (with lazy/greedy projections) [134], exponentiated gradient [4, 51, 50], p-norm [40]. However, the difficulties in online learning are that loss functions to assess the predictions are often nonsmooth, nonconvex (for example, 0-1 loss function returning 1 if the prediction is correct and 0 otherwise) or the domain of predictions is not convex. Hence solving the corresponding optimization problem becomes more intractable. Encountering the difficulty as just mentioned above, the disadvantage of using online convex optimization

approaches has been highlighted recently. Thus, it is essential to resort to nonconvex optimization in online mode to overcome the difficulties, which is the main purpose of our works in this chapter. There exist several challenges that should be addressed when solving the online learning problems: what optimization models in online mode for these problems, how to design simple, efficient online learning techniques, what convergence properties of these techniques, etc.

Recently, several related works have been studied for the problems in online learning with nonconvex loss functions (see e.g. [33, 37, 132]). In [33], the authors proposed a nonconvex online algorithm for Support Vector Machine problems with ramp loss function based on the concave-convex procedure. [37] presented an online algorithm for nonconvex Neyman-Pearson classification problems using gradient method. However, the analysis of the online algorithms in both of them in terms of the bounds of regret have been not indicated yet, where the regret is defined by the difference between the cumulative loss suffered and the possibly smallest cumulative loss along its run. With a formal regret bound, an algorithm for online bandit learning problems with nonconvex loss function (where only the suffered loss is available) was proposed in [132]. However, algorithms for general online learning problems in full information setting (i.e. where the knowledge of loss function is used) with theoretical guarantees have been not investigated yet. This motivates us to develop a nonconvex optimization approach to fill up the family of online algorithms for such online learning problems. The main algorithmic methodologies of this chapter are DC (Difference of Convex functions) programming and DCA (DC Algorithm) and their online version, which are well-known as powerful nonsmooth, nonconvex optimization tools. DC programming and DCA were introduced by Pham Dinh Tao in a preliminary form in 1985 and extensively developed since 1994 by Le Thi Hoai An and Pham Dinh Tao (see [66, 95, 96] and the references therein). DCA has been successfully applied to many (smooth or nonsmooth) large-scale nonconvex programs in various domains of applied sciences, in particular in machine learning, for which they provide quite often a global solution and are proved to be more robust and efficient than the standard methods (see the list of references in [56]).

2.1.2 Our contributions

This chapter focuses on solving a class of problems in online learning by an optimization approach based on the online version of DC programming and DCA. Our contributions are multiple.

Firstly, we present Online DC programming for online learning problems where the loss function to assess the quality of predictions at each step is a DC function. Then, we develop an online version of DCA, namely Online DCA, with a generalized update rule. And next, we propose a particular Online DCA based scheme, namely ODCA, where the convex subproblem is solved by approximating by one iteration of the classic subgradient method.

Secondly, we thoroughly analyze the regret bounds for the proposed ODCA algorithm

under some appropriate conditions for DC functions. Depending on the special properties of DC functions, the regret of ODCA grows sublinearly in the number of steps. Moreover, we are able to achieve a better regret bound, specifically logarithmic regret.

Thirdly, as an application, we focus on developing online learning techniques for solving the problems in an interesting topic of online classification, specifically online binary linear classification (OBLC). In this topic, instance becomes available in a sequential order and is used to update the linear classifier incrementally so as to predict the corresponding binary label for future instance. The loss function to assess the linear classifier at each step is usually 0-1 loss function, as defined above. In this chapter, we propose three DC approximation functions for this loss function, develop three corresponding ODCA based algorithms for solving these problems. Moreover, we also show that these proposed algorithms archive a logarithmic regret and as a result, the bounds of mistakes by predicting false labels are also given.

Finally, with the aim of evaluating the efficiency of our proposed online algorithms for solving the problems in OBLC, we conduct several numerical experiments on a variety of benchmark datasets in comparison with many state-of-the-art online algorithms.

The rest of the chapter is organized as follows. In Section 2.2, we first introduce Online DC programming and Online DCA for online learning, then indicate that online gradient descent algorithms are special cases of Online DCA and finally, show the formal regret bounds for a particular ODCA scheme. How to solve the OBLC problems by ODCA is described in Section 2.3. Section 2.4 reports the numerical results on several test problems which is followed by some conclusions in Section 2.5.

2.2 Online DC programming and Online DCA for Online learning

2.2.1 An introduction to Online DC programming and Online DCA for Online learning

Throughout this chapter, we will formally describe Online DC programming and Online DCA in the context of online learning, which is similar to Section 1.2.1. We further assume that T is the number of steps and the set of hypotheses, denoted by $\mathcal{S}$, is closed convex in $\mathbb{R}^n$. In this context, an Online DC programming problem involves the set of hypotheses $\mathcal{S}$ and a sequence of loss functions $\{f_t\}_{t=1,\ldots,T}$ where each real-value function $f_t : \mathcal{S} \to \mathbb{R}$ is a DC function. In particular, at step t, when a question is incoming, the learner must choose a hypothesis vector $w^t \in \mathcal{S}$ to predict an answer. The vector w^t is computed as an optimal solution to a DC program, which depends mainly on previous loss functions f_i and previous hypothesis vectors w^i, $i = 1, \ldots, t-1$. Then he observes a new DC function f_t to assess the quality of w^t. Finally he suffers a loss between the predicted answer and the correct one, defined by $f_t(w^t)$. The main goal of the Online DC programming problem is to generate the sequence of $\{w^t\}_{t=1,\ldots,T}$ that

minimizes the cumulative suffered loss

$$\sum_{t=1}^{T} f_t(w^t). \tag{2.1}$$

As Section 1.2.1, in this chapter, we consider the generalized update rule as follows.

$$w^{t+1} \in \arg\min_{w \in \mathcal{S}} \; F_t(w), \; t = 1, \ldots, T, \tag{2.2}$$

where the objective function

$$F_t(w) := \sum_{i=t_0}^{t} f_i(w) + R(w),$$

the function $R : \mathcal{S} \to \mathbb{R}$ is a convex regularization function, and $t_0 \in \{1, t\}$. It is worth noting that the update rule (2.2) covers existing update rules in online convex optimization such as FTL, RFTL mentioned in Section 2.1. However, (2.2) is difficult due to the nonconvexity of F_t. Let us assume that for $i = 1, \ldots, T$, a DC decomposition of f_i is $g_i - h_i$. It is evident to show that F_t is a DC function with a DC decomposition defined as

$$F_t = G_t - H_t,$$

where the function $G_t(w) = \sum_{i=t_0}^{t} g_i(w) + R(w)$ and $H_t(w) = \sum_{i=t_0}^{t} h_i(w)$.

Thus, we derive the online version of DCA, namely Online DCA, which is summarized as follows.

Online DCA

Initialization: let w^1 be an initial point in $\mathcal{S}$.
for $t = 1, 2, \ldots, T$ **do**
1. Compute $z^t \in \partial h_t(w^t)$.
2. Compute

$$w^{t+1} \in \arg\min \left\{ \sum_{i=t_0}^{t} g_i(w) + R(w) - \sum_{i=t_0}^{t} \langle z^i, w \rangle : w \in \mathcal{S} \right\}. \tag{2.3}$$

end for

2.2.2 Online gradient descent: special version of Online DCA

In this section, we will show that some variants of online gradient descent for online convex programming are special cases of Online DCA. First of all, in online convex programming, the loss function at the step t, denoted by $\tilde{g}_t$, is convex. In such a case,

the online gradient descent algorithm with greedy projection and with lazy projection, named OGD and OGD-L respectively, are described as follows [134].

OGD: Online Gradient Descent with Greedy projection

Initialization: let w^1 be an initial point in $\mathcal{S}$, $\{\eta_t\}$ be the sequence of step sizes

for $t = 1, 2, \ldots, T$ **do**

1. Compute $\tilde{s}^t \in \partial \tilde{g}_t(w^t)$.
2. Compute w^{t+1} by the update rule

$$w^{t+1} = \text{Proj}_{\mathcal{S}}(w^t - \eta_t \tilde{s}^t). \tag{2.4}$$

end for

OGD-L: Online Gradient Descent with Lazy projection

Initialization: let w^1 be an initial point in $\mathcal{S}$, $\{\eta_t\}$ be the sequence of step sizes, $y^1 = 0 \in \mathbb{R}^n$

for $t = 1, 2, \ldots, T$ **do**

1. Compute $\tilde{s}^t \in \partial \tilde{g}_t(w^t)$.
2. Compute w^{t+1} by the update rule

$$y^{t+1} = y^t - \eta_t \tilde{s}^t, \tag{2.5}$$
$$w^{t+1} = \text{Proj}_{\mathcal{S}}(y^{t+1}). \tag{2.6}$$

end for

Now, we are going to indicate that in some cases of DC functions, the corresponding Online DCA algorithm is exactly the same as the versions of online gradient descent algorithm, which is presented in Propositions 2.1, 2.2.

Proposition 2.1. *Assume that $R = 0$, $t_0 = t$ and at the step t, there exists a positive number ρ_t such that $\frac{\rho_t}{2}\|\cdot\|^2 - \tilde{g}_t$ is convex on $\mathcal{S}$. Then,* OGD *is a special case of* Online DCA.

Proof. At the step t, we derive the following DC decomposition of $\tilde{g}_t$:

$$\tilde{g}_t(w) = g_t(w) - h_t(w),$$

where $g_t(w) = \frac{\rho_t}{2}\|w\|^2$ and $h_t(w) = \frac{\rho_t}{2}\|w\|^2 - \tilde{g}_t(w)$. Thus, solving the subproblem (2.3) in Online DCA amounts to solving the problem

$$\min\left\{\frac{\rho_t}{2}\|w\|^2 - \left\langle \rho_t w^t - \tilde{s}^t, w\right\rangle : w \in \mathcal{S}\right\}.$$

It can be equivalently reformulated to

$$\min\left\{\left\|w - \left(w^t - \frac{1}{\rho_t}\tilde{s}^t\right)\right\|^2 : w \in \mathcal{S}\right\}.$$

Thus, we deduce that

$$w^{t+1} = \text{Proj}_{\mathcal{S}}\left(w^t - \frac{1}{\rho_t}\tilde{s}^t\right),$$

where $\text{Proj}_{\mathcal{S}}(x)$ is the projection of x on $\mathcal{S}$. It is exactly the same as the update rule (2.4) in OGD with step size of $1/\rho_t$. The proof is complete. □

Remark 2.1. *With the same assumptions in Proposition 2.1 but* $t_0 = 1$ *and* $\mathcal{S} = \mathbb{R}^n$, *we also indicate that* OGD *is a special case of* Online DCA. *In particular,* (2.3) *would be*

$$\min\left\{\sum_{i=1}^{t}\frac{\rho_i}{2}\|w\|^2 - \sum_{i=1}^{t}\left\langle \rho_i w^i - \tilde{s}^i, w\right\rangle : w \in \mathbb{R}^n\right\}$$

or, equivalently,

$$\min\left\{\left\|w - \frac{1}{\rho^{(t)}}\sum_{i=1}^{t}\left(\rho_i w^i - \tilde{s}^i\right)\right\|^2 : w \in \mathbb{R}^n\right\},$$

where $\rho^{(t)} := \sum_{i=1}^{t}\rho_i$, $\forall t$. *Thus, we obtain the following update rule*

$$w^{t+1} = \frac{1}{\rho^{(t)}}\sum_{i=1}^{t}\left(\rho_i w^i - \tilde{s}^i\right) = \frac{1}{\rho^{(t)}}\left[\left(\rho_t w^t - \tilde{s}^t\right) + \rho^{(t-1)}w^t\right] = w^t - \frac{1}{\rho^{(t)}}\tilde{s}^t.$$

Proposition 2.2. *Assume that* $t_0 = 1$, $R(w) = \frac{1}{2\eta}\|w\|_2^2$ $(\eta > 0)$ *and at the step* t, *the loss function is linear i.e.* $f_t(w) = \langle w, \tilde{s}^t\rangle$ *where* $\tilde{s}^t \in \partial\tilde{g}_t(w^t)$. *Then* OGD-L *is a special case of* Online DCA.

Proof. We can see that DC components of f_t are $g_t(w) = \langle w, \tilde{s}^t\rangle$ and $h_t(w) = 0$. According to Online DCA, (2.3) can be reformulated as

$$\min\left\{\frac{1}{2\eta}\|w\|^2 + \left\langle\sum_{i=1}^{t}\tilde{s}^i, w\right\rangle : w \in \mathcal{S}\right\}$$

or, equivalently,

$$\min\left\{\left\|w + \eta\sum_{i=1}^{t}\tilde{s}^i\right\|^2 : w \in \mathcal{S}\right\}.$$

Thus, we have

$$y^{t+1} = y^t - \eta\tilde{s}^t, \quad w^{t+1} = \text{Proj}_{\mathcal{S}}\left(y^{t+1}\right). \tag{2.7}$$

This update rule is the same as (2.5)–(2.6) in the algorithm OGD-L with step size of η. The proof is complete. □

Next, we are going to present a particular Online DCA based scheme which is very efficient in practice problems.

2.2.3 ODCA: a proposed Online DCA based scheme

According to Section 1.2.3, we propose a particular scheme based on Online DCA where the covex subproblem (2.3) is solved by approximating by one iteration of subgradient method (see e.g. [111] for the definition). In particular, applying one iteration of the projected subgradient method at the point w^t with step size of η_t, we derive from (1.13) the following update rule:

$$w^{t+1} \quad = \quad \text{Proj}_{\mathcal{S}}(w^t - \eta_t s^t), \tag{2.8}$$

where the vector $s^t \in \partial g_t(w^t) - z^t$. Thus, a particular Online DCA based scheme (ODCA) applied to (2.2) can be described as follows.

ODCA: Online DCA-projected Subgradient scheme

Initialization: let w^1 be an initial point, $\{\eta_t\}$ be the sequence of step sizes
for $t = 1, 2, \ldots, T$ **do**
1. Compute $z^t \in \partial h_t(w^t)$.
2. Compute $s^t \in (\partial g_t(w^t) - z^t)$.
3. Compute $w^{t+1} \in \mathcal{S}$ by the update rule

$$w^{t+1} = \text{Proj}_{\mathcal{S}}(w^t - \eta_t s^t). \tag{2.9}$$

end for

Remark 2.2. *It is worth noting that in the* Online DCA *scheme, the problem* (1.12) *can be solved approximating by k iterations of subgradient method at each step. In such a case, the quality of the solution* w^{t+1} *to the problem* (1.12) *may be improved, however one possible disadvantage is the expensive computation time. This chapter will address the case of one iteration. Studying the effect of the number of iterations of subgradient method on the efficiency of* ODCA *algorithms will be a part of future works.*

2.2.4 Analysis of ODCA

In this section, we will present the regret of the sequence $\{w^t\}$ generated by ODCA. Recall the regret of the algorithm A is defined by

$$\mathsf{Regret}_{\mathrm{A}}^{T} := \sum_{t=1}^{T} f_t(w^t) - \min_{w \in \mathcal{S}} \sum_{t=1}^{T} f_t(w). \tag{2.10}$$

First we make four necessary assumptions below and then we indicate the regret bound of ODCA, which is stated in Theorem 2.1 and Theorem 2.2.

Assumption 2.1. *There exists a vector* $u^* \in \mathcal{S}$ *such that for* $t = 1, \ldots, T$,

$$u^* \in \arg\min_{w \in \mathcal{S}} f_t(w). \tag{2.11}$$

Assumption 2.2. *There exists a positive parameter α such that at the step t,*

$$\frac{\alpha}{2}\|u^* - w^t\|^2 \leq g_t(w^t) - g_t(u^*) - \langle z^t, w^t - u^*\rangle. \tag{2.12}$$

Assumption 2.3. *There exists a nonnegative parameter β such that at the step t,*

$$h_t(u^*) - h_t(w^t) - \langle z^t, u^* - w^t\rangle \leq \frac{\beta}{2}\|u^* - w^t\|^2. \tag{2.13}$$

Assumption 2.4. *There exists a positive parameter γ such that at the step t,*

$$g_t(w^t) - g_t(u^*) \leq \langle r^t, w^t - u^*\rangle - \frac{\gamma}{2}\|u^* - w^t\|^2. \tag{2.14}$$

where $r^t \in \partial g_t(w^t)$.

Suppose that L and D are respectively positive upper bounds of the sequence $\{s^t\}_{t=1,\ldots,T}$ and $\{w^t - u^*\}_{t=1,\ldots,T}$ that is

$$\max_{t\in\{1,2,\ldots,T\}} \|s^t\| \leq L \text{ and } \max_{t\in\{1,2,\ldots,T\}} \|w^t - u^*\| \leq D. \tag{2.15}$$

Let us define the convex function

$$\overline{g}_t(w) := g_t(w) - \langle z^t, w\rangle$$

for $t = 1, 2, \ldots, T$.

Theorem 2.1. *Suppose that* ODCA *generates the sequence $\{w^t\}_{t=1,\ldots,T}$ and Assumptions 2.1, 2.2, 2.3 are verified. Then, we have*

$$\mathsf{Regret}^T_{\mathrm{ODCA}} \leq \frac{3DL(\alpha+\beta)\sqrt{T}}{2\alpha}.$$

Proof. From the definition of $\mathsf{Regret}^T_{\mathrm{A}}$ (2.10), we have

$$\mathsf{Regret}^T_{\mathrm{ODCA}} = \sum_{t=1}^{T} f_t(w^t) - \min_{w\in\mathcal{S}} \sum_{t=1}^{T} f_t(w) \leq \sum_{t=1}^{T} \left[f_t(w^t) - \min_{w\in\mathcal{S}} f_t(w) \right]. \tag{2.16}$$

It derives from Assumption 2.1 that

$$\begin{aligned} & f_t(w^t) - \min_{w\in\mathcal{S}} f_t(w) \\ = \; & f_t(w^t) - f_t(u^*) \\ = \; & [g_t(w^t) - g_t(u^*)] + [h_t(u^*) - h_t(w^t)] \\ = \; & [\overline{g}_t(w^t) - \overline{g}_t(u^*)] + [h_t(u^*) - h_t(w^t) - \langle z^t, u^* - w^t\rangle]. \end{aligned} \tag{2.17}$$

From (2.16), (2.17) and Assumptions 2.2, 2.3, we obtain

$$\mathsf{Regret}^T_{\mathrm{ODCA}} \;\leq\; \left(1 + \frac{\beta}{\alpha}\right) \sum_{t=1}^{T} [\overline{g}_t(w^t) - \overline{g}_t(u^*)]. \tag{2.18}$$

Since $s^t \in \partial \overline{g}_t(w^t)$, it follows

$$\mathsf{Regret}^T_{\mathrm{ODCA}} \leq \left(1 + \frac{\beta}{\alpha}\right) \sum_{t=1}^{T} \langle s^t, w^t - u^* \rangle. \tag{2.19}$$

Similarly to the proof of Theorem 3.1 in [45], we find an upper bound of $\langle s^t, w^t - u^* \rangle$, $t \in \{1, \ldots, T\}$. In particular, by the definition of the sequence $\{w^t\}_{t=1,\ldots,T}$ as (2.8), we have

$$\begin{aligned} \|w^{t+1} - u^*\|^2 &\leq \|(w^t - \eta_t s^t) - u^*\|^2 \\ &= \|(w^t - u^*) - \eta_t s^t\|^2 \\ &= \|w^t - u^*\|^2 - 2\eta_t \langle s^t, w^t - u^* \rangle + \eta_t^2 \|s^t\|^2. \end{aligned}$$

Thus,

$$\langle s^t, w^t - u^* \rangle \leq \frac{\|w^t - u^*\|^2 - \|w^{t+1} - u^*\|^2}{2\eta_t} + \frac{\eta_t}{2} \|s^t\|^2. \tag{2.20}$$

It derives that

$$\begin{aligned} \mathsf{Regret}^T_{\mathrm{ODCA}} &\leq \left(1 + \frac{\beta}{\alpha}\right) \left\{ \sum_{t=1}^{T} \left[\|w^t - u^*\|^2 \left(\frac{1}{2\eta_t} - \frac{1}{2\eta_{t-1}} \right) \right] + \frac{L^2}{2} \sum_{t=1}^{T} \eta_t \right\} \\ &\leq \left(1 + \frac{\beta}{\alpha}\right) \left[\frac{D^2}{2} \sum_{t=1}^{T} \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}} \right) + \frac{L^2}{2} \sum_{t=1}^{T} \eta_t \right] \\ &\leq \left(1 + \frac{\beta}{\alpha}\right) \left(\frac{D^2}{2\eta_T} - \frac{D^2}{2\eta_0} + \frac{L^2}{2} \sum_{t=1}^{T} \eta_t \right). \end{aligned}$$

Let us define $\frac{1}{\eta_0} := 0$ and $\eta_t = \frac{D}{L\sqrt{t}}$ for all $t = 1, \ldots, T$. Then, we have

$$\mathsf{Regret}^T_{\mathrm{ODCA}} \leq \left(1 + \frac{\beta}{\alpha}\right) \left[\frac{DL\sqrt{T}}{2} + \frac{DL}{2} \left(2\sqrt{T} - 1\right) \right] \leq \frac{3DL(\alpha + \beta)\sqrt{T}}{2\alpha}.$$

The proof is complete. □

From Theorem 2.1, we can say that $\mathsf{Regret}^T_{\mathrm{ODCA}}$ grows sublinearly with the number of steps T, i.e. $\lim_{T \to +\infty} \mathsf{Regret}^T_{\mathrm{ODCA}}/T = 0$. However we can achieve a better regret bound, specifically *logarithmic regret* – $\mathcal{O}(\log(T))$ if Assumption 2.4 is also verified, which is stated in the following theorem.

Theorem 2.2. *Suppose that* ODCA *generates the sequence* $\{w^t\}_{t=1,\ldots,T}$ *and Assumptions 2.1, 2.2, 2.3 and 2.4 are verified. Then, we have*

$$\mathsf{Regret}^T_{\mathrm{ODCA}} \leq \frac{L^2(\alpha + \beta)\,(1 + \log(T))}{2\alpha\gamma}.$$

Proof. From (2.18) in Theorem 2.1 and Assumption 2.4, we derive that

$$\mathsf{Regret}^T_{\mathrm{ODCA}} \leq \left(1+\frac{\beta}{\alpha}\right)\sum_{t=1}^{T}\left[\langle s^t, w^t - u^*\rangle - \frac{\gamma}{2}\|u^* - w^t\|^2\right]. \tag{2.21}$$

Similarly to the proof of Theorem 3.3 in [45], let us define $\dfrac{1}{\eta_0} := 0$ and $\eta_t = \dfrac{1}{\gamma t}$ for all $t = 1, \ldots, T$. From (2.20) and (2.21), we have

$$\begin{aligned}
\mathsf{Regret}^T_{\mathrm{ODCA}} &\leq \left(1+\frac{\beta}{\alpha}\right)\left\{\sum_{t=1}^{T}\left[\|w^t - u^*\|^2 \underbrace{\left(\frac{1}{2\eta_t} - \frac{1}{2\eta_{t-1}} - \frac{\gamma}{2}\right)}_{=0}\right] + \frac{L^2}{2}\sum_{t=1}^{T}\eta_t\right\} \\
&\leq \left(1+\frac{\beta}{\alpha}\right)\frac{L^2}{2\gamma}\sum_{t=1}^{T}\frac{1}{t} \\
&\leq \frac{L^2(\alpha+\beta)\,(1+\log(T))}{2\alpha\gamma}.
\end{aligned}$$

The proof is complete. □

Remark 2.3. *(Comments about Assumptions 2.1–2.4)*

First of all, Assumption 2.1 can be replaced by the following assumption – that is, there exists a vector u^* such that

$$u^* \in \arg\min_{w\in\mathcal{S}} \sum_{t=1}^{T} f_t(w).$$

In this case, the inequality (2.18) *is guaranteed and we still yield the same regret bound. In practice problems, Assumption 2.1 is verified more easily.*

Next, Assumption 2.2 can be deduced from the inequality

$$\frac{\mu}{2}\|u^* - w^t\|^2 \leq f_t(w^t) - f_t(u^*)$$

where $\mu > \beta$ and β is defined in Assumption 2.3. Indeed, we have

$$\begin{aligned}
\overline{g}_t(w^t) - \overline{g}_t(u^*) &= f_t(w^t) - f_t(u^*) - \left[h_t(u^*) - h_t(w^t) - \langle z^t, u^* - w^t\rangle\right] \\
&\geq \frac{\alpha}{2}\|u^* - w^t\|^2,
\end{aligned}$$

$\alpha = \mu - \beta > 0$. Moreover, it is easy to verify Assumption 2.3 if h_t is differentiable with β-Lipschitz gradient on $\mathcal{S}$. Meanwhile, Assumption 2.4 is satisfied if g_t is γ-convex on $\mathcal{S}$ ($\gamma > 0$) i.e. $g_t - (\gamma/2)\|\cdot\|^2$ is convex (see e.g. [95]).

In the sequel, we are going to present how to employ ODCA for solving the problems in the topic of online classification, specifically online binary linear classification.

2.3 Online DCA for Online Binary Linear Classification problems

In this section, we focus on handling the problems in the topic of online binary linear classification (OBLC) where the set of instances is $\mathcal{X} = \mathbb{R}^n$, the set of linear classifiers is $\mathcal{S} = \mathbb{R}^n$ and the set of binary labels is $\mathcal{Y} = \{-1, 1\}$. Formally, at the step t, after receiving an instance $x_t \in \mathcal{X}$, the learner will find a linear classifier $w^t \in \mathcal{S}$ and then predict the corresponding binary label $\overline{p}_t = p_t(w^t) \in \mathcal{Y}$ where the function p_t is defined by

$$p_t(w) := \mathsf{sign}(\langle w, x_t\rangle) = \begin{cases} 1 & \text{if } \langle w, x_t\rangle \geq 0, \\ -1 & \text{otherwise.} \end{cases} \tag{2.22}$$

After that, the correct label $y_t \in \mathcal{Y}$ is revealed. In such a case, the quality of predictions is assessed by a loss function, specifically 0-1 loss function, which measures the difference between the predicted label and the correct one. The 0-1 loss function, denoted by ℓ_t, is defined as

$$\ell_t(p_t(w), y_t) := \mathbb{1}_{\{p_t(w) \neq y_t\}}(w) = \mathbb{1}_{\{y_t\langle w, x_t\rangle \leq 0\}}(w) = \begin{cases} 1 & \text{if } y_t\langle w, x_t\rangle \leq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2.23}$$

where $\mathbb{1}_X$ is the indicator function of X. Obviously, this loss function is nonconvex, nonsmooth.

There exist many online classification algorithms with different approaches such as Perceptron [91, 102, 121], Approximate Maximal Margin Classification Algorithm (ALMA) [39], Relaxed Online Maximum Margin Algorithm (ROMMA) [76], Passive-Aggressive learning algorithms (PA) [29] and their variants. Hoi et al. [47] conducted a library of scalable and efficient online learning algorithms for large-scale online classification tasks. As the first works, [46] developed an online classification technique based on an online version of DCA where a DC function was explored to approximate the nonconvex, nonsmooth 0-1 loss function. Our works will thoroughly study more efficient OBLC algorithms with formal regret bounds.

First of all, we propose three different DC approximation functions f_t on $\mathcal{S}$ instead of the 0-1 loss function ℓ_t. Let us denote by $\mathcal{M}_{\mathrm{A}}$ the set of steps where the prediction of the algorithm A may be wrong in the sense that $f_t(w^t) > 0$. If a step $t \notin \mathcal{M}_{\mathrm{A}}$, then we observe the function $f_t(w) = 0$, $\forall w \in \mathbb{R}^n$. Otherwise, we define the DC approximation function f_t satisfying the following condition

$$f_t(w^t) \geq \ell_t(\overline{p}_t, y_t). \tag{2.24}$$

In particular, we suggest three DC functions: two piecewise linear functions (as in e.g. [46]) similar to ramp loss [27, 28] and one sigmoid function (see e.g. [83]). In the following, we will present how to design online algorithms based on ODCA for these functions.

2.3.1 First piecewise linear function

At step t, we observe the function $f_t^{(1)} : \mathbb{R}^n \to [0,1]$ defined as follows [46]:

$$f_t^{(1)}(w) := \max\left\{0, \min\left\{\frac{\tau_{2t}}{\tau_{1t}}, \frac{-y_t\langle w, x_t\rangle}{\tau_{1t}}\right\}\right\} \tag{2.25}$$

where τ_{2t}, τ_{1t} are two positive parameters. It is evident that if $\tau_{2t} = \tau_{1t}$, then for any $w \in \mathbb{R}^n$, $0 \leq f_t^{(1)}(w) \leq \ell_t(w) \leq 1$ and $f_t^{(1)}(w) \to \ell_t(w)$ as $\tau_{1t} \to 0$. Moreover, DC components of $f_t^{(1)}$ are

$$g_t^{(1)}(w) = \max\left\{0, \frac{-y_t\langle w, x_t\rangle}{\tau_{1t}}\right\} \text{ and } h_t^{(1)}(w) = \max\left\{0, \frac{-\tau_{2t} - y_t\langle w, x_t\rangle}{\tau_{1t}}\right\}.$$

Remark that we have to choose the parameters τ_{1t}, τ_{2t} such that (2.24) is satisfied. However, it is never true if there exists a step t such that $y_t\langle w^t, x_t\rangle = 0$ since $\ell_t(\overline{p}_t, y_t) = 1$ and $f_t^{(1)}(w^t) = 0$. In such a case, we investigate another DC function, namely $f_t^{(2)}$, defined by

$$f_t^{(2)}(w) := \max\left\{0, \min\left\{1, \frac{\tau_{1t} - y_t\langle w, x_t\rangle}{\tau_{1t}}\right\}\right\}, \tag{2.26}$$

and its DC decomposition is

$$f_t^{(2)}(w) = g_t^{(2)}(w) - g_t^{(1)}(w),$$

where

$$g_t^{(2)}(w) = \max\left\{0, \frac{\tau_{1t} - y_t\langle w, x_t\rangle}{\tau_{1t}}\right\}.$$

Obviously, we have $f_t^{(2)}(w^t) = 1 = \ell_t(\overline{p}_t, y_t)$.

For these DC functions, let us define the set

$$\begin{aligned} \mathcal{M}_{\text{PiL1}} &:= \{t \in \{1,\ldots,T\} : f_t^{(1)}(w^t) > 0 \text{ or } y_t\langle w^t, x_t\rangle = 0\} \\ &= \{t \in \{1,\ldots,T\} : y_t\langle w^t, x_t\rangle \leq 0\} \end{aligned} \tag{2.27}$$

and the set

$$\mathcal{N}_{\text{PiL1}} := \{t \in \mathcal{M}_{\text{PiL1}} : y_t\langle w^t, x_t\rangle < 0\}.$$

According to the ODCA algorithm, at step t, we compute first the vector $z^t \in \partial h_t^{(1)}(w^t)$ (resp. $z^t \in \partial g_t^{(1)}(w^t)$), then the vector $s^t \in (\partial g_t^{(1)}(w^t) - z^t)$ (resp. $s^t \in (\partial g_t^{(2)}(w^t) - z_t)$) when $t \in \mathcal{N}_{\text{PiL1}}$ (resp. $t \in \mathcal{M}_{\text{PiL1}} \setminus \mathcal{N}_{\text{PiL1}}$) and then the vector w^{t+1} using (2.8).

■ Compute $\partial g_t^{(1)}$, $\partial h_t^{(1)}$ and $\partial g_t^{(2)}$: by the definition of $g_t^{(1)}$, $g_t^{(2)}$ and $h_t^{(1)}$, we have

$$\partial g_t^{(1)}(w) = \begin{cases} \{0\} & \text{if } y_t\langle w, x_t\rangle > 0, \\ \left[\frac{-y_t x_t}{\tau_{1t}}, 0\right] & \text{if } y_t\langle w, x_t\rangle = 0, \\ \left\{\frac{-y_t x_t}{\tau_{1t}}\right\} & \text{if } y_t\langle w, x_t\rangle < 0, \end{cases} \tag{2.28}$$

$$\partial h_t^{(1)}(w) = \begin{cases} \{0\} & \text{if } y_t\langle w, x_t\rangle > -\tau_{2t}, \\ \left[\dfrac{-y_t x_t}{\tau_{1t}}, 0\right] & \text{if } y_t\langle w, x_t\rangle = -\tau_{2t}, \\ \left\{\dfrac{-y_t x_t}{\tau_{1t}}\right\} & \text{if } y_t\langle w, x_t\rangle < -\tau_{2t}, \end{cases} \tag{2.29}$$

and

$$\partial g_t^{(2)}(w) = \begin{cases} \{0\} & \text{if } y_t\langle w, x_t\rangle > \tau_{1t}, \\ \left[\dfrac{-y_t x_t}{\tau_{1t}}, 0\right] & \text{if } y_t\langle w, x_t\rangle = \tau_{1t}, \\ \left\{\dfrac{-y_t x_t}{\tau_{1t}}\right\} & \text{if } y_t\langle w, x_t\rangle < \tau_{1t}. \end{cases} \tag{2.30}$$

Here $[a, b]$ is the line segment between a and b. When $t \in \mathcal{N}_{\text{PiL1}}$, we choose $z^t \in \partial h_t^{(1)}(w^t)$, $r^t \in \partial g_t^{(1)}(w^t)$ and $s^t = r^t - z^t \in (\partial g_t^{(1)}(w^t) - z_t)$ as follows:

$$r^t = \frac{-y_t x_t}{\tau_{1t}}, z^t = \begin{cases} \dfrac{-y_t x_t}{\tau_{1t}} & \text{if } y_t\langle w^t, x_t\rangle < -\tau_{2t}, \\ 0 & \text{if } -\tau_{2t} \le y_t\langle w^t, x_t\rangle < 0, \end{cases}$$

and

$$s^t = \begin{cases} 0 & \text{if } y_t\langle w^t, x_t\rangle < -\tau_{2t}, \\ \dfrac{-y_t x_t}{\tau_{1t}} & \text{if } -\tau_{2t} \le y_t\langle w^t, x_t\rangle < 0. \end{cases}$$

Similarly, when $t \in \mathcal{M}_{\text{PiL1}} \setminus \mathcal{N}_{\text{PiL1}}$, we have

$$r^t = \frac{-y_t x_t}{\tau_{1t}}, z^t = 0 \text{ and } s^t = \frac{-y_t x_t}{\tau_{1t}}.$$

■ Choose the parameters τ_{1t}, τ_{2t}: in order to satisfy (2.24), for any $t \in \mathcal{M}_{\text{PiL1}}$, we propose the following choices

$$\tau_{1t} = \begin{cases} \min\{\tau_1, -y_t\langle w^t, x_t\rangle\} & \text{if } t \in \mathcal{N}_{\text{PiL1}}, \\ \tau_1 & \text{if } t \in \mathcal{M}_{\text{PiL1}} \setminus \mathcal{N}_{\text{PiL1}}, \end{cases}$$

where τ_1 is a positive tuning parameter, and

$$\tau_{2t} = -y_t\langle w^t, x_t\rangle. \tag{2.31}$$

■ Choose the learning rate η_t: we consider the constant learning rate that is $\eta_t = \eta$ for all t. In this case, the prediction labels defined as (2.22) and the set $\mathcal{M}_{\text{PiL1}}$ is completely independent of the value of η (see [109] for more details). Consequently, we can choose $\eta_t = 1$ for all t.

Finally, ODCA with the first piecewise linear function is given by Algorithm 2.1 (ODCA-PiL1).

Algorithm 2.1 ODCA with first piecewise linear function (ODCA-PiL1)

Initialization: let τ_1 be a positive number, w^1 be an initial point
for $t = 1, 2, \ldots, T$ **do**
  if $-\tau_1 \leq y_t\langle w^t, x_t\rangle < 0$ **then**
    $w^{t+1} = w^t - \dfrac{x_t}{\langle w^t, x_t\rangle}$.
  else if $y_t\langle w^t, x_t\rangle = 0$ or $y_t\langle w^t, x_t\rangle < -\tau_1$ **then**
    $w^{t+1} = w^t + \dfrac{y_t x_t}{\tau_1}$.
  else
    $w^{t+1} = w^t$.
  end if
end for

According to the analysis of ODCA in Section 2.2.4, we archive the logarithmic regret of the ODCA-PiL1 algorithm as stated in Theorem 2.2 for this nonconvex piecewise linear function. In order to get this result, we need to indicate that all four Assumptions 2.1–2.4 are satisfied for two cases: $t \in \mathcal{M}_{\text{PiL1}} \setminus \mathcal{N}_{\text{PiL1}}$ and $t \in \mathcal{N}_{\text{PiL1}}$ as the following lemmas. Throughout these lemmas, we further assume that at the step $t \in \mathcal{M}_{\text{PiL1}}$, $x_t \neq 0$. When $x_t = 0$, the algorithm makes no update, which still holds for the update step in Algorithm 2.1 (corresponding to $w^{t+1} = w^t$). We also assume that there exists a vector $u^* \in \mathbb{R}^n$ such that for all $t = 1, \ldots, T$,

$$y_t\langle u^*, x_t\rangle \geq 2\tau_1. \tag{2.32}$$

Lemma 2.1. *For the DC function* (2.25) *and step* $t \in \mathcal{N}_{\text{PiL1}}$*, there exist the parameters* α*,* γ *and the vector* u^* *such that Assumptions 2.1, 2.2, 2.4 are satisfied. Moreover, Assumption 2.3 is satisfied for all* $\beta \geq 0$*.*

Proof. As for $t \in \mathcal{N}_{\text{PiL1}}$, from (2.32), it is easy to check that Assumption 2.1 is satisfied. Moreover, it is worth noting that $u^* \neq w^t$ for all $t = 1, \ldots, T$. Indeed, assume the contrary that is $u^* = w^t$ for some $t \in \mathcal{N}_{\text{PiL1}}$, so $y_t\langle u^*, x_t\rangle < 0$ contradicting (2.32).

Now, we will show that Assumptions 2.2–2.4 can be satisfied. In particular, as for Assumption 2.2, let us define the function $\overline{g}_t^{(1)} := g_t^{(1)} - \langle z^t, \cdot\rangle$ and we have

$$\overline{g}_t^{(1)}(w^t) - \overline{g}_t^{(1)}(u^*) = -\frac{y_t\langle w^t, x_t\rangle}{\tau_{1t}} > 0.$$

Thus, the inequality (2.12) is satisfied with $\alpha \leq \min\limits_{t \in \mathcal{N}_{\text{PiL1}}} \dfrac{-2y_t\langle w^t, x_t\rangle}{\tau_{1t}\|u^* - w^t\|^2}$.

Regarding Assumption 2.3, from (2.31) and the definition of $h_t^{(1)}$, we have that for any $t \in \mathcal{N}_{\text{PiL1}}$ and $\beta \geq 0$,

$$h_t^{(1)}(u^*) - h_t^{(1)}(w^t) - \langle z^t, u^* - w^t\rangle = 0 \leq \frac{\beta}{2}\|u^* - w^t\|^2.$$

Concerning Assumption 2.4, we have that for any $t \in \mathcal{N}_{\mathrm{PiL1}}$,

$$\begin{aligned} g_t^{(1)}(u^*) - g_t^{(1)}(w^t) - \langle r^t, u^* - w^t \rangle &= \frac{y_t \langle w^t, x_t \rangle}{\tau_{1t}} - \langle \frac{-y_t x_t}{\tau_{1t}}, u^* - w^t \rangle \\ &= \frac{y_t \langle u^*, x_t \rangle}{\tau_{1t}} > 0. \end{aligned}$$

Thus, the inequality (2.14) is satisfied with $\gamma \leq \min_{t \in \mathcal{N}_{\mathrm{PiL1}}} \frac{2 y_t \langle u^*, x_t \rangle}{\tau_{1t} \|u^* - w^t\|^2}$. The proof is complete. □

Lemma 2.2. *For step $t \in \mathcal{M}_{\mathrm{PiL1}} \backslash \mathcal{N}_{\mathrm{PiL1}}$ and the loss function* (2.26)*, Assumptions 2.1, 2.2, 2.4 are satisfied with the suitable parameters α, γ and vector u^*. Moreover, Assumption 2.3 is satisfied for all $\beta \geq 0$.*

Proof. In case $t \in \mathcal{M}_{\mathrm{PiL1}} \setminus \mathcal{N}_{\mathrm{PiL1}}$ (that is $y_t \langle w^t, x_t \rangle = 0$), we can readily verify Assumption 2.1 by (2.32). We can see that if $t \in \mathcal{M}_{\mathrm{PiL1}} \setminus \mathcal{N}_{\mathrm{PiL1}}$, then $u^* \neq w^t$, since $y_t \langle w^t, x_t \rangle = 0$ while $y_t \langle u^*, x_t \rangle \geq 2\tau_1 > 0$.

We are going to show that Assumptions 2.2–2.4 are also satisfied. In particular, as for Assumption 2.2, let us define the function $\overline{g}_t^{(2)} := g_t^{(2)} - \langle z^t, \cdot \rangle$ for $t \in \mathcal{M}_{\mathrm{PiL1}} \setminus \mathcal{N}_{\mathrm{PiL1}}$ and we have

$$\overline{g}_t^{(2)}(w^t) - \overline{g}_t^{(2)}(u^*) = 1 \geq \frac{\alpha}{2} \|u^* - w^t\|^2,$$

where $\alpha \leq \min_{t \in \mathcal{M}_{\mathrm{PiL1}} \backslash \mathcal{N}_{\mathrm{PiL1}}} \frac{2}{\|u^* - w^t\|^2}$.

Regarding Assumption 2.3, we have that for any $t \in \mathcal{M}_{\mathrm{PiL1}} \setminus \mathcal{N}_{\mathrm{PiL1}}$ and $\beta \geq 0$,

$$g_t^{(1)}(u^*) - g_t^{(1)}(w^t) - \langle z^t, u^* - w^t \rangle = 0 \leq \frac{\beta}{2} \|u^* - w^t\|^2.$$

Concerning Assumption 2.4, we have that for any $t \in \mathcal{M}_{\mathrm{PiL1}} \setminus \mathcal{N}_{\mathrm{PiL1}}$,

$$\begin{aligned} g_t^{(2)}(u^*) - g_t^{(2)}(w^t) - \langle r^t, u^* - w^t \rangle &= -1 - \langle \frac{-y_t x_t}{\tau_{1t}}, u^* - w^t \rangle \\ &= \frac{y_t \langle u^*, x_t \rangle}{\tau_{1t}} - 1 \\ &\geq \frac{\gamma}{2} \|u^* - w^t\|^2, \end{aligned}$$

where $\gamma \leq \min_{t \in \mathcal{M}_{\mathrm{PiL1}} \backslash \mathcal{N}_{\mathrm{PiL1}}} \frac{2}{\|u^* - w^t\|^2}$.

The proof is complete. □

Consequently, we derive the following corollary for the regret bound of the ODCA-PiL1 algorithm (whose proof is straightforward with the choice $\beta = \alpha$).

Corollary 2.1. *Assume that* ODCA-PiL1 *generates the sequence* $\{w^t\}_{t=1,\ldots,T}$. *Then, we have*

$$\mathsf{Regret}^T_{\text{ODCA-PiL1}} \leq \frac{L^2\left(1+\log(T)\right)}{\gamma} \tag{2.33}$$

where L *is defined as* (2.15) *and the positive parameter*

$$\gamma \leq \min_{t\in\mathcal{M}_{\text{PiL1}}} \frac{2\min\left\{\tau_{1t}, \psi(y_t\langle u^*, x_t\rangle)\right\}}{\tau_{1t}\|u^* - w^t\|^2},$$

the real-value function $\psi(x) = x$ *if* $x > 0$, $+\infty$ *otherwise.*

The following theorem provides a bound on the number of prediction mistakes (which is called a *mistake bound*, for short) for the ODCA-PiL1 algorithm, which is defined as the bound on the number of steps where $\overline{p}_t \neq y_t$.

Theorem 2.3. *For any* $w \in \mathbb{R}^n$, *the number of prediction mistakes made by* ODCA-PiL1 *has an upper bound that is the root,* $\overline{x}_1$, *of the equation*

$$x - \overline{a}_{\text{PiL1}} - \overline{b}_{\text{PiL1}}\left(1+\log(x)\right) = 0$$

where $\overline{a}_{\text{PiL1}} = \sum\limits_{t\in\mathcal{N}_{\text{PiL1}}} f_t^{(1)}(w) + \sum\limits_{t\in\mathcal{M}_{\text{PiL1}}\setminus\mathcal{N}_{\text{PiL1}}} f_t^{(2)}(w)$, $\overline{b}_{\text{PiL1}} = L^2/\gamma_{\text{PiL1}}$, $\overline{x}_1 \geq \overline{b}_{\text{PiL1}}$, *the positive parameter* $\gamma_{\text{PiL1}} \leq \min\{\gamma, L^2\}$.

Proof. From the inequality (2.24), Corollary 2.1 and the definition of γ_{PiL1}, we derive that for any $w \in \mathbb{R}^n$,

$$\begin{aligned} |\mathcal{M}_{\text{PiL1}}| &\leq \sum_{t\in\mathcal{N}_{\text{PiL1}}} f_t^{(1)}(w^t) + \sum_{t\in\mathcal{M}_{\text{PiL1}}\setminus\mathcal{N}_{\text{PiL1}}} f_t^{(2)}(w^t) \\ &\leq \sum_{t\in\mathcal{N}_{\text{PiL1}}} f_t^{(1)}(w) + \sum_{t\in\mathcal{M}_{\text{PiL1}}\setminus\mathcal{N}_{\text{PiL1}}} f_t^{(2)}(w) + \frac{L^2\left(1+\log(|\mathcal{M}_{\text{PiL1}}|)\right)}{\gamma_{\text{PiL1}}} \end{aligned} \tag{2.34}$$

where $|\mathcal{M}_{\text{PiL1}}|$ is the number of steps of the set $\mathcal{M}_{\text{PiL1}}$.

Using the definition of $\overline{a}_{\text{PiL1}}$ and $\overline{b}_{\text{PiL1}}$, it is evident that $\overline{a}_{\text{PiL1}} \geq 0$, $\overline{b}_{\text{PiL1}} \geq 1$ and the inequality (2.34) can be rewritten as follows.

$$|\mathcal{M}_{\text{PiL1}}| \leq \overline{a}_{\text{PiL1}} + \overline{b}_{\text{PiL1}}\left(1+\log(|\mathcal{M}_{\text{PiL1}}|)\right).$$

The real function $r_{\text{PiL1}} : (0,+\infty) \to \mathbb{R}$, $r_{\text{PiL1}}(x) = x - \overline{a}_{\text{PiL1}} - \overline{b}_{\text{PiL1}}\left(1+\log(x)\right)$ is strictly convex on $(0,+\infty)$. Since $\lim\limits_{x\to 0^+} r_{\text{PiL1}}(x) = \lim\limits_{x\to+\infty} r_{\text{PiL1}}(x) = +\infty$ and $r_{\text{PiL1}}(\overline{b}_{\text{PiL1}}) \leq 0$, the equation $r_{\text{PiL1}}(x) = 0$ has two roots $\overline{x}_1$, $\overline{x}_2$ such that $0 < \overline{x}_2 \leq \overline{b}_{\text{PiL1}} \leq \overline{x}_1$. The proof is complete. □

2.3.2 Second piecewise linear function

In this section, we propose another DC approximation function which is also of form of piecewise linear function. Specifically, at step t, we observe the function $f_t^{(3)} : \mathbb{R}^n \to [0,1]$ defined as follows:

$$f_t^{(3)}(w) := \begin{cases} \max\left\{0, \min\left\{1 + \dfrac{\tau_{2t}}{\tau_{1t}}, 1 - \dfrac{y_t\langle w, x_t\rangle}{\tau_{1t}}\right\}\right\} & \text{if } y_t\langle w^t, x_t\rangle \geq -\tau_{2t}, \\ 1 & \text{otherwise,} \end{cases} \tag{2.35}$$

where τ_{2t}, τ_{1t} are two positive parameters. Obviously, (2.24) is always true for the function $f_t^{(3)}$. The set $\mathcal{M}_{\text{PiL2}}$ is defined by

$$\begin{aligned} \mathcal{M}_{\text{PiL2}} &:= \{t \in \{1,\ldots,T\} : f_t^{(3)}(w^t) > 0\} \\ &= \{t \in \{1,\ldots,T\} : y_t\langle w^t, x_t\rangle < \tau_{1t}\} \end{aligned} \tag{2.36}$$

and the set

$$\mathcal{N}_{\text{PiL2}} := \{t \in \{1,\ldots,T\} : -\tau_{2t} \leq y_t\langle w^t, x_t\rangle < \tau_{1t}\}.$$

For $t \in \mathcal{M}_{\text{PiL2}}$, $f_t^{(3)}$ is a DC function whose DC decomposition is

$$f_t^{(3)}(w) = g_t^{(3)}(w) - h_t^{(3)}(w), w \in \mathbb{R}^n,$$

where

$$g_t^{(3)}(w) := \begin{cases} g_t^{(2)}(w) & \text{if } t \in \mathcal{N}_{\text{PiL2}}, \\ 1 & \text{if } t \in \mathcal{M}_{\text{PiL2}} \setminus \mathcal{N}_{\text{PiL2}}, \end{cases}$$

and

$$h_t^{(3)}(w) := \begin{cases} h_t^{(1)}(w) & \text{if } t \in \mathcal{N}_{\text{PiL2}}, \\ 0 & \text{if } t \in \mathcal{M}_{\text{PiL2}} \setminus \mathcal{N}_{\text{PiL2}}. \end{cases}$$

Applying ODCA with this loss function leads us at step $t \in \mathcal{N}_{\text{PiL2}}$ to compute $z^t \in \partial h_t^{(3)}(w^t)$, $s^t \in (\partial g_t^{(3)}(w^t) - z^t)$ and w^{t+1} based on (2.8).

■ Compute $\partial g_t^{(3)}$, $\partial h_t^{(3)}$: from (2.29) and (2.30), for $t \in \mathcal{M}_{\text{PiL2}}$, we choose $z^t \in \partial h_t^{(3)}(w^t)$, $r^t \in \partial g_t^{(3)}(w^t)$ and $s^t = r^t - z^t \in (\partial g_t^{(3)}(w^t) - z^t)$ as follows: $z^t = 0$,

$$r^t = \begin{cases} \dfrac{-y_t x_t}{\tau_{1t}} & \text{if } t \in \mathcal{N}_{\text{PiL2}}, \\ 0 & \text{if } t \in \mathcal{M}_{\text{PiL2}} \setminus \mathcal{N}_{\text{PiL2}}, \end{cases}$$

and

$$s^t = \begin{cases} \dfrac{-y_t x_t}{\tau_{1t}} & \text{if } t \in \mathcal{N}_{\text{PiL2}}, \\ 0 & \text{if } t \in \mathcal{M}_{\text{PiL2}} \setminus \mathcal{N}_{\text{PiL2}}. \end{cases}$$

■ Choose the parameters τ_{1t}, τ_{2t} and learning rate η_t: we propose the following choice: $\tau_{1t} = \tau_1\|x_t\|$, $\tau_{2t} = \tau_2\|x_t\|$ and $\eta_t = C/\sqrt{t}$ for all t where τ_1, τ_2 and C are positive tuning parameters.

ODCA applied to OBLC problems with this second piecewise linear loss function can be summarized in Algorithm 2.2 (ODCA-PiL2).

Algorithm 2.2 ODCA with second piecewise linear function (ODCA-PiL2)

Initialization: let w^1 be an initial point, $\{\eta_t\}$ be a sequence of learning rates, and $\{\tau_{1t}\}$, $\{\tau_{2t}\}$ be a sequence of positive parameters mentioned above
for $t = 1, 2, \ldots, T$ **do**
  if $-\tau_{2t} \leq y_t\langle w^t, x_t\rangle < \tau_{1t}$ **then**
    $w^{t+1} = w^t + \eta_t \dfrac{y_t x_t}{\tau_{1t}}$.
  else
    $w^{t+1} = w^t$.
  end if
end for

In the following, we analyze the regret bound of the ODCA-PiL2 algorithm thanks to the analysis of ODCA in both cases $t \in \mathcal{N}_{\text{PiL2}}$ and $t \in \mathcal{M}_{\text{PiL2}} \setminus \mathcal{N}_{\text{PiL2}}$, as the following lemma. Before stating, we assume that $x_t \neq 0, \forall t$ and there exists a vector $u^* \in \mathbb{R}^n$ such that for all $t = 1, \ldots, T$,

$$y_t\langle u^*, x_t\rangle \geq 2\tau, \tag{2.37}$$

where $\tau = \max_{t=1,\ldots,T} \tau_{1t}$.

Lemma 2.3. *For the loss function* (2.35) *and step* $t \in \mathcal{M}_{\text{PiL2}}$*, the conditions (i)-(iv) are satisfied with the suitable parameters* α*,* γ *and vector* u^**. Moreover, the condition (iii) is satisfied for all* $\beta \geq 0$*.*

Proof. Similarly to Lemma 2.1 and Lemma 2.2, it is easy to check that $u^* \in \arg\min_{w\in\mathbb{R}^n} f_t^{(3)}(w)$ for all $t \in \mathcal{M}_{\text{PiL2}}$ and if $t \in \mathcal{N}_{\text{PiL2}}$, then $u^* \neq w^t$, since $y_t\langle w^t, x_t\rangle < \tau_{1t}$ while $y_t\langle u^*, x_t\rangle \geq 2\tau_{1t}$.

We will show that Assumptions 2.2–2.4 are satisfied for both cases $t \in \mathcal{N}_{\text{PiL2}}$ and $t \in \mathcal{M}_{\text{PiL2}} \setminus \mathcal{N}_{\text{PiL2}}$. Let us define the function $\overline{g}_t^{(3)} := g_t^{(3)} - \langle z^t, \cdot\rangle$.

• As for $t \in \mathcal{N}_{\text{PiL2}}$, we have

$$\overline{g}_t^{(3)}(w^t) - \overline{g}_t^{(3)}(u^*) = \frac{\tau_{1t} - y_t\langle w^t, x_t\rangle}{\tau_{1t}} \geq \frac{\alpha}{2}\|u^* - w^t\|^2,$$

where $\alpha \leq \min\limits_{t\in\mathcal{N}_{\text{PiL2}}} \dfrac{2(\tau_{1t} - y_t\langle w^t, x_t\rangle)}{\tau_{1t}\|u^* - w^t\|^2}$. Thus, Assumption 2.2 is satisfied.

Regarding Assumption 2.3, we have that for $\beta \geq 0$,

$$h_t^{(1)}(u^*) - h_t^{(1)}(w^t) - \langle z^t, u^* - w^t\rangle = 0 \leq \frac{\beta}{2}\|u^* - w^t\|^2.$$

Concerning Assumption 2.4, we have

$$\begin{aligned} g_t^{(3)}(u^*) - g_t^{(3)}(w^t) - \langle r^t, u^* - w^t \rangle &= g_t^{(2)}(u^*) - g_t^{(2)}(w^t) - \langle r^t, u^* - w^t \rangle \\ &= \frac{y_t \langle w^t, x_t \rangle}{\tau_{1t}} - 1 + \langle \frac{y_t x_t}{\tau_{1t}}, u^* - w^t \rangle \\ &= \frac{y_t \langle u^*, x_t \rangle}{\tau_{1t}} - 1 \geq 1. \end{aligned}$$

With $\gamma \leq \min\limits_{t \in \mathcal{N}_{\text{PiL2}}} \dfrac{2}{\|u^* - w^t\|^2}$, Assumption 2.4 is verified.

• As for $t \in \mathcal{M}_{\text{PiL2}} \backslash \mathcal{N}_{\text{PiL2}}$, it is evident to notice that Assumptions 2.2–2.4 are satisfied for all $\alpha > 0$, $\beta \geq 0$, $\gamma > 0$.

The proof is complete. □

As a result of Lemma 2.3, the regret bound of the ODCA-PiL2 algorithm is stated as the following corollary.

Corollary 2.2. *Assume that* ODCA-PiL2 *generates the sequence* $\{w^t\}_{t=1,\ldots,T}$. *Then, we have*

$$\mathsf{Regret}^T_{\text{ODCA-PiL2}} \leq \frac{L^2 \left(1 + \log(T)\right)}{\gamma}$$

where the positive parameter $\gamma \leq \min\limits_{t \in \mathcal{N}_{\text{PiL2}}} \dfrac{2}{\|u^* - w^t\|^2}$ *and* L *is defined as* (2.15).

We also archive the mistake bound for the ODCA-PiL2 algorithm as the following theorem.

Theorem 2.4. *For any* $w \in \mathbb{R}^n$, *the number of prediction mistakes made by* ODCA-PiL2 *has an upper bound that is the root,* $\overline{x}_1$, *of the equation*

$$x - \overline{a}_{\text{PiL2}} - \overline{b}_{\text{PiL2}} \left(1 + \log(x)\right) = 0$$

where $\overline{a}_{\text{PiL2}} = \sum\limits_{t \in \mathcal{M}_{\text{PiL2}}} f_t^{(3)}(w)$, $\overline{b}_{\text{PiL2}} = L^2/\gamma_{\text{PiL2}}$, $\overline{x}_1 \geq \overline{b}_{\text{PiL2}}$ *and the positive parameter* $\gamma_{\text{PiL2}} \leq \min\{\gamma, L^2\}$.

Proof. This theorem is proven similarly to Theorem 2.3 but remark that from (2.23) and (2.36), the number of prediction mistakes is bounded from above by $|\mathcal{M}_{\text{PiL2}}|$. □

2.3.3 Sigmoid function

We here consider a loss function which takes a form of sigmoid function [83]. In particular, at step t, we observe the following loss function:

$$f_t^{(4)}(w) := \max\{1 - \tanh(\delta_t), 1 - \tanh(\kappa_t y_t \langle w, x_t \rangle)\} \tag{2.38}$$

where δ_t, κ_t is positive parameters, the increasing function $\tanh : \mathbb{R} \to [-1, 1]$ is defined by

$$\tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}.$$

It is straightforward to see that (2.24) is satisfied for the sequence $\{f_t^{(4)}\}_{t=1,\ldots,T}$. Since $f_t^{(4)}(w^t) > 0$ at all steps, we have to update w^t at all steps, which seems to be expensive in terms of rapidity. In order to alleviate this trouble, we propose a threshold $\varepsilon \in [0, 1)$ to define the set $\mathcal{M}_{\text{Sig}}$ as follows.

$$\mathcal{M}_{\text{Sig}} := \{t \in \{1, \ldots, T\} : 1 - \tanh(\kappa_t y_t \langle w^t, x_t \rangle) > \varepsilon\}.$$

In the case $t \notin \mathcal{M}_{\text{Sig}}$ i.e. $1 - \tanh(\kappa_t y_t \langle w^t, x_t \rangle) \leq \varepsilon < 1$, we obtain $y_t \langle w^t, x_t \rangle > 0$ that is to say $\ell_t(\overline{p}_t, y_t) = 0$. Thus, the condition (2.24) can be satisfied with the observed loss function $f_t^{(4)} = 0$.

When $t \in \mathcal{M}_{\text{Sig}}$, we see that $f_t^{(4)}(w)$ is DC and its DC decomposition is

$$f_t^{(4)}(w) = g_t^{(4)}(w) - h_t^{(4)}(w)$$

where

$$h_t^{(4)}(w) := \frac{2e^{-4\kappa_t y_t \langle w, x_t \rangle}}{e^{-2\kappa_t y_t \langle w, x_t \rangle} + 1},$$

$$g_t^{(4)}(w) := \max\left\{1 - \tanh(\delta_t) + h_t^{(4)}(w), c_t(w)\right\},$$

and the function $c_t : \mathbb{R}^n \to \mathbb{R}$ is defined by

$$c_t(w) := 2e^{-2\kappa_t y_t \langle w, x_t \rangle}.$$

It is evident that $g_t^{(4)}$ and $h_t^{(4)}$ are convex, positive functions on $\mathbb{R}^n$.

With the sigmoid function (2.38), ODCA consists of, at the step t, first calculating the vector $s^t \in (\partial g_t^{(4)} - \partial h_t^{(4)})(w^t)$ and then computing the prediction vector w^{t+1} as the update rule (2.8).

■ Compute $\partial g_t^{(4)}$, $\partial h_t^{(4)}$: the functions c_t and $h_t^{(4)}$ are differentiable and we have

$$\begin{aligned} \partial c_t(w) &= \{\nabla c_t(w)\} = \{-2\kappa_t y_t x_t c_t(w)\}, \\ \partial h_t^{(4)}(w) &= \{\nabla h_t^{(4)}(w)\} = \left\{-\frac{4\kappa_t y_t x_t e^{-2\kappa_t y_t \langle w, x_t \rangle}(2e^{2\kappa_t y_t \langle w, x_t \rangle} + 1)}{(e^{2\kappa_t y_t \langle w, x_t \rangle} + 1)^2}\right\}, \end{aligned}$$

and

$$\partial g_t^{(4)}(w) = \begin{cases} \{\nabla c_t(w)\} & \text{if } \delta_t > \kappa_t y_t \langle w, x_t \rangle, \\ \left[\nabla c_t(w), \nabla h_t^{(4)}(w)\right] & \text{if } \delta_t = \kappa_t y_t \langle w, x_t \rangle, \\ \{\nabla h_t^{(4)}(w)\} & \text{if } \delta_t < \kappa_t y_t \langle w, x_t \rangle. \end{cases}$$

Thus, for $t \in \mathcal{M}_{\text{Sig}}$, we choose $z^t \in \partial h_t^{(4)}(w^t)$, $r^t \in \partial g_t^{(4)}(w^t)$ and $s^t = r^t - z^t \in (\partial g_t^{(4)}(w^t) - z^t)$ as follows:

$$\begin{aligned} z^t &= \nabla h_t^{(4)}(w^t) = -2\kappa_t y_t x_t c_t(w^t) m_t, \\ r^t &= \begin{cases} \nabla c_t(w^t) & \text{if } \delta_t > \kappa_t y_t \langle w^t, x_t \rangle, \\ \nabla h_t^{(4)}(w^t) & \text{if } \delta_t \le \kappa_t y_t \langle w^t, x_t \rangle, \end{cases} \end{aligned}$$

where $m_t = \dfrac{2e^{2\kappa_t y_t \langle w^t, x_t \rangle} + 1}{(e^{2\kappa_t y_t \langle w^t, x_t \rangle} + 1)^2}$, and

$$s^t = \begin{cases} -\dfrac{4\kappa_t y_t x_t e^{2\kappa_t y_t \langle w^t, x_t \rangle}}{\left(e^{2\kappa_t y_t \langle w^t, x_t \rangle} + 1\right)^2} & \text{if } \delta_t > \kappa_t y_t \langle w^t, x_t \rangle, \\ 0 & \text{if } \delta_t \le \kappa_t y_t \langle w^t, x_t \rangle. \end{cases}$$

■ Choose the parameters $\delta_t, \kappa_t, \varepsilon$ and learning rate η_t: we propose the following choice: for all t, $\kappa_t = \kappa / \|x_t\|$ and $\eta_t = C/\sqrt{t}$ where κ, C are positive tuning parameters, ε is a tuning threshold in $[0, 1)$ and the positive number

$$\delta_t = \kappa_t y_t \langle w^t, x_t \rangle - \ln(m_t)/2. \tag{2.39}$$

Since $m_t \in (0, 1)$, then $\delta_t > \kappa_t y_t \langle w^t, x_t \rangle$.

Finally, ODCA applied to OBLC problems with the loss function (2.38) is described in Algorithm 2.3 (ODCA-Sig).

Algorithm 2.3 ODCA with sigmoid function (ODCA-Sig)

Initialization: let w^1 be an initial point, $\{\eta_t\}$ be a sequence of learning rates, $\{\kappa_t\}$, ε be the parameters mentioned above,
for $t = 1, 2, \ldots, T$ **do**
  if $1 - \tanh(\kappa_t y_t \langle w^t, x_t \rangle) > \varepsilon$ **then**
    $w^{t+1} = w^t + \eta_t \dfrac{4\kappa_t y_t x_t e^{2\kappa_t y_t \langle w^t, x_t \rangle}}{\left(e^{2\kappa_t y_t \langle w^t, x_t \rangle} + 1\right)^2}$.
  else
    $w^{t+1} = w^t$.
  end if
end for

We are going to present the regret bound of the ODCA-Sig algorithm. First, we indicate in Lemma 2.4 that Assumptions 2.1–2.4 can be satisfied for the proposed loss function (2.38). Then, we can archive the logarithmic regret as presented in Corollary 2.3. Finally, the mistake bound of the ODCA-Sig algorithm is stated in Theorem 2.5.

Before showing Lemma 2.4, we assume that $x_t \neq 0 \; \forall t$ and there exists a vector $u^* \in \mathbb{R}^n$ such that for all $t = 1, \ldots, T$,

$$\kappa_t y_t \langle u^*, x_t \rangle \ge \delta, \tag{2.40}$$

where $\delta = \max_{t=1,\ldots,T} \delta_t$.

Lemma 2.4. *For the loss function* (2.38) *and* $t \in \mathcal{M}_{\rm Sig}$*, there exist the parameters* $\alpha > 0$, $\beta \geq 0$, $\gamma > 0$ *and vector* u^* *such that Assumptions 2.1–2.4 are satisfied.*

Proof. From (2.40), we derive that $f_t^{(4)}(w) \geq 1 - \tanh(\delta_t) = f_t^{(4)}(u^*)$ for all $w \in \mathbb{R}^n$ which means $u^* \in \arg\min_{w\in\mathbb{R}^n} f_t^{(4)}(w)$. We also notice that $u^* \neq w^t$ for all $t \in \mathcal{M}_{\rm Sig}$ since $f_t^{(4)}(w^t) > 1 - \tanh(\delta_t)$.

Let us define for $t \in \mathcal{M}_{\rm Sig}$ the function

$$\begin{aligned}\overline{g}_t^{(4)}(w) &:= g_t^{(4)}(w) - \langle z^t, w\rangle \\ &= \max\left\{1 - \tanh(\delta_t) + h_t^{(4)}(w), c_t(w)\right\} + 2c_t(w^t)m_t\kappa_t y_t\langle w, x_t\rangle.\end{aligned}$$

As for Assumption 2.2, we have

$$\begin{aligned}&\overline{g}_t^{(4)}(w^t) - \overline{g}_t^{(4)}(u^*) \\ =\ & c_t(w^t) - c_t(u^*) + 2c_t(w^t)m_t\kappa_t y_t\langle w^t - u^*, x_t\rangle \\ =\ & c_t(w^t)\left[1 - \frac{c_t(u^* - w^t)}{2} - 2m_t\kappa_t y_t\langle u^* - w^t, x_t\rangle\right] \\ =\ & c_t(w^t)\left[1 - e^{-2\kappa_t y_t\langle u^* - w^t, x_t\rangle} - 2m_t\kappa_t y_t\langle u^* - w^t, x_t\rangle\right].\end{aligned}$$

Due to (2.39), it is easy to prove that $\overline{g}_t^{(4)}(w^t) - \overline{g}_t^{(4)}(u^*) > 0$. By setting

$$\alpha \leq \min_{t\in\mathcal{M}_{\rm Sig}} \frac{c_t(w^t) - c_t(u^*) + 2c_t(w^t)m_t\kappa_t y_t\langle w^t - u^*, x_t\rangle}{\|u^* - w^t\|^2}, \tag{2.41}$$

we derive that this assumption is verified.

Concerning Assumption 2.3, since $h_t^{(4)}$ is convex and differentiable we have

$$\begin{aligned}&h_t^{(4)}(u^*) - h_t^{(4)}(w^t) - \langle z^t, u^* - w^t\rangle \\ \leq\ & \langle \nabla h_t^{(4)}(u^*) - \nabla h_t^{(4)}(w^t), u^* - w^t\rangle \\ \leq\ & \|\nabla h_t^{(4)}(u^*) - \nabla h_t^{(4)}(w^t)\|.\|u^* - w^t\|, \\ \text{and}\quad & \|\nabla h_t^{(4)}(u^*) - \nabla h_t^{(4)}(w^t)\| \\ =\ & 8\kappa_t\|x_t\|\left|\frac{c_t(u^*)[c_t(-u^*)+1]}{[c_t(-u^*)+2]^2} - \frac{c_t(w^t)[c_t(-w^t)+1]}{[c_t(-w^t)+2]^2}\right| \\ \leq\ & \frac{\kappa_t\|x_t\|}{2}\left|[c_t(u^*)+2][c_t(-w^t)+2]^2 - [c_t(w^t)+2][c_t(-u^*)+2]^2\right| \\ \leq\ & \kappa_t\|x_t\|\left[|c_t(u^* - 2w^t) - c_t(w^t - 2u^*)| + 4|c_t(u^* - w^t) - c_t(w^t - u^*)|\right. \\ & \left. +2|c_t(u^*) - c_t(w^t)| + |c_t(-2w^t) - c_t(-2u^*)| + 2|c_t(-w^t) - c_t(-u^*)|\right].\end{aligned}$$

For any x, $y \in \mathbb{R}^n$, we have

$$\begin{aligned}c_t(x) - c_t(y) &= c_t(x)\left[1 - c_t(y - x)/2\right] \\ &= c_t(x)\left(1 - e^{-2\kappa_t y_t\langle y - x, x_t\rangle}\right) \\ &\leq 2c_t(x)\kappa_t y_t\langle y - x, x_t\rangle \\ &\leq 2c_t(x)\kappa_t\|x_t\|\|y - x\|.\end{aligned}$$

Thus, we readily derive that

$$|c_t(x) - c_t(y)| \leq 2\max\{c_t(x), c_t(y)\}\kappa_t\|x_t\|\|y - x\|.$$

Let us define

$$K_t = 2e^{2\kappa_t\left(|\langle u^* - w^t, x_t\rangle| + \max\{|\langle u^*, x_t\rangle|, |\langle w^t, x_t\rangle|\}\right)}.$$

It follows that

$$h_t^{(4)}(u^*) - h_t^{(4)}(w^t) - \langle z^t, u^* - w^t\rangle \;\; \leq \;\; 26\kappa_t^2\|x_t\|^2 K_t\|u^* - w^t\|^2.$$

Consequently, Assumption 2.3 is satisfied with

$$\beta \geq \max_{t\in\mathcal{M}_{\text{Sig}}} 26\kappa_t^2\|x_t\|^2 K_t. \tag{2.42}$$

Finally, regarding Assumption 2.4, we have

$$\begin{aligned}
& g_t^{(4)}(u^*) - g_t^{(4)}(w^t) - \langle r^t, u^* - w^t\rangle \\
= \;& c_t(u^*) - c_t(w^t) - \langle \nabla c_t(w^t), u^* - w^t\rangle \\
= \;& c_t(u^*) - c_t(w^t) + 2c_t(w^t)\kappa_t y_t\langle u^* - w^t, x_t\rangle \\
= \;& c_t(w^t)\left[\frac{c_t(u^* - w^t)}{2} - 1 + 2\kappa_t y_t\langle u^* - w^t, x_t\rangle\right] \\
= \;& c_t(w^t)\left[e^{-2\kappa_t y_t\langle u^* - w^t, x_t\rangle} - 1 + 2\kappa_t y_t\langle u^* - w^t, x_t\rangle\right].
\end{aligned}$$

Obviously, $e^{-2\kappa_t y_t\langle u^* - w^t, x_t\rangle} + 2\kappa_t y_t\langle u^* - w^t, x_t\rangle > 1$. If

$$\gamma \leq \min_{t\in\mathcal{M}_{\text{Sig}}} \frac{c_t(u^*) - c_t(w^t) + 2c_t(w^t)\kappa_t y_t\langle u^* - w^t, x_t\rangle}{\|u^* - w^t\|^2}, \tag{2.43}$$

then this assumption is verified.

The proof is complete. □

Corollary 2.3. *Assume that* ODCA-Sig *generates the sequence* $\{w^t\}_{t=1,\ldots,T}$. *Then, we have*

$$\mathsf{Regret}^T_{\text{ODCA-Sig}} \leq \frac{L^2(\alpha + \beta)\,(1 + \log(T))}{2\alpha\gamma}.$$

where the parameters α, β, γ, L *are defined as* (2.41), (2.42), (2.43), (2.15), *respectively.*

Theorem 2.5. *For any* $w \in \mathbb{R}^n$, *the number of prediction mistakes made by* ODCA-Sig *has an upper bound that is the root,* $\overline{x}_1$, *of the equation*

$$x - \overline{a}_{\text{Sig}} - \overline{b}_{\text{Sig}}\,(1 + \log(x)) = 0$$

where $\overline{a}_{\text{Sig}} = \sum_{t\in\mathcal{M}_{\text{Sig}}} f_t^{(4)}(w)$, $\overline{b}_{\text{Sig}} = L^2(\alpha + \beta)/(2\alpha\gamma_{\text{Sig}})$, $\overline{x}_1 \geq \overline{b}_{\text{Sig}}$ *and the positive parameter* $\gamma_{\text{Sig}} \leq \min\{\gamma, L^2(\alpha + \beta)/2\alpha\}$.

Remark 2.4. *(Time complexity)*

All three proposed ODCA *algorithms have the same time complexity of* $\mathcal{O}(nT)$ *where* T *is the total number of steps (more exactly, the total number of instances) and* n *is the number of features.*

2.4 Numerical Experiments

In the numerical experiments, we study the performance of the proposed Online DCA algorithms ODCA-PiL1, ODCA-PiL2, ODCA-Sig for OBLC problems and compare them with five state-of-the-art first-order learning algorithms: Perceptron [91, 102, 121], Online Gradient Descent (OGD) [134], Relaxed Online Maximum Margin Algorithm (ROMMA) [76], Approximate Maximal Margin Classification Algorithm (ALMA) [39], Passive-Aggressive learning algorithms (PA) [29] which are summarized in the paper [47]. We tested on a variety of benchmark datasets from UCI Machine Learning Repository [2] and LIBSVM website [3]. The datasets used in our experiments cover many areas (e.g. social sciences, biology, physics) and are shown in Table 2.1.

Table 2.1 – Datasets used in our experiments

Dataset	Name	# Instances (T)	# Features (n)
D1	a8a	32561	123
D2	cod-rna	271617	8
D3	colon-cancer	62	2000
D4	covtype	581012	54
D5	diabetes	768	8
D6	ijcnn1	141691	22
D7	magic04	19020	10
D8	splice	3175	60
D9	svmguide1	7089	4
D10	w7a	49749	300

■ Set up experiments: All experiments were implemented in MATLAB R2013b and performed on a PC Intel(R) Xeon(R) CPU E5-2630 v2, @ 2.60GHz of 32GB RAM. The open source MATLAB package for the state-of-the-art algorithms is available in [47]. The initial point of all algorithms is $0 \in \mathbb{R}^n$. We are interested in the following criteria to evaluate the effectiveness of the proposed algorithms: the mistake rate (defined as the ratio of the number of mistakes to the number of instances T) and the CPU time (in seconds). For a fair comparison, we follow a so-called validation procedure as described in [47] so as to choose the best parameters for different algorithms. In particular, we first perform each algorithm by running over one random permutation of the dataset with the different parameter values and then take the value corresponding to the smallest mistake rate. The ranges of parameters for the state-of-the-art algorithms are completely described in [47] while as for our algorithms, the best parameters τ_1, τ_2, C, κ and ε are searched from the range of $\{2^{-4}, 2^{-3}, \ldots, 2^4\}$, $\{1, 3, \ldots, 9\}$, $\{2^{-4}, 2^{-3}, \ldots, 2^4\}$, $\{0.1, 0.2, \ldots, 1\}$ and $\{0, 0.1, \ldots, 0.9\}$, respectively. After the validation procedure, each algorithm is conducted over 20 runs of different random permutations for each dataset with the best parameters chosen. The average results and their standard deviation over these 20 runs of all algorithms are reported in Table 2.2 and Table 2.3.

2. `http://www.ics.uci.edu/~mlearn/MLRepository.html`
3. `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

Table 2.2 – Average mistake rate (upper row) and its standard deviation (lower row) obtained by ODCA-PiL1, ODCA-PiL2, ODCA-Sig and Perceptron, ROMMA, ALMA, OGD, PA. Bold (resp. underlining) values indicate the first best (resp. second best) results.

Dataset	ODCA-PiL1	ODCA-PiL2	ODCA-Sig	Perceptron	ROMMA	ALMA	OGD	PA
D1	0.2088	<u>0.1575</u>	**0.1574**	0.2100	0.2249	0.1581	0.1577	0.2108
	0.001	0.001	0.001	0.001	0.002	0.001	0.001	0.002
D2	0.1739	<u>0.1176</u>	**0.1149**	0.1749	0.1517	0.1994	0.1657	0.2074
	0.001	0.001	0.000	0.001	0.065	0.001	0.000	0.001
D3	0.3088	**0.2379**	**0.2379**	0.3137	0.3771	0.4435	0.3032	0.2637
	0.039	0.046	0.050	0.043	0.086	0.056	0.060	0.041
D4	0.4697	<u>0.4237</u>	**0.4231**	0.4697	0.4804	0.4839	0.4676	0.4835
	0.001	0.001	0.000	0.001	0.011	0.001	0.001	0.000
D5	0.3194	<u>0.2615</u>	0.2621	0.3265	0.3072	0.2655	**0.2586**	0.3346
	0.015	0.008	0.007	0.013	0.015	0.010	0.007	0.016
D6	0.1045	<u>0.0705</u>	0.0740	0.1062	0.1008	**0.0699**	0.0767	0.1023
	0.024	0.001	0.018	0.000	0.001	0.001	0.001	0.001
D7	0.3593	<u>0.2786</u>	**0.2775**	0.3645	0.3365	0.3636	0.3557	0.3835
	0.007	0.002	0.001	0.002	0.034	0.003	0.003	0.003
D8	0.2969	0.2329	**0.2150**	0.2732	0.2684	0.2283	<u>0.2168</u>	0.2617
	0.056	0.006	0.003	0.004	0.009	0.006	0.004	0.007
D9	0.2492	0.2723	**0.2026**	0.2560	0.3037	0.2564	<u>0.2466</u>	0.3130
	0.007	0.116	0.002	0.004	0.032	0.004	0.010	0.005
D10	0.1147	**0.1005**	**0.1005**	0.1151	0.1094	0.1028	0.1037	0.1051
	0.012	0.000	0.000	0.000	0.001	0.001	0.001	0.000

Table 2.3 – Average CPU time (in seconds) (upper row) and its standard deviation (lower row) obtained by ODCA-PiL1, ODCA-PiL2, ODCA-Sig and Perceptron, ROMMA, ALMA, OGD, PA. Bold values indicate the best results.

Dataset	ODCA-PiL1	ODCA-PiL2	ODCA-Sig	Perceptron	ROMMA	ALMA	OGD	PA
D1	1.113	1.569	1.571	**1.084**	1.169	1.309	1.527	1.254
	0.007	0.020	0.027	0.014	0.017	0.016	0.021	0.016
D2	8.088	11.91	11.79	**7.807**	8.160	9.494	11.09	9.582
	0.057	0.060	0.057	0.042	0.243	0.111	0.048	0.045
D3	**0.003**	0.005	0.004	**0.003**	0.004	0.004	**0.003**	0.004
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
D4	22.72	33.28	33.68	**22.07**	24.36	25.63	29.51	26.80
	1.665	2.467	2.435	1.760	1.903	1.942	2.194	2.039
D5	**0.028**	0.041	0.042	**0.028**	0.029	0.034	0.041	0.034
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
D6	5.024	7.062	7.097	**4.873**	5.070	5.919	6.982	5.444
	0.045	0.046	0.053	0.019	0.025	0.044	0.038	0.028
D7	0.706	1.024	1.036	**0.687**	0.730	0.817	0.960	0.848
	0.003	0.004	0.003	0.002	0.010	0.002	0.002	0.002
D8	0.121	0.184	0.178	**0.117**	0.124	0.138	0.167	0.143
	0.002	0.001	0.000	0.000	0.000	0.000	0.000	0.001
D9	0.255	0.360	0.373	**0.248**	0.269	0.295	0.348	0.301
	0.001	0.007	0.001	0.003	0.003	0.001	0.001	0.001
D10	2.403	3.096	3.111	**2.365**	2.430	2.709	3.115	2.487
	0.086	0.121	0.113	0.092	0.065	0.089	0.079	0.050

Comments on computational results

i) In terms of the mistake rate: from Table 2.2, we observe that ODCA-Sig is the best algorithm, ODCA-PiL2 is the second and ODCA-PiL1 is slightly more efficient than the existing algorithms. In particular, ODCA-Sig is the first best on 8/10 datasets – the gain varies from 1e-4 to 0.2056 – especially, very efficient for the large datasets D2 (271617 instances) and D4 (581012 instances). The second is ODCA-PiL2 which outperforms the existing algorithms on 8/10 datasets (2 for the first best and 6 for the second best) – the gain varies 2e-4 to 0.2056. In addition, the mistake rate of ODCA-PiL2 is comparable to that of ODCA-Sig on 8/10 datasets with the difference from 0 to 0.0035. OGD and ROMMA come next and are fairly better than ODCA-PiL1 – the gain varies from 0.0037 to 0.028 and from 0.0021 to 0.0801, respectively. As for ODCA-PiL1, it is slightly more efficient than the existing algorithms Perceptron, ALMA, PA on 9/10, 5/10, 6/10 datasets – the gain varies from 4e-4 to 0.0071, from 0.0043 to 0.1347, from 0.002 to 0.0638 respectively.

ii) Concerning CPU time: all algorithms run very fast and can be classified as follows: Perceptron and ODCA-PiL1 are the fastest algorithms, three algorithms ROMMA, ALMA and PA come next and finally, ODCA-Sig, ODCA-PiL2 and OGD. More specifically, Perceptron runs the fastest on all 10 datasets while ODCA-PiL1 is comparable with Perceptron – the ratio of gain of Perceptron versus ODCA-PiL1 varies from 1 to 1.036 times. As for ODCA-Sig and ODCA-PiL2, although they are the slowest, their CPU time is fairly small and acceptable on all datasets – the ratio of gain of Perceptron versus ODCA-Sig (resp. ODCA-PiL2) varies from 1.315 to 1.526 (resp. from 1.309 to 1.666) times.

In summary, these orders of the algorithms in terms of the quality and the rapidity can state the fact that the better results the algorithm reaches, the more CPU time it needs. Among the comparative algorithms in these experiments, ODCA-Sig could be the algorithm which realizes the best trade-off between these two criteria.

2.5 Conclusion

In this chapter, we have intensively investigated an approach based on Online DC programming for the problems in online learning where the loss function at each online step is nonconvex and/or nonsmooth. We have developed an online version of DCA (Online DCA) and indicated that Online DCA covers some variants of well-known online gradient descent algorithm. Moreover, we have proposed a particular Online DCA based scheme (ODCA) where we have employed subgradient method with one iteration to treat each convex subproblem. We have analyzed the formal regret bound of ODCA, specially the logarithmic regret $\mathcal{O}(\log(T))$. As an application, we have considered a class of problems in the topic of online binary linear classification. In particular, we have proposed three different DC approximation functions (which are of the form of piecewise linear, sigmoid functions) and have developed the corresponding

ODCA based schemes (ODCA-PiLs, ODCA-Sig). These algorithms enjoy the logarithmic regret and their prediction mistake bounds are also given. Moreover, numerical experiments on various benchmark datasets have proved the efficiency of our three proposed algorithms when comparing with the five state-of-the-art online classification algorithms. It turns out that ODCA-Sig realizes the best trade-off between the quality and the rapidity.

Chapter 3

Online DCA for Prediction with Expert Advice

Abstract: *In this chapter, we are interested in developing an important class of online learning techniques, namely prediction with expert advice. It is characterized by the fact that, at each online step, making a prediction rests on the basis of experts' predictions. One common difficulty is that the loss function to assess the quality of the prediction at each step is often 0-1 (where 0-1 loss function returns 1 if the prediction is correct and 0 otherwise), more generally, nonconvex and nonsmooth. Thus, our works investigate a DC approximation approach to overcome this difficulty. We propose two particular schemes based on Online DCA, named ODCA-SG and ODCA-ESG. Specifically, each convex subproblem in two schemes is solved by approximating by one iteration of projected subgradient method and exponentiated subgradient method, respectively. We thoroughly study the analysis of ODCA-SG and ODCA-ESG schemes in terms of regret. In particular, ODCA-SG enjoys the logarithmic regret whereas ODCA-ESG yields the sublinear regret. As an application, we develop the techniques of prediction with expert advice for solving a class of online classification problems in which the experts are represented by the well-known online classification algorithms. Proposing DC approximation functions for 0-1 loss function, we derive two corresponding online algorithms based on ODCA-SG and ODCA-ESG. The performance of the proposed algorithms is verified on various benchmark classification datasets by comparing with two online convex algorithms and a well-known algorithm, namely weighted majority.*

3.1 Introduction

3.1.1 Background and related works

Online learning is a general class of techniques at the interface between machine learning and online optimization [43, 48]. As an important topic in online learning, prediction with expert advice establishes the foundations to the theory of prediction of individual sequences [23, 24]. From the late 1980s, the framework of prediction with expert advice was first introduced as a model of online learning by DeSantis, Markowsky, and Wegman; Littlestone and Warmuth [31, 78]. It is characterized by the fact that an online learner will directly take advice from a group of given experts. In particular, at each step, after receiving a question, the experts give their answers to that question. On their advice, the learner will predict the corresponding answer. After that, the correct answer is revealed. The quality of predictions is assessed by a loss function between the predicted answer and the correct one. The learner and each expert suffer corresponding losses. The goal for the learner is to make a sequence of predicted answers such that the cumulative regret with respect to each expert (that is the difference between the cumulative loss of the learner and that of the expert) is as small as possible.

In the literature, [78] proposed a well-known algorithm, namely weighted majority (WM), to handle the problems in prediction with expert advice. Its prediction's principle is quite simple: all weights of experts are initially set to 1 and at each step, if any expert has the false prediction, then his weight is reduced. In fact, there are different prediction ways which result in the different online learning algorithms. For example, the framework of prediction with expert advice rests on the weighted average prediction strategy [24, 133]. In the past decades, such problems with different approaches have been exploited by many researchers (see e.g. [23, 24, 31, 36, 44, 124]).

Another class of techniques of prediction with expert advice is derived from the online convex optimization approach. Many effective online convex algorithms were developed such as online gradient descent algorithm (with lazy/greedy projections) [134], exponentiated gradient algorithm [4, 51, 50]. However, there is one common difficulty in many problems in prediction with expert advice: the prediction domain or the loss function is not convex. Although two convexification techniques, described in [109], allow to overcome this difficulty, the disadvantage of online convex optimization approach has recently been highlighted. There exist several challenges that should be addressed when solving such problems: what optimization models for these problems, how to design simple, efficient prediction techniques, what convergence properties of these techniques, etc. These motivate us to investigate a new approach for such problems based on DC (Difference of Convex functions) and DCA (DC Algorithm), which are well-known as powerful nonconvex, nonsmooth optimization tools.

Our works develop the techniques of prediction with expert advice for solving the problems in the topic of online binary linear classification (OBLC) where each expert is represented by an OBCL algorithm. In OBLC, instance becomes available in a

sequential order. For the incoming new instance, the learner must predict the corresponding binary label and then the correct label is revealed. The knowledge of the correct label and the loss function to assess the predicted label is used to make the prediction better for future instance. There exist many online classification algorithms with different ways to make the prediction and to define the loss functions such as perceptron [91, 102, 121], relaxed online maximum margin algorithm (ROMMA) [76], approximate maximal margin classification algorithm (ALMA) [39], passive-aggressive learning algorithms (PA) [29], online gradient descent (OGD) [134] and their variants.

3.1.2 Our contributions

In this chapter, our contributions are multiple. Firstly, we present Online DC programming for prediction with expert advice where Online DCA can be applied. Secondly, we propose two particular Online DCA based schemes, namely ODCA-SG and ODCA-ESG, where each convex subproblem is solved by approximating by one iteration of projected subgradient method and exponentiated subgradient method, respectively. Thirdly, we study the analysis of these schemes in terms of regret (that is the difference between the cumulative suffered loss and the possibly smallest cumulative loss along the learning process). Specifically, the ODCA-SG scheme enjoys the logarithmic regret whereas the ODCA-ESG scheme yields the sublinear regret. Fourthly, we propose specific DC approximation functions for the 0-1 loss function and derive two corresponding online algorithms based on ODCA-SG and ODCA-ESG for solving the problems in OBLC. Finally, in order to evaluate the efficiency of our approach, we conduct some numerical experiments on many benchmark classification datasets and compare with two state-of-the-art online convex algorithms and the well-known WM algorithm.

The rest of the chapter is organized as follows. In Section 3.2, we describe a class of problems in prediction with expert advice and present Online DC programming for these problems where Online DCA can be applied. In Section 3.3, we propose two particular Online DCA based schemes and show the analysis of these schemes in terms of regret. How to develop the techniques of prediction with expert advice based on Online DCA for solving the problems in OBLC is shown in Section 3.4. Section 3.5 reports the numerical results on several test problems which is followed by some conclusions in Section 3.6.

3.2 Prediction with Expert Advice

In this section, we formally describe the prediction with expert advice for solving the problems in OBLC [109]. In particular, at the step t, the learner receives an instance with n features, denoted by $x_t \in \mathbb{R}^n$. He predicts a corresponding binary label, denoted by $\overline{p}_t \in \{0, 1\}$, based on the advice of d given experts, denoted by $\{k_i\}_{i=1,\ldots,d}$. The function $k_i : \mathbb{R}^n \to \{0, 1\}$ corresponds to the ith linear classifier, denoted by $u_i \in \mathbb{R}^n$,

where

$$k_i(x) := \mathbb{1}_{\langle u_i,x\rangle\geq 0}(x) = \begin{cases} 1 & \text{if } \langle u_i, x\rangle \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

In addition, the experts' advice is cast as a vector $v_t = (\tilde{p}_{i,t})_{i=1,\ldots,d} \in \{0,1\}^d$ where its ith element

$$\tilde{p}_{i,t} := k_i(x_t),\ i = 1,\ldots,d.$$

Let us define the set of weight vectors

$$\mathcal{S} := \left\{ w \in \mathbb{R}^d : \sum_{i=1}^{d} w[i] = 1 \text{ and } w[i] \geq 0,\ i = 1,\ldots,d \right\}$$

where $w[i]$ is the weight assigned to the ith expert. Remark that $\mathcal{S}$ is the probability simplex in $\mathbb{R}^d$.

At the step t, based on the experts' advice v_t and the updated weight vector $w^t \in \mathcal{S}$, the prediction label is chosen as $\overline{p}_t = p_t(w^t)$ where the function $p_t(w) = \mathbb{1}_{\langle w,v_t\rangle\geq\rho}(w)$ and ρ is a positive index. After predicting the label $\overline{p}_t$, the correct label, denoted by y_t, is revealed. The predictions of the learner are qualified by the loss function, denoted by ℓ_t, defined as follows.

$$\ell_t(p_t(w), y_t) := \mathbb{1}_{\{p_t(w)\neq y_t\}}(w) = \begin{cases} 1 & \text{if } p_t(w) \neq y_t, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

Recall that the learner's goal is to make a sequence of weight vectors $\{w^t\}$ so as to minimize the cumulative regret with respect to all d experts, defined by

$$\mathsf{Regret}_d := \sum_{t=1}^{T} \ell_t(\overline{p}_t, y_t) - \min_{i=1,\ldots,d} \sum_{t=1}^{T} \ell_t(\tilde{p}_{i,t}, y_t). \tag{3.2}$$

It is worth noting that $\mathsf{Regret}_d \leq \sum_{t=1}^{T} \ell_t(p_t(w^t), y_t) - \min_{w\in\mathcal{S}} \sum_{t=1}^{T} \ell_t(p_t(w), y_t)$. Obviously, the difficulty is that the 0-1 loss function ℓ_t is nonsmooth, nonconvex. Thus, in order to alleviate the difficulty, we suggest DC approximation functions for this loss function. In this case, we are able to deal with the resulting optimization problems by Online DC programming and Online DCA. In the next section, we are going to present how to develop the Online DCA based schemes and study the analysis of these schemes in terms of regret.

3.3 Solution methods based on Online DC programming and Online DCA

3.3.1 ODCA-SG and ODCA-ESG: ODCA schemes for Prediction with Expert Advice

Recall that at step t, we have a DC function, denoted by f_t, whose DC decomposition is $g_t - h_t$ and the set of predictions $\mathcal{S}$ is the nonempty probability simplex in the

Euclidean space $\mathbb{R}^d$. According to the Online DCA scheme in Section 1.2.2, at each step, we need to solve a convex subproblem (1.11). In this chapter, we propose two particular Online DCA based schemes in which (1.11) is solved by approximating by one iteration of projected subgradient method and exponentiated subgradient method respectively [22, 50, 111].

Applying one iteration of the projected subgradient method at the point w^t with constant step size of η, we have the following update rule.

$$w^{t+1} = \text{Proj}_{\mathcal{S}}(w^t - \eta s^t), \tag{3.3}$$

where $s^t \in (\partial g_t(w^t) - z^t)$. Thus, the ODCA-SG scheme is described as follows.

ODCA-SG: Online DCA-projected SubGradient scheme
Initialization: let w^1 be an initial point, η be the constant step size
for $t = 1, 2, \ldots, T$ **do**
1. Compute $z^t \in \partial h_t(w^t)$.
2. Compute $s^t \in (\partial g_t(w^t) - z^t)$.
3. Compute $w^{t+1} \in \mathcal{S}$ using (3.3).

end for

Similarly, for exponentiated subgradient method, we derive the following update rule

$$w^{t+1}[i] = \frac{w^t[i]e^{-\eta s^t[i]}}{\sum_{j=1}^{d} w^t[j]e^{-\eta s^t[j]}},\ i = 1, \ldots, d, \tag{3.4}$$

and the corresponding ODCA-ESG scheme is summarized as follows.

ODCA-ESG: Online DCA-Exponentiated SubGradient scheme
Initialization: let w^1 be an initial point, η be the constant step size
for $t = 1, 2, \ldots, T$ **do**
1. Compute $z^t \in \partial h_t(w^t)$.
2. Compute $s^t \in (\partial g_t(w^t) - z^t)$.
3. Compute $w^{t+1} \in \mathcal{S}$ using (3.4).

end for

Before going in detail to design the Online DCA algorithms for prediction with expert advice, we present the analysis of ODCA-SG and ODCA-ESG in terms of regret.

3.3.2 Analysis of ODCA-SG and ODCA-ESG

We concentrate on analyzing the regret bound of ODCA-SG and ODCA-ESG. Recall that the regret of an algorithm A until step T is defined as (1.14):

$$\mathsf{Regret}_A^T = \sum_{t=1}^{T} f_t(w^t) - \min_{w \in \mathcal{S}} \sum_{t=1}^{T} f_t(w),$$

where the sequence $\{w^1, w^2, \ldots, w^T\}$ is generated by the algorithm A.

In order to archive the regret bound of ODCA-SG and ODCA-ESG, we will work on four necessary assumptions about three parameters $\alpha > 0$, $\beta \geq 0$, $\gamma > 0$ and a vector $u^* \in \mathcal{S}$ which are similar to Assumptions 2.1–2.4 in the Section 2.2.4.

Suppose that L, U and D are respectively positive upper bounds of the sequence $\{s^t\}_{t=1,\ldots,T}$ in ℓ_2-norm, ℓ_∞-norm and $\{w^t - u^*\}_{t=1,\ldots,T}$ in ℓ_2-norm that is

$$\max_{t \in \{1,\ldots,T\}} \|s^t\|_2 \leq L, \sup_{t \in \{1,\ldots,T\}} \|s^t\|_\infty \leq U \text{ and } \sup_{t \in \{1,\ldots,T\}} \|w^t - u^*\|_2 \leq D. \tag{3.5}$$

Under these assumptions, the results of the regret bound of ODCA-SG can be achieved similarly in the analysis of ODCA in Section 2.2.4 (Theorems 2.1, 2.2), which is stated as the following theorems for ODCA-SG.

Theorem 3.1. *Assume that* ODCA-SG *generates the sequence* $\{w^t\}_{t=1,\ldots,T}$ *where Assumptions 2.1–2.3 are verified. Then, we have*

$$\mathsf{Regret}_{\text{ODCA-SG}}^T \leq \frac{3DL(\alpha+\beta)\sqrt{T}}{2\alpha}.$$

Theorem 3.2. *Assume that* ODCA-SG *generates the sequence* $\{w^t\}_{t=1,\ldots,T}$ *where Assumptions 2.1–2.4 are verified. Then, we have*

$$\mathsf{Regret}_{\text{ODCA-SG}}^T \leq \frac{L^2(\alpha+\beta)\,(1+\log(T))}{2\alpha\gamma}.$$

Next, the regret bound of ODCA-ESG can be archived as stated in Theorem 3.3.

Theorem 3.3. *Assume that* ODCA-ESG *generates the sequence* $\{w^t\}_{t=1,\ldots,T}$ *where Assumptions 2.1–2.3 are satisfied and* $w^1[i] = 1/d$, $i = 1, \ldots, d$. *Then, we have*

$$\mathsf{Regret}_{\text{ODCA-ESG}}^T \leq \frac{(\alpha+\beta)\,U\sqrt{2\log(d)T}}{\alpha}.$$

Proof. From the fact that Assumptions 2.1–2.3 are verified, we have

$$\mathsf{Regret}_{\text{ODCA-ESG}}^T \leq \left(1 + \frac{\beta}{\alpha}\right) \sum_{t=1}^{T} \langle s^t, w^t - u^* \rangle.$$

For the update rule (3.4), we need to find the upper bound of the sequence $\{\langle s^t, w^t - u^*\rangle\}_{t=1,\ldots,T}$. To obtain that, we use the proof technique of Lemma 2 in [22]. In particular, let

$$\mathfrak{D}(a\|b) = \sum_i a[i]\log(a[i]/b[i])$$

be the Kullback-Leibler distance between any two real vectors a and b belonging to the probability simplex $\mathcal{S}$. We have

$$\begin{aligned}
& \mathfrak{D}(u^*\|w^t) - \mathfrak{D}(u^*\|w^{t+1}) \\
= & \sum_i u^*[i]\log(w^{t+1}[i]/w^t[i]) \\
= & \sum_i u^*[i]\left(-\eta s^t[i]\right) - \sum_i u^*[i]\log\left(\sum_j w^t[j]e^{-\eta s^t[j]}\right) \\
= & -\eta\langle s^t, u^*\rangle - \log\left(\sum_j w^t[j]e^{-\eta s^t[j]+\eta\langle s^t,w^t\rangle-\eta\langle s^t,w^t\rangle}\right) \\
= & -\eta\langle s^t, u^*\rangle + \eta\langle s^t, w^t\rangle - \log\left(\sum_j w^t[j]e^{-\eta s^t[j]+\eta\langle s^t,w^t\rangle}\right) \\
= & \eta\langle s^t, w^t - u^*\rangle - \log\left(\sum_j w^t[j]e^{-\eta s^t[j]+\eta\langle s^t,w^t\rangle}\right) \\
\geq & \eta\langle s^t, w^t - u^*\rangle - \frac{\eta^2U^2}{2}.
\end{aligned}$$

The last inequality is obtained by the result of Lemma 12 in [22] and the fact that the expectation $\mathbb{E}[s^t[j]] = \langle s^t, w^t\rangle$. Thus, we yields

$$\langle s^t, w^t - u^*\rangle \leq \frac{\mathfrak{D}(u^*\|w^t) - \mathfrak{D}(u^*\|w^{t+1})}{\eta} + \frac{\eta U^2}{2}.$$

Summing up over t, we get

$$\mathsf{Regret}^T_{\text{ODCA-ESG}} \leq \left(1+\frac{\beta}{\alpha}\right)\left[\frac{\mathfrak{D}(u^*\|w^1) - \mathfrak{D}(u^*\|w^{t+1})}{\eta} + \frac{\eta U^2 T}{2}\right].$$

Since $\mathfrak{D}(a\|b) \geq 0$ for any $a, b \in \mathcal{S}$ and if $b[i] = 1/d$ for $i = 1, \ldots, T$ then $\mathfrak{D}(a\|b) \leq \log(d)$, for all $a \in \mathcal{S}$, we derive that

$$\mathsf{Regret}^T_{\text{ODCA-ESG}} \leq \left(1+\frac{\beta}{\alpha}\right)\left[\frac{\log(d)}{\eta} + \frac{\eta U^2 T}{2}\right].$$

Let us define $\eta := \dfrac{\sqrt{2\log(d)}}{U\sqrt{T}}$, it concludes the proof. □

In the next section, we will present how to design two algorithms based ODCA-SG and ODCA-ESG for solving a class of OBLC problems and provide a bound on the number of prediction mistakes (called a mistake bound, for short) for these algorithms based on the regret bound.

3.4 Online DCA for prediction with expert advice

Recall that the loss function ℓ_t defined by (3.1) is nonsmooth and nonconvex. At each step, we propose a DC approximation function f_t on $\mathcal{S}$ instead of the loss function ℓ_t. Let us denote by $\mathcal{M}$ the set of steps where the prediction of the algorithm may be false in the sense that $f_t(w^t) > 0$. If a step $t \notin \mathcal{M}$, then we observe the function $f_t(w) = 0$, $\forall w \in \mathbb{R}^n$. Otherwise, we define the DC function f_t satisfying the following condition

$$f_t(w^t) \geq \ell_t(p_t(w^t), y_t). \tag{3.6}$$

In this chapter, we suggest appropriate DC functions with the form of piecewise linear function. It is noticeable that if $y_t = 0$, then

$$\ell_t(p_t(w^t), y_t) = p_t(w^t) = \mathbb{1}_{\langle w, v_t\rangle \geq \rho}(w^t)$$

since $p_t(w^t) \in \mathcal{Y}$. Otherwise,

$$\ell_t(p_t(w^t), y_t) = 1 - p_t(w^t) = \mathbb{1}_{\langle w, v_t\rangle < \rho}(w^t).$$

Thus, there are two cases $y_t = 1$ and $y_t = 0$ corresponding to different DC functions. Let define $\tau_{1t}, \tau_{2t}, \tau_{3t}$ be the positive parameters and less than ρ for all t.

■ **Case 1** ($y_t = 1$): $\ell_t(w) = \mathbb{1}_{\langle w, v_t\rangle < \rho}(w)$.

A natural DC function would be defined as

$$\begin{array}{rccl} f_t^{(1)}: & \mathcal{S} & \to & [0,1] \\ & w & \mapsto & f_t^{(1)}(w) := \max\left\{0, \min\left\{1, \dfrac{\rho - \langle w, v_t\rangle}{\rho - \tau_{1t}}\right\}\right\}. \end{array} \tag{3.7}$$

Remark 3.1. *It is easy to check that for $w \in \mathcal{S}$, we have $0 \leq f_t^{(1)}(w) \leq \ell_t(w) \leq 1$ and $f_t^{(1)}(w) \to \ell_t(w)$ as $\tau_{1t} \to \rho$.*

A DC decomposition of $f_t^{(1)}$ is proposed as follows

$$f_t^{(1)} = g_t^{(1)} - h_t^{(1)}$$

where

$$g_t^{(1)}(w) = \max\left\{0, \frac{\rho - \langle w, v_t\rangle}{\rho - \tau_{1t}}\right\}, h_t^{(1)}(w) = \max\left\{0, \frac{\tau_{1t} - \langle w, v_t\rangle}{\rho - \tau_{1t}}\right\}.$$

For this DC function and $t \in \{1, \ldots, T\}$, we define that $t \in \mathcal{M}$ if and only if $f_t^{(1)}(w^t) > 0$ (which is equivalent to $\langle w^t, v_t\rangle < \rho$). For simplifying the presentation, throughout the chapter, we will restrict to the case $t \in \mathcal{M}$.

According to the ODCA-SG scheme (resp. ODCA-ESG scheme), we compute first the vector $z^t \in \partial h_t^{(1)}(w^t)$, $s^t \in (\partial g_t^{(1)}(w^t) - z^t)$ and then the vector w^{t+1} using (3.3) (resp. (3.4)).

■ Compute $\partial g_t^{(1)}$, $\partial h_t^{(1)}$: by the definition of $g_t^{(1)}$ and $h_t^{(1)}$, we have

$$\partial g_t^{(1)}(w) = \begin{cases} \left\{\dfrac{-v_t}{\rho - \tau_{1t}}\right\} & \text{if } \langle w, v_t\rangle < \rho, \\ \left[\dfrac{-v_t}{\rho - \tau_{1t}}, 0\right] & \text{if } \langle w, v_t\rangle = \rho, \\ \{0\} & \text{if } \langle w, v_t\rangle > \rho, \end{cases}$$

and

$$\partial h_t^{(1)}(w) = \begin{cases} \left\{\dfrac{-v_t}{\rho - \tau_{1t}}\right\} & \text{if } \langle w, v_t\rangle < \tau_{1t}, \\ \left[\dfrac{-v_t}{\rho - \tau_{1t}}, 0\right] & \text{if } \langle w, v_t\rangle = \tau_{1t}, \\ \{0\} & \text{if } \langle w, v_t\rangle > \tau_{1t}. \end{cases}$$

Here $[a, b]$ is the line segment between a and b. We choose $z^t \in \partial h_t^{(1)}(w^t)$, $r^t \in \partial g_t^{(1)}(w^t)$ and $s^t = r^t - z^t \in (\partial g_t^{(1)}(w^t) - z^t)$ as follows:

$$r^t = \frac{-v_t}{\rho - \tau_{1t}}, z^t = \begin{cases} \dfrac{-v_t}{\rho - \tau_{1t}} & \text{if } \langle w^t, v_t\rangle < \tau_{1t}, \\ 0 & \text{if } \tau_{1t} \le \langle w^t, v_t\rangle < \rho, \end{cases}$$

and

$$s^t = \begin{cases} 0 & \text{if } \langle w^t, v_t\rangle < \tau_{1t}, \\ \dfrac{-v_t}{\rho - \tau_{1t}} & \text{if } \tau_{1t} \le \langle w^t, v_t\rangle < \rho. \end{cases}$$

• Choose the parameter τ_{1t}: in order to satisfy (3.6), we propose the following choices

$$\tau_{1t} = \max\{\langle w^t, v_t\rangle, \tau_1\},\ t = 1, \ldots, T, \tag{3.8}$$

where $\tau_1 < \rho$ is a positive tuning parameter.

■ **Case 2** ($y_t = 0$): $\ell_t(w) = \mathbb{1}_{\langle w, v_t\rangle \ge \rho}(w)$.

We adopt a loss function as follows

$$\begin{aligned} f_t^{(2)}: \ \mathcal{S} &\to [0, 1] \\ w &\mapsto f_t^{(2)}(w) := \max\left\{0, \min\left\{1, \frac{\langle w, v_t\rangle - \rho}{\rho - \tau_{2t}}\right\}\right\}. \end{aligned} \tag{3.9}$$

Remark 3.2. *It is evident that for any $w \in \mathcal{S}$, we have $0 \le f_t^{(2)}(w) \le \ell_t(w) \le 1$ and $f_t^{(2)}(w) \to \ell_t(w)$ as $\tau_{2t} \to \rho$.*

It is easy to see that $f_t^{(2)}$ is a DC function with the following DC components $g_t^{(2)}$ and $h_t^{(2)}$:

$$g_t^{(2)}(w) = \max\left\{0, \frac{\langle w, v_t\rangle - \rho}{\rho - \tau_{2t}}\right\},\ h_t^{(2)}(w) = \max\left\{0, \frac{\langle w, v_t\rangle - 2\rho + \tau_{2t}}{\rho - \tau_{2t}}\right\}.$$

Remark that we have to choose the parameters τ_{2t} such that (3.6) holds true. However, it is never true if there exists a step t such that $\langle w^t, v_t\rangle = \rho$, since $\ell_t(\overline{p}_t, y_t) = 1$ and $f_t^{(2)}(w^t) = 0$. In such a case, we investigate another DC function, namely $f_t^{(3)}$, defined by

$$f_t^{(3)}(w) := \max\left\{0, \min\left\{1, \frac{\langle w, v_t\rangle - \tau_{3t}}{\rho - \tau_{3t}}\right\}\right\}, \tag{3.10}$$

and its DC decomposition is

$$f_t^{(3)}(w) = g_t^{(3)}(w) - h_t^{(3)}(w),$$

where

$$g_t^{(3)}(w) = \max\left\{0, \frac{\langle w, v_t\rangle - \tau_{3t}}{\rho - \tau_{3t}}\right\},\ h_t^{(3)}(w) = \max\left\{0, \frac{\langle w, v_t\rangle - \rho}{\rho - \tau_{3t}}\right\}.$$

Obviously, we have $f_t^{(3)}(w^t) = 1 = \ell_t(\overline{p}_t, y_t)$. For both DC functions, we define that the step t belongs to $\mathcal{M}$ if and only if $\langle w^t, v_t\rangle \geq \rho$.

■ Compute $\partial g_t^{(2)}$, $\partial h_t^{(2)}$: we have

$$\partial g_t^{(2)}(w) = \begin{cases} \left\{\dfrac{v_t}{\rho - \tau_{2t}}\right\} & \text{if } \langle w, v_t\rangle > \rho, \\ \left[\dfrac{v_t}{\rho - \tau_{2t}}, 0\right] & \text{if } \langle w, v_t\rangle = \rho, \\ \{0\} & \text{if } \langle w, v_t\rangle < \rho, \end{cases}$$

and

$$\partial h_t^{(2)}(w) = \begin{cases} \left\{\dfrac{v_t}{\rho - \tau_{2t}}\right\} & \text{if } \langle w, v_t\rangle > 2\rho - \tau_{2t}, \\ \left[\dfrac{v_t}{\rho - \tau_{2t}}, 0\right] & \text{if } \langle w, v_t\rangle = 2\rho - \tau_{2t}, \\ \{0\} & \text{if } \langle w, v_t\rangle < 2\rho - \tau_{2t}. \end{cases}$$

Thus, we choose $s^t = r^t - z^t \in (\partial g_t^{(2)} - \partial h_t^{(2)})(w^t)$ as follows:

$$r^t = \frac{v_t}{\rho - \tau_{2t}}, z^t = \begin{cases} \dfrac{v_t}{\rho - \tau_{2t}} & \text{if } \langle w^t, v_t\rangle > 2\rho - \tau_{2t}, \\ 0 & \text{if } \rho < \langle w^t, v_t\rangle \leq 2\rho - \tau_{2t}, \end{cases}$$

and

$$s^t = \begin{cases} 0 & \text{if } \langle w^t, v_t\rangle > 2\rho - \tau_{2t}, \\ \dfrac{-v_t}{\rho - \tau_{2t}} & \text{if } \rho < \langle w^t, v_t\rangle \leq 2\rho - \tau_{2t}. \end{cases}$$

• Choose the parameter τ_{2t}: in order to satisfy (3.6), we propose the following choices

$$\tau_{2t} = \max\{2\rho - \langle w^t, v_t\rangle, \tau_2\},\ t = 1, \ldots, T,$$

where $\tau_2 < \rho$ is a positive tuning parameter.

It is similar for the case $\langle w^t, v_t\rangle = \rho$.

■ Compute $\partial g_t^{(3)}$, $\partial h_t^{(3)}$: similarly, we have

$$r^t = \frac{v_t}{\rho - \tau_{3t}}, z^t = 0 \text{ and } s^t = \frac{v_t}{\rho - \tau_{3t}}.$$

• Choose the parameter τ_{3t}: we choose $\tau_{3t} = \tau_3$ where $\tau_3 < \rho$ is a positive tuning parameter.

Finally, from two ODCA-SG and ODCA-ESG schemes, Online DCA applied to prediction with expert advice is given by Algorithm 3.1 (ODCA-SG) and Algorithm 3.2 (ODCA-ESG) as follows.

Algorithm 3.1 ODCA-SG for prediction with expert advice (ODCA-SG)

Initialization: let w^1 be an initial point, η be the constant step size,
τ_1, τ_2, τ_3 be positive numbers less than ρ.
for $t = 1, 2, \ldots, T$ **do**
  if $y_t = p_t(w^t)$ **then**
    $w^{t+1} = w^t$
  else
    if $\tau_1 \leq \langle w^t, v_t\rangle < \rho$ or $\rho < \langle w^t, v_t\rangle \leq 2\rho - \tau_2$ **then**
      $s^t = \dfrac{-v_t}{\rho - \langle w^t, v_t\rangle}$
    else if $\langle w^t, v_t\rangle = \rho$ **then**
      $s^t = \dfrac{v_t}{\rho - \tau_3}$
    else
      $s^t = 0$
    end if
    $w^{t+1} = \mathrm{Proj}_{\mathcal{S}}(w^t - \eta s^t)$
  end if
end for

Thanks to the analysis of both ODCA-SG and ODCA-ESG schemes in Section 3.3.2, we archive the logarithmic regret of the ODCA-SG algorithm and sublinear regret of the ODCA-ESG algorithm as stated in Theorem 3.2 and Theorem 3.3 for DC functions. In order to get this result, we need to indicate that all four Assumptions 2.1–2.4 are satisfied as in Lemma 3.1 and Lemma 3.2. Throughout the lemmas, we further assume that at each step $t \in \mathcal{M}$,

$$\tau_{1t} = \langle w^t, v_t\rangle \text{ (resp. } \tau_{2t} = 2\rho - \langle w^t, v_t\rangle) \text{ and } v_t \neq 0. \tag{3.11}$$

When $v_t = 0 \in \mathcal{Y}^d$ or $\tau_{1t} > \langle w^t, v_t\rangle$ (resp. $\tau_{2t} > 2\rho - \langle w^t, v_t\rangle$) (which means $\tau_1 > \langle w^t, v_t\rangle$ (resp. $\tau_2 > 2\rho - \langle w^t, v_t\rangle$)), the algorithm makes no update, which still holds true for the update step in both algorithms (corresponding to $s^t = 0$).

Algorithm 3.2 ODCA-ESG for prediction with expert advice (ODCA-ESG)

Initialize: let w^1 be an initial point, η be the constant step size,
τ_1, τ_2, τ_3 be positive numbers less than ρ.
for $t = 1, 2, \ldots, T$ **do**
  if $y_t = p_t(w^t)$ **then**
    $w^{t+1} = w^t$
  else
    if $\tau_1 \leq \langle w^t, v_t \rangle < \rho$ or $\rho < \langle w^t, v_t \rangle \leq 2\rho - \tau_2$ **then**
      $s^t = \dfrac{-v_t}{\rho - \langle w^t, v_t \rangle}$
    else if $\langle w^t, v_t \rangle = \rho$ **then**
      $s^t = \dfrac{v_t}{\rho - \tau_3}$
    else
      $s^t = 0$
    end if
    for $i = 1, 2, \ldots, d$ **do**
      $w^{t+1}[i] = \dfrac{w^t[i] e^{-\eta s^t[i]}}{\sum_{j=1}^{d} w^t[j] e^{-\eta s^t[j]}}$
    end for
  end if
end for

We also assume that there exists a vector $u^* \in \mathcal{S}$ such that for all $t = 1, \ldots, T$,

$$\begin{array}{ll} \sum_{i \in I_t} u^*[i] > \rho & \text{if } y_t = 1, \\ \sum_{i \in I_t} u^*[i] < \rho & \text{if } y_t = 0 \text{ and } \langle w^t, v_t \rangle > \rho, \\ \sum_{i \in I_t} u^*[i] < \tau_{3t} & \text{otherwise,} \end{array} \tag{3.12}$$

where the set $I_t := \{i \in \{1, \ldots, T\} : v_t[i] = 1\}$.

Lemma 3.1. *For the DC function* (3.7) *and step $t \in \mathcal{M}$ with $y_t = 1$, there exist α, γ and u^* such that Assumptions 2.1, 2.2, 2.4 are verified. Moreover, Assumption 2.3 is satisfied for all $\beta \geq 0$.*

Proof. For each $t \in \mathcal{M}$, since $v_t \neq 0$, the set I_t is not empty. From (3.12), it is easy to check that $u^* \in \arg\min_{w \in \mathcal{S}} f_t^{(1)}(w)$ and thus, Assumption 2.1 is satisfied. It is worth noting that $u^* \neq w^t$ for all t. Indeed, assume the contrary that is some step t such that $u^* = w^t$, since $t \in \mathcal{M}$ then $\langle u^*, v_t \rangle < \rho$ which contradicts (3.12) that $\langle u^*, v_t \rangle \geq \rho$.

Now, we will show that Assumptions 2.2–2.4 can be satisfied.

• As for Assumption 2.2, let us define the function $\overline{g}_t^{(1)} := g_t^{(1)} - \langle z^t, \cdot \rangle$. From (3.11) and the definition of $g_t^{(1)}$, we derive that

$$\overline{g}_t^{(1)}(w^t) - \overline{g}_t^{(1)}(u^*) = \frac{\rho - \langle w^t, v_t \rangle}{\rho - \tau_{1t}} = 1 > 0.$$

Thus, there exists $\alpha > 0$ such that the condition (ii) is satisfied, in paricular, $\alpha \leq \min_{t\in\mathcal{M}} \dfrac{2}{\|u^* - w^t\|_2^2}$.

• Assumption 2.3 is satisfied for any $\beta \geq 0$. In particular, we have

$$h_t^{(1)}(u^*) - h_t^{(1)}(w^t) - \langle z^t, u^* - w^t\rangle = 0 \leq \frac{\beta}{2}\|u^* - w^t\|_2^2.$$

• Concerning Assumption 2.4, we have that for any $t \in \mathcal{M}$,

$$\begin{aligned} g_t^{(1)}(u^*) - g_t^{(1)}(w^t) - \langle r^t, u^* - w^t\rangle &= -\frac{\rho - \langle w^t, v_t\rangle}{\rho - \tau_{1t}} - \langle \frac{-v_t}{\rho - \tau_{1t}}, u^* - w^t\rangle \\ &= \frac{\langle u^*, v_t\rangle - \rho}{\rho - \tau_{1t}} \\ &\geq \frac{\gamma}{2}\|u^* - w^t\|_2^2, \end{aligned}$$

where $\gamma \leq \min_{t\in\mathcal{M}} \dfrac{2(\langle u^*, v_t\rangle - \rho)}{(\rho - \tau_{1t})\|u^* - w^t\|_2^2}$. The proof is complete. □

Lemma 3.2. *For the DC functions* (3.9), (3.10) *and step* $t \in \mathcal{M}$ *with* $y_t = 0$, *Assumptions 2.1,2.2 and 2.4 are satisfied with the suitable parameters* α, γ *and vector* u^*. *Moreover, Assumption 2.3 is satisfied for all* $\beta \geq 0$.

Proof. We readily derive from (3.12) that $f_t^{(2)}(u^*) = 0$ ($f_t^{(3)}(u^*) = 0$ when $\langle w^t, v_t\rangle = \rho$). Thus, Assumption 2.1 for both functions is satisfied and $u^* \neq w^t$ for all t. Similarly to Lemma 3.1, as for the function (3.9) (resp. (3.10)), if $\alpha \leq \min_{t\in\mathcal{M}} \dfrac{2}{\|u^* - w^t\|_2^2}$, $\beta \geq 0$, $\gamma \leq \min_{t\in\mathcal{M}} \dfrac{2(\rho - \langle v_t, u^*\rangle)}{(\rho - \tau_{2t})\|u^* - w^t\|_2^2}$ (resp. $\gamma \leq \min_{t\in\mathcal{M}} \dfrac{2(\tau_{3t} - \langle v_t, u^*\rangle)}{(\rho - \tau_{3t})\|u^* - w^t\|_2^2}$ when $\langle w^t, v_t\rangle = \rho$), then Assumptions 2.2–2.4 are satisfied, which concludes the proof. □

Consequently, we derive the following corollaries for the regret bound of the ODCA-SG and ODCA-ESG algorithm.

Corollary 3.1. *Assume that* ODCA-SG *generates the sequence* $\{w^t\}_{t=1,\ldots,T}$. *Then, we have*

$$\mathsf{Regret}^T_{\text{ODCA-SG}} \leq \frac{L^2\,(1 + \log(T))}{\gamma}$$

where L *is defined as* (3.5), *the positive parameter*

$$\gamma \leq \min_{t\in\mathcal{M}} \left\{ \frac{2}{\|u^* - w^t\|_2^2} \min\left\{ \frac{\kappa(\tau_{3t} - \langle v_t, u^*\rangle)}{\rho - \tau_{3t}}, \frac{\kappa(\rho - \langle v_t, u^*\rangle)}{\rho - \tau_{2t}}, \frac{\kappa(\langle u^*, v_t\rangle - \rho)}{\rho - \tau_{1t}} \right\} \right\}, \tag{3.13}$$

and the real function $\kappa(x) = x$ *if* $x > 0$, $+\infty$ *otherwise.*

Corollary 3.2. *Assume that* ODCA-ESG *generates the sequence* $\{w^t\}_{t=1,\ldots,T}$ *where* $w^1[i] = 1/d$, $i = 1, \ldots, d$. *Then, we have*

$$\mathsf{Regret}^T_{\text{ODCA-ESG}} \leq 2U\sqrt{2\log(d)T}.$$

where U *is defined as* (3.5).

The following theorems provide a mistake bound for the ODCA-SG/ODCA-ESG algorithm that is the bound on the number of steps at which $\overline{p}_t \neq y_t$. Before stating the theorems, at the step $t \in \mathcal{M}$, we define the function

$$f_t(w) := \begin{cases} f_t^{(1)}(w) & \text{if } y_t = 1, \\ f_t^{(2)}(w) & \text{if } y_t = 0, \langle w^t, v_t\rangle > \rho, \\ f_t^{(3)}(w) & \text{if } y_t = 0, \langle w^t, v_t\rangle = \rho. \end{cases}$$

Theorem 3.4. *For* $w \in \mathcal{S}$, *the number of prediction mistakes made by* ODCA-SG *has an upper bound that is the root,* $\overline{x}_1$, *of the equation*

$$x - \overline{a} - \overline{b}\,(1 + \log(x)) = 0$$

where $\overline{a} := \sum\limits_{t\in\mathcal{M}} f_t(w)$, $\overline{b} := L^2/\gamma_{\text{SG}}$, $\overline{x}_1 \geq \overline{b}$, *the positive parameter* $\gamma_{\text{SG}} \leq \min\{\gamma, L^2\}$.

Proof. From the inequality (3.6), Corollary 3.1 and the definition of γ_{SG}, we derive that for any $w \in \mathcal{S}$,

$$|\mathcal{M}| \;\leq\; \sum_{t\in\mathcal{M}} f_t(w^t) \leq \sum_{t\in\mathcal{M}} f_t(w) + \frac{L^2\,(1 + \log(|\mathcal{M}|))}{\gamma_{\text{SG}}},$$

where $|\mathcal{M}|$ is the number of steps of the set $\mathcal{M}$.

Using the definition of $\overline{a}$ and $\overline{b}$, it is evident that $\overline{a} \geq 0$, $\overline{b} \geq 1$ and the inequality (3.14) can be rewritten as follows.

$$|\mathcal{M}| \leq \overline{a} + \overline{b}\,(1 + \log(|\mathcal{M}|))\,.$$

The real function $r : (0, +\infty) \to \mathbb{R}$, $r(x) = x - \overline{a} - \overline{b}\,(1 + \log(x))$ is strictly convex on $(0, +\infty)$. Since $\lim\limits_{x\to 0^+} r(x) = \lim\limits_{x\to+\infty} r(x) = +\infty$ and $r(\overline{b}) \leq 0$, the equation $r(x) = 0$ has two roots $\overline{x}_1$, $\overline{x}_2$ such that $0 < \overline{x}_2 \leq \overline{b} \leq \overline{x}_1$. The proof is complete. □

Theorem 3.5. *For any* $w \in \mathcal{S}$, *the number of prediction mistakes made by* ODCA-ESG *is upper bounded by* $\left(\overline{c} + \sqrt{\overline{c}^2 + 4\overline{a}}\right)^2/4$ *where* $\overline{c} := 2U\sqrt{2\log(d)}$.

Proof. From the inequality (3.6), Corollary 3.2, we get that for any $w \in \mathcal{S}$,

$$|\mathcal{M}| \;\leq\; \sum_{t\in\mathcal{M}} f_t(w^t) \leq \sum_{t\in\mathcal{M}} f_t(w) + 2U\sqrt{2\log(d)|\mathcal{M}|}. \tag{3.14}$$

Using the definition of $\overline{a}$ and $\overline{c}$, it is evident that $\overline{a}, \overline{c} \geq 0$ and the inequality (3.14) can be rewritten as follows.

$$|\mathcal{M}| \leq \overline{a} + \overline{c}\sqrt{|\mathcal{M}|}.$$

It leads to the bound

$$|\mathcal{M}| \leq \frac{\left(\overline{c} + \sqrt{\overline{c}^2 + 4\overline{a}}\right)^2}{4}.$$

which concludes the proof. □

Remark 3.3. *(Comparison with some existing algorithms: Time complexity)*

We consider three existing algorithms for prediction with expert advice: a well-known WM *algorithm [78] and two online convex algorithms, namely online gradient descent with greedy projection* (OGD) *[134] and normalized exponentiated gradient* (NEG) *[4, 51, 50]. In these online convex algorithms, at the step t, the 0-1 function is approximated by hinge loss function f_t^{cv} [109] which is defined as $f_t^{cv}(w) = 2|\langle w, v_t\rangle - y_t|$ if $t \in \mathcal{M}$, 0 otherwise.*

From the work of [125], the complexity of the projection algorithm $\text{Proj}_{\mathcal{S}}(x)$ *in* $\mathbb{R}^d$ *is* $\mathcal{O}(d \log d)$ *with d be the number of given experts. Thus, we derive the complexity of all comparative algorithms given in Table 3.1. In view of the fact that the number of experts is small in comparison with T (for example, d = 5 as in the following experiments), the computation-time complexity of our* Online DCA *based algorithms is the same as that of other algorithms – the worst-case complexity of* $\mathcal{O}(T)$.

Table 3.1 – Time complexity of comparative algorithms with T and d be the number of rounds and the number of experts respectively.

ODCA-SG	ODCA-ESG	OGD	NEG	WM
$\mathcal{O}(Td \log d)$	$\mathcal{O}(Td)$	$\mathcal{O}(Td \log d)$	$\mathcal{O}(Td)$	$\mathcal{O}(Td)$

3.5 Numerical experiments

With the aim of evaluating the performance of our proposed algorithms, we conduct online binary classification tasks with expert advice. In order to construct the group of experts, we used five well-known online classification algorithms ($d = 5$) mentioned in Section 3.1: Perceptron [91, 102, 121], ROMMA [76], ALMA [39], PA [29] and OGD [134]. As mentioned previously, we compare our algorithms, namely ODCA-SG and ODCA-ESG, with the WM algorithm and two online convex algorithms, namely OGD and NEG. We tested on a variety of benchmark datasets from UCI Machine Learning Repository [1] and LIBSVM website [2]. The datasets used in our experiments cover many areas (e.g. social sciences, biology, physics, life sciences), which is shown in Table 3.2.

1. `http://www.ics.uci.edu/~mlearn/MLRepository.html`
2. `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

Table 3.2 – Datasets used in our experiments.

Dataset	Name	# Instances	# Features (n)
D1	a8a	32561	123
D2	cod-rna	271617	8
D3	colon-cancer	62	2000
D4	covtype	581012	54
D5	diabetes	768	8
D6	german.number	1000	24
D7	ionosphere	351	34
D8	madelon	1549	500
D9	mushrooms	8124	112
D10	spambase	4601	57
D11	svmguide1	7089	4

■ Set up experiments: All algorithms were implemented in Visual C++ version 11.0 and run on a PC Intel(R) Core(TM) i5-3470 CPU 3.20GHz of 8GB RAM. All experts are the first-order learning algorithms for large-scale online classification tasks as surveyed in [47] and implemented in MATLAB R2013b. The open source MATLAB package for the expert algorithms is available in [47]. In our experiment, each dataset is randomly divided into two sets as follows. A so-called training set including 20% of the whole data is used by the system of experts to learn linear classifiers u_i $(i = 1, \ldots, d)$, while a so-called test set consisting of the remaining dataset is adopted by all algorithms to make the predictions. The initial prediction vector $w^1 \in \mathcal{S}$ is set to $(1/d, \ldots, 1/d)^\top$. The positive index ρ is set to 0.5. The projection algorithm $\text{Proj}_{\mathcal{S}}(\cdot)$ is described in [125]. We are interested in the following criteria to evaluate the effectiveness of the proposed algorithms: the percentage of regret (denoted by %regret in %) and CPU time (in seconds). Specifically, the %regret is computed as

$$\%\text{regret} = \frac{\texttt{Regret}_d}{T}.100. \tag{3.15}$$

where $\texttt{Regret}_d$ is defined as (3.2) and T is the number of steps (corresponding to the number of instances in the test set). This value means the distance between the number of mistakes of the algorithm and that of the best expert. The smaller the %regret is, the better the algorithm would be. For a fair comparison, we follow a so-called validation procedure on the test set which is described as in [47] so as to choose the best parameters for different algorithms. In particular, we first perform each algorithm by running over one random permutation of the dataset with the different parameter values and then take the value corresponding to the smallest mistake rate. The ranges of parameters for the expert algorithms and existing algorithms are completely described in [47] while as for our algorithms, the parameters τ_1, τ_2, τ_3, η are chosen as follows. The parameters τ_1, τ_2, τ_3 are set to the same positive tuning parameter τ for all t $(\tau < \rho)$ and the constant step size η for ODCA-SG (resp. ODCA-ESG) is $\eta = C/\sqrt{T}$ (resp. $\eta = C\sqrt{\log(d)}/\sqrt{T}$) where τ and C are searched from the range of $\{0.00, 0.02, \ldots, 0.48\}$ and $\{2^{-4}, 2^{-3}, \ldots, 2^4\}$ respectively. The best values of τ of our algorithms on each dataset are shown in Table 3.3. After the validation procedure,

Table 3.3 – The best value of τ for ODCA-SG and ODCA-ESG during the parameter validation ($\tau < 0.5$)

Data	ODCA-SG	ODCA-ESG
a8a	0.40	0.40
cod-rna	0.48	0.48
colon-cancer	0.48	0.48
covtype	0.38	0.38
diabetes	0.48	0.48
german.number	0.00	0.40
ionosphere	0.40	0.26
madelon	0.14	0.22
mushrooms	0.40	0.18
spambase	0.48	0.48
svmguide1	0.38	0.38

each algorithm is conducted over 20 runs of different random permutations for each test set with the best parameters chosen. The average results and their standard deviation over these 20 runs of all algorithms are reported in Table 3.4 and Table 3.5. Figure 3.1 shows the number of mistakes of all algorithms along online process in the validation procedure on several notable datasets.

Comments on numerical results:

We observe from the numerical results of all algorithms that:

• In terms of %regret, ODCA-SG and ODCA-ESG are the most efficient. In particular, ODCA-SG is the best with the first best on 9/11 datasets and the second best on 1/11 datasets – the gain varies from 0.003% to 8.977% – especially, for the large datasets D2 (271617 instances) and D4 (581012 instances) (see Table 3.4). The second is ODCA-ESG which outperforms the existing algorithms on 9/11 datasets (6 for the first best and 3 for the second best) – the gain varies 0.005% to 8.415%. However, the %regret of ODCA-ESG is fairly comparable to that of ODCA-SG on 8/11 datasets with the difference from 0% to 0.076%. In addition, the values of %regret of our algorithms are actually small and stable (with the small standard deviation) on most datasets. In fact, there are several datasets (e.g. D2, D5, D9, D10) on which ODCA-SG and ODCA-ESG provided the *negative* value of %regret. That is to say, our algorithms can make predictions even better than the best experts. Additionally, Figure 3.1 indicates that the number of mistakes of our algorithms is less than that of other algorithms along the online process, particularly in large datasets.

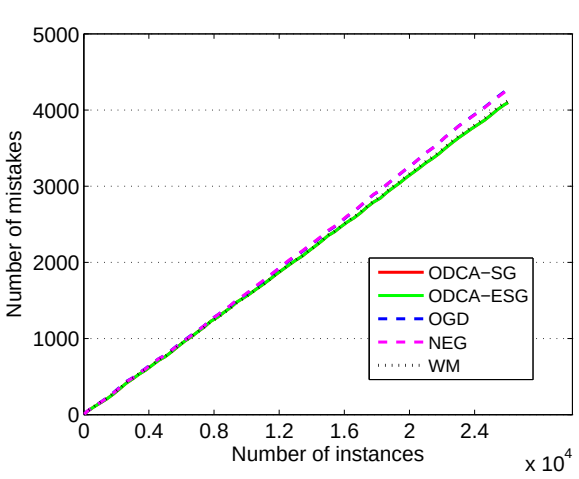
• Concerning CPU time: all algorithms run very fast. From Table 3.5, their CPU time values are comparable on most datasets. However, for the large datasets (D2 and D4), the rapidity of the algorithms can be classified as follows: ODCA-ESG and NEG are the fastest algorithms, the algorithm ODCA-SG comes next and finally, OGD and WM – the ratio of gain of the fastest ODCA-ESG versus the slowest WM is up to 1.143 times. This can be easily explained by the time complexity of all algorithms as

Table 3.4 – Average percentage of regret (%regret in %) (upper row) defined as (3.15) and its standard deviation (lower row) obtained by ODCA-SG, ODCA-ESG, OGD, NEG and WM. Bold (resp. underlining) values indicate the first best (resp. second best) results.

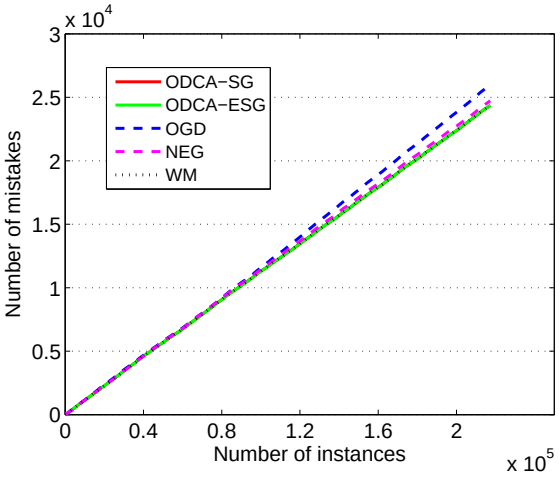
Dataset	ODCA-SG	ODCA-ESG	OGD	NEG	WM
D1	**0.010**	<u>0.013</u>	0.865	0.847	0.083
	0.014	0.016	0.095	0.085	0.019
D2	**-0.084**	**-0.084**	0.711	0.079	-0.079
	0.000	0.000	0.033	0.014	0.003
D3	**0.000**	**0.000**	3.500	2.200	**0.000**
	.000	0.000	3.514	3.516	0.000
D4	**0.000**	<u>0.562</u>	5.960	2.652	8.977
	0.001	2.446	0.054	0.028	0.024
D5	**-1.140**	**-1.140**	**-1.140**	-1.116	0.098
	0.000	0.000	0.000	0.232	0.567
D6	0.869	**0.268**	1.063	1.000	<u>0.744</u>
	0.756	0.782	0.862	0.746	0.341
D7	<u>1.263</u>	1.281	1.708	1.548	**0.765**
	0.931	0.715	1.117	0.909	0.304
D8	**0.153**	<u>0.452</u>	0.823	0.791	1.105
	0.364	0.660	0.839	0.903	0.556
D9	**-0.012**	0.064	0.030	0.036	<u>0.014</u>
	0.012	0.059	0.036	0.042	0.022
D10	**-0.217**	**-0.217**	7.311	2.017	0.073
	0.000	0.000	0.540	0.320	0.038
D11	**0.007**	**0.007**	5.820	2.527	0.040
	0.014	0.014	0.359	0.203	0.017

Table 3.5 – Average CPU time (in seconds) obtained by ODCA-SG, ODCA-ESG, OGD, NEG and WM. Bold values indicate the best results.

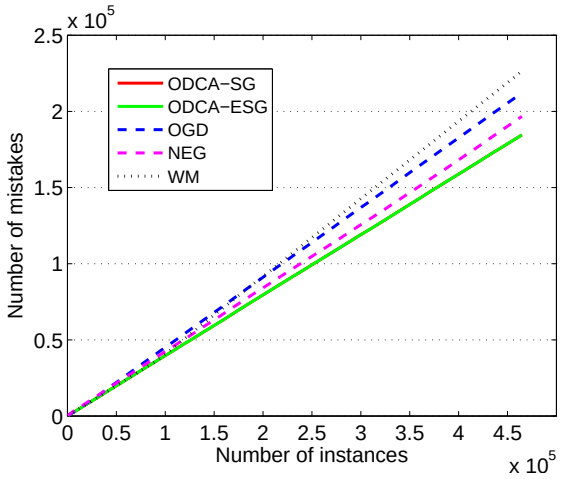
Dataset	ODCA-SG	ODCA-ESG	OGD	NEG	WM
D1	0.019	0.019	0.019	**0.018**	**0.018**
D2	0.039	**0.036**	0.040	0.037	0.041
D3	0.001	**0.000**	0.001	**0.000**	0.001
D4	0.194	**0.175**	0.200	0.190	0.200
D5	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
D6	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
D7	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
D8	**0.003**	**0.003**	**0.003**	**0.003**	**0.003**
D9	**0.004**	**0.004**	**0.004**	**0.004**	**0.004**
D10	**0.001**	**0.001**	0.002	0.002	**0.001**
D11	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**



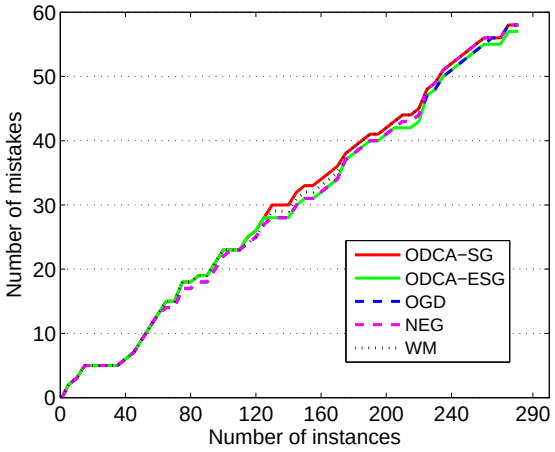
(a) D1: a8a



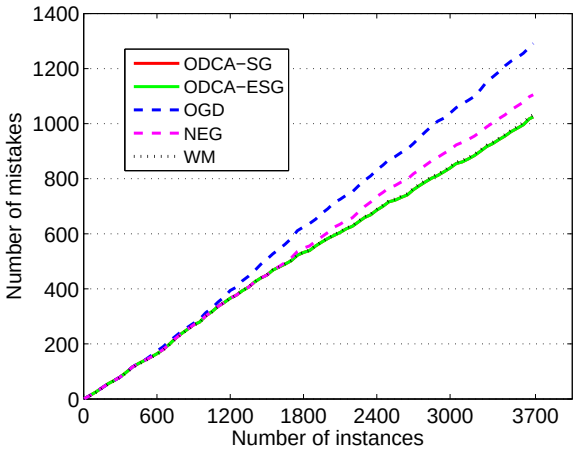
(b) D2: cod-rna



(c) D4: covtype



(d) D7: ionosphere



(e) D10: spambase

Figure 3.1 – The number of mistakes of all five algorithms with respect to the best value of parameters in the validation procedure on five notable datasets

discussed in Remark 3.3 and the fact that the learner only updates the predictions when one makes a false mistake on the previous round. Therefore, the more the algorithm makes the good predictions, the faster it would be.

• As for the value of τ, from Table 3.3, the best value for ODCA-SG and ODCA-ESG is almost near to ρ ($\rho = 0.5$) in the procedure of parameter validation. This is an interesting illustration for the particular property of the DC function as noticed in Remark 3.1 and Remark 3.2.

In summary, our Online DCA based algorithms work well in terms of the quality and the rapidity on these test problems. Specially, ODCA-SG seems to be preferred since it realizes the best trade-off between these two criteria.

3.6 Conclusions

In this chapter, we have intensively investigated an Online DC programming and Online DCA approach for developing a class of online learning techniques, namely prediction with expert advice. In particular, we have exploited DC approximation functions for the nonsmooth, nonconvex 0-1 loss function on each online step. The resulting optimization problems are solved by Online DCA. Solving each convex subproblem in Online DCA by approximating by one iteration of two variants of subgradient method, we have developed two particular Online DCA based schemes, namely ODCA-SG and ODCA-ESG. We have studied the analysis of both schemes in terms of regret, which enjoys the sublinear/logarithmic regret. As an application, we have derived two algorithms based on ODCA-SG and ODCA-ESG for OBLC. Numerical results on various benchmark classification datasets show the efficiency of our approach when comparing with the well-known existing algorithms.

Part II

Reinforcement learning

Chapter 4

Reinforcement Learning: Introduction and Related Works

Abstract: *In this chapter, we briefly introduce reinforcement learning which is a general class of machine learning techniques for dealing with sequential decision problems. The goal of reinforcement learning is to estimate the optimal learning policy in a dynamic environment typically formulated as a Markov decision process (with an incomplete model). It is well-known that one can tackle this task through the problem of finding the zero of the so-called optimal Bellman residual, a classical concept of dynamic programming. The background of reinforcement learning and its related works which we concern in this dissertation are also presented.*

Reinforcement learning (RL) is a general class of machine learning techniques where an agent must learn behavior through trial-and-error interactions with a dynamic environment which is typically modeled as a Markov Decision Process (MDP). RL is an intersection of many active research subfields including machine learning, statistical learning, behavior optimization, robotics, etc (see e.g. [127]). Currently, the applications of RL are very wide in various areas such as manufacturing, management, transportation.

The goal of RL is to estimate the optimal policy of an MDP without knowing its complete model. From the last two decades, the power of RL in dealing with the curse of dimensionality and the curse of modeling on large-scale and complex problems of dynamic optimization, in particular the MDP problem and its variants, has been more and more confirmed by various works. This success of RL is due to its strong mathematical tools in the principles of Dynamic Programming (DP). This dissertation contributes to enrich these tools by investigating a unified DC (Difference of Convex functions) programming framework and efficient DCA based approach for solving the problems of RL, in particular, in batch mode (i.e. a fixed set of learning experience is given a priori) or online mode (i.e. the learning experience is collected piece-by-piece through the interaction with the environment).

4.1 Background and related works

An MDP [7, 9, 99] can be described by the 5-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma >$ where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function on state-action transitions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a state transition probability function in which $\mathcal{P}(s'|s, a)$ represents the probability of transition from $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ upon taking action $a \in \mathcal{A}$ and $\gamma \in (0, 1)$ is a discount factor. To avoid the complication of systems having continuous state spaces and continuous action spaces, a *finite* MDP is often considered, i.e. the state space and the action space are finite, specifically $\mathcal{S} = \{s_i\}_{i=1,\ldots,N_{\mathcal{S}}}, \mathcal{A} = \{a_i\}_{i=1,\ldots,N_{\mathcal{A}}}$. A policy π of an MDP is called deterministic stationary if it specifies the same action each time a state is visited, i.e. π is a mapping from $\mathcal{S}$ to $\mathcal{A}$ (for short, $\pi \in \mathcal{A}^{\mathcal{S}}$) where $\pi(s)$ is the action the agent takes in the state s. Broadly, the aim of the agent learning in this environment is to maximize the cumulative reward it receives (which can be the total reward, the average reward, or the total discounted reward).

RL methods [118] are employed to address MDPs without knowledge of the perfect dynamic (a model of $\mathcal{P}$ or $\mathcal{R}$). Hence, value functions are usually used in RL to measure how good is each state and/or action, and an RL agent may include one or more of three components: the policy (agent's behavior), the value function (which can be the state value function and/or the state-action value function), and the model (agent's representation of the environment). Generally, RL techniques can be divided into three categories: the value-based approach estimates first the optimal value function (which is the maximum value achievable under any policy) and then determines an optimal policy with relative easiness, the policy-based approach directly searches for

the optimal policy, and the model-based approach builds a model of the environment. For an overview and a tutorial survey on RL, refer to [13, 17, 42, 113, 118, 119].

In this work we focus on RL using the value function based approach to address the above finite MDP in an infinite time horizon, discounted reward setting. More precisely, given a fixed set of samples generated from state transitions in a finite MDP, the goal is to search for an optimal policy π^* that maximizes a discounted, infinite-horizon optimality criterion (see e.g. [113, 118]):

$$\pi^* \in \arg\max_{\pi \in \mathcal{A}^{\mathcal{S}}} \mathbb{E}^{\pi} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, \pi(s_t)) \right],$$

where $\mathbb{E}^{\pi}[\cdot]$ is the expectation given that the agent follows the policy π. Instead of directly finding π^* over the space $\mathcal{A}^{\mathcal{S}}$, we estimate its quality via the state-action value function Q^{π}, viewed as a link between that criterion and π.

For a given policy π, the state-action value function under policy π is defined as $Q^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$,

$$Q^{\pi}(s, a) := \mathbb{E}^{\pi} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, \pi(s_t)) \middle| s_0 = s, a_0 = a \right]. \tag{4.1}$$

The optimal state-action value function, denoted by Q^*, is given by

$$Q^*(s, a) := \max \left\{ Q^{\pi}(s, a) : \pi \in \mathcal{A}^{\mathcal{S}} \right\}. \tag{4.2}$$

When Q^* is obtained, the optimal policy is computed as

$$\pi^*(s) = \arg\max \left\{ Q^*(s, a) : a \in \mathcal{A} \right\}.$$

Given a function $Q \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$, a policy $\pi \in \mathcal{A}^{\mathcal{S}}$ satisfying

$$\pi(s) \in \arg\max_{a \in \mathcal{A}} Q(s, a), \forall s \in \mathcal{S}$$

is said to be *greedy* with respect to Q. Specially, π^* is greedy with respect to Q^*, say $Q^* = Q^{\pi^*}$.

RL methods are employed to solve two basic problems: the prediction problem and the control problem. The goal of the prediction problem is to compute the value function for an arbitrary given policy (or evaluate the given policy), while the control problem aims to estimate the optimal value function (or optimal policy) directly. In both problems, most of the RL techniques often improve the policy in different ways to realize the best trade-off between exploitation and exploration [118] (for example, ε-greedy methods [126], softmax methods [80]). Our works concern RL techniques for control problems with ε-greedy method.

To deal with large state and action spaces in MDPs, RL methods with function approximation techniques have been intensively studied in the literature, a comprehensible

survey on these techniques is given in [17, 127, 129]. In the framework of the value function based approach, several value function approximation algorithms using different function approximators were proposed. These RL algorithms either determine (incrementally) a value for each state through satisfaction of the Bellman equation (e.g. approximate Q-learning [115, 118, 120, 126], approximate SARSA algorithms [114, 117], adaptive learning) or minimize directly a norm of empirical Bellman residual/Optimal Bellman residual (the direct Bellman residual minimization approach, e.g. [3, 5, 38, 81, 86, 98, 106, 107, 113]).

When an approximate function is used to learn a value function, tight performance bounds on greedy policies are needed to get good resulting control. Bellman Residual minimization (used at the first time in the work of Schweitzer and Seidman in 1985 [107] for computing approximate state value functions assuming the full knowledge of a finite MDP) is an efficient approach in RL which can meet such requirement (see e.g. [81, 86, 128]). In fact, a common practice of RL applications is to minimize the Bellman residual and then use the corresponding greedy policy.

The Bellman operator is a widely used concept in DP. In RL, the Bellman operator is defined on the set of state value functions and/or the set of state-action value functions. As we focus on state-action value functions, all notations presented here concern only state-action value functions (similar notations exist for state value functions). The Bellman operator on state-action functions is defined by $B^\pi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$,

$$B^\pi Q(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s,a) Q(s', \pi(s')),\ \forall (s,a) \in \mathcal{S}\times\mathcal{A},$$

and the Optimal Bellman operator on state-action functions is defined by (see e.g. [128]) $B^* : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$,

$$B^* Q(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s,a) \max_{a'\in\mathcal{A}} Q(s', a'),\ \forall (s,a) \in \mathcal{S}\times\mathcal{A}.$$

The concepts of Bellman Residual (BR) and Optimal Bellman Residual (OBR) arise from the standard results of the theory of DP that are, for any policy π the value function Q^π is the unique solution of the Bellman equation $Q = B^\pi Q$, and the optimal value function Q^* is the unique solution of the Bellman equation $Q = B^* Q$. The BR is then defined by $B^\pi Q - Q$ while the OBR is $B^* Q - Q$ (see e.g. [128]). Thus, to find the zero of BR (resp. OBR), the idea of minimizing a norm (ℓ_∞/ℓ_p, $p \geq 1$) of BR (resp. OBR) is natural. Moreover, as mentioned in [128], there is a very natural correspondence between Temporal Difference (TD) errors in TD approach [116] and the BR measures, and similarly, the individual components of the BR across the state-action function Q are very closely related to what Q-learning [126] tries to reduce toward zero. In the earlier step of RL, the ℓ_∞-norm and ℓ_2-norm were often used, and later, the weighted ℓ_p-norm was investigated.

BR minimization approach is intensively studied in RL literature on both theoretical and algorithmic point of views (e.g. [3, 5, 38, 81, 86, 106, 107, 113, 128]). Tight

performance bounds on greedy policies were proved in [128] for ℓ_∞-norm and in e.g. [81, 86] for ℓ_p-norm and weighted ℓ_p-norm. On the algorithmic level, works in this direction can be divided into two groups. The first (OBR-based approach) minimizes directly a norm of OBR while the second (BR-based approach) works by alternating between the evaluation of the policy through BR minimization and the policy improvement. In fact, the function OBR is more complex than BR due to the max operator over the space $\mathcal{A}$, and, as indicated in [113], minimizing a norm of the empirical OBR over the space of approximation functions is difficult, even in the case of linear approximation, since the resulting optimization problems are nonconvex, nonsmooth and thus solving them by global approaches is very hard in large-scale settings. Hence, the OBR-based approach was less worked than the BR-based approach, there exist some local optimization techniques, for example, neural networks for ℓ_2-norm of OBR, although global convergence is not guaranteed.

In the literature, Baird [5] proposed residual algorithms which can be regarded as a weighted average of a direct algorithm with a residual gradient algorithm. The update step in the direct algorithm is the same as the TD(0) algorithm, whereas the residual gradient algorithm updates the weight in gradient descent on the ℓ_2-norm of BR and OBR. Direct algorithms run fast but unstably, while residual gradient algorithms enjoy guaranteed convergence to a local optimum but slow. This residual algorithm takes advantage of the benefits of both direct and residual gradient algorithms in terms of convergence and its speed when using function approximator. Moreover, Geist and Pietquin [38] have indicated that the RL methods via value-function approximation can be divided into three main approaches: bootstrapping, residual and projected fixed-point. Among them, residual approaches with gradient descent, the closest to our works, are represented in more detail for both BR and OBR in the state-action value function as the main works in [5]. However, as indicated in [38], the computation of the gradient of sampled OBR functions is not straightforward since these functions are nonsmooth and nonconvex in the function-approximation space. Our works focus on developing RL techniques in both batch mode and online mode by addressing the different norms of empirical OBR via value-function linear approximation. Actually, most works in RL are considered in online mode since the sample is collected via the exploitation and exploration procedure with the environment after each step of updating the policy, meanwhile in batch mode, the set of samples is given a priori (see e.g. [32, 52, 53]). Thus, for simplicity, in the rest of this part, we mention the RL problems as those in online mode, and abbreviate the RL problems in batch mode by Batch RL.

In the context of Batch RL, there are some popular algorithms to deal with these problems such as Fitted Q-Iteration (FQI) [32, 122], Least Square Policy Iteration (LSPI) [113, 52, 122] which are developed from the standard algorithms of DP such as Approximate Value Iteration (AVI) [8, 105], Approximate Policy Iteration (API) [13, 113] respectively. These Batch RL algorithms are achieved by combining RL and DP methods with approximated function. While FQI can be seen as approximate value iteration for the state-action value functions, LSPI combines state-action value function approximation with linear representation and approximate policy iteration.

4.2 Motivation

It is worth remembering that the problem of minimizing a norm of the empirical OBR is nonconvex, nonsmooth. In the literature, there are the rare algorithms for nonconvex, nonsmooth programming framework. One of them is DC (Difference of Convex functions) programming and DCA (DC Algorithm) (see [66, 93, 95, 96, 97] and the references in [56]) which are powerful, nonsmooth, nonconvex optimization tools. In this dissertation, we will tackle this resulting nonconvex problem by DC programming and DCA. Moreover, our work is also motivated by the fact that DCA has been successfully applied to many (smooth or nonsmooth) large-scale nonconvex programs in various domains of applied sciences, in particular in machine learning (see the list of references in [56]). As for the RL problems, the Online DC programming and Online DCA based approach is also investigated recently.

Recall briefly that DCA aims to solve a standard DC program that consists of minimizing a DC function $f = g - h$ (with g and h being convex functions) over a convex set or on the whole space. Here $g - h$ is called a DC decomposition of f, while the convex functions g and h are DC components of f. The main idea of DCA is approximating the second DC component h by its affine minorant and then solving the resulting convex subproblem at each iteration. The construction of DCA is relied on the convex DC components g and h but not the DC function f itself. Hence, for a DC program, each DC decomposition corresponds to a different version of DCA. Since a DC function f has an infinite number of DC decompositions which have crucial impacts on the qualities (speed of convergence, robustness, efficiency, globality of computed solutions,...) of DCA, the search for a "good"DC decomposition is vital from the algorithmic point of views and is the most important key issue while designing DCA for a practical problem. Furthermore, how to efficiently solve convex subproblems in DCA is a crucial question as well (evidently the form of convex subproblems depends on the DC decomposition). In fact, although convex programming has been studied for about a century, an increasing amount of effort has been put recently into developing fast and scalable algorithms to deal with large-scale problems. Moreover, as DCA is a local approach, finding a good starting point is also an important matter to be studied.

In the next chapters, we will present how to develop efficient DCA based algorithms for large-scale Batch RL problems and design Online DCA based algorithms for RL problems with online mode.

Chapter 5

A unified DC programming framework and efficient DCA based approaches for large scale batch Reinforcement Learning[1]

Abstract: *In this chapter, we investigate a powerful nonconvex optimization approach based on DC (Difference of Convex functions) programming and DCA (DC Algorithm) for reinforcement learning problems in batch mode (Batch RL) (i.e., a fixed set of learning experience is given a priori). These problems can be tackled through the problem of finding the zero of the so-called Optimal Bellman Residual. However, there exist so far a few works in the literature following this direction, knowing that it usually leads to a nonconvex optimization problem which is very hard to solve exactly. In this chapter we consider four optimization formulations of this problem that are the minimization of the ℓ_p-norm with $p \in \{1, 2, +\infty\}$ of optimal Bellman residual and the new concave minimization to design these large-scale problems using linear value-function approximation. They are all formulated as DC programs for which four attractive DCA schemes are developed. Exploiting the special structure of the empirical optimal Bellman residual with linear value-function approximation we carefully address the key issues of DCA, namely the effect of DC decompositions, the efficiency of solution methods for the resulting convex subproblem, and the search for good starting points, when designing the four DCA based algorithms. Numerical experiments on various examples of the two benchmarks of Markov decision process problems - Garnet and Gridworld problems, show the efficiency of our approaches in comparison with two existing DCA based algorithms and two state-of-the-art reinforcement learning algorithms.*

1. The material of this chapter is based on the following work:
[1]. Hoai An Le Thi, Vinh Thanh Ho, Tao Pham Dinh. A unified DC Programming Framework and Efficient DCA based Approaches for Large Scale Batch Reinforcement Learning. *Submitted* to the Journal of Global Optimization.

5.1 Our contributions

In this chapter, we investigate a unified DC (Difference of Convex functions) programming based approach in the context of Batch Reinforcement Learning (Batch RL), i.e. a fixed set of learning experience is given a priori. In particular, as mentioned in Chapter 4, we address the problem of finding the zero of the empirical Optimal Bellman Residual (OBR) via linear approximation under the different norms.

DC programming and DCA were studied in [98] for minimizing the ℓ_p-norm of the empirical OBR via linear approximation ($p \geq 1$), in which two DCA schemes, namely DCA_1 and DCA_2, were developed for the case $p = 1$ and $p = 2$ respectively. However, interesting perspectives leaved by the authors of [98] are the choice of DC decompositions and the solution methods for convex subproblems. In this work, these key issues of DC programming and DCA are addressed in deeper ways to design new DCA based algorithms (for the same optimization problems considered in [98]) which are much more efficient than the DCA_1 and DCA_2 on both quality and rapidity. That is one among several contributions of this work.

Our contributions are multiple.

Firstly, we investigate more attractive DCA based algorithms for minimizing the ℓ_p-norm of the empirical OBR in case $p = 1$ and $p = 2$. Exploiting the effect of DC decomposition and the special structure of the resulting convex subproblems, we propose two DCA schemes, named ℓ_1-DCA and ℓ_2-DCA in case $p = 1$ and $p = 2$ respectively, which require solving one linear program (ℓ_1-DCA) and one convex quadratic program (ℓ_2-DCA) at each iteration. It turns out that our algorithm ℓ_1-DCA (resp. ℓ_2-DCA) outperforms DCA_1 (resp. DCA_2) proposed in [98] in terms of both quality and rapidity. For medium MDPs having 100 states, the ratio of gain in terms of rapidity of ℓ_1-DCA versus DCA_1 (resp. ℓ_2-DCA versus DCA_2) is up to 11.5 (resp. 4.17) times, while the ratio of gain in terms of quality of ℓ_1-DCA versus DCA_1 (resp. ℓ_2-DCA versus DCA_2) is up to 70.2% (resp. 84%). Further, the gain increases considerably when the number of states increases.

Secondly, we consider the ℓ_∞-norm of the empirical OBR and develop a DCA scheme (ℓ_∞-DCA) for solving this problem. The ℓ_∞-DCA enjoys interesting convergence properties thanks to the fact that both DC components are polyhedral convex functions, and it requires also one linear program at each iteration. Numerical experiments show that the ℓ_∞-DCA is slightly better than the existing DCA_1 and DCA_2 – the ratio of gain in terms of quality is up to 40.1%.

Thirdly, we propose a new formulation of the OBR *without using the ℓ_∞/ℓ_p-norm*. We consider the problem as finding the zero of a function (the empirical OBR) which has a very special structure and highlight various possible formulations, the link between them and finally introduce a constrained optimization problem enjoying several advantages. This new formulation is in fact a concave minimization problem under linear constraints (which can also be interpreted as minimizing the restricted ℓ_1-norm of the empirical OBR under a polytope). Fortunately, the addition of constraints

does not make the new problem more difficult than the above formulations which are unconstrained minimization problems. On contrary, the new DCA (named cc-DCA), consisting of solving at each iteration one linear program (whose the number of constraints is, interestingly, smaller than that in linear subproblems of ℓ_1-DCA and ℓ_∞-DCA), is more efficient than the existing DCA_1 and DCA_2 in terms of both quality and rapidity. The ratio of gain is up to 131 times on rapidity and 68.3% on quality.

Throughout the chapter, we carefully studied the key issues of DCA, namely the effect of DC decompositions, the efficiency of solution methods for the resulting convex subproblem, and the search for good starting points, when developing the four DCA based algorithms. We are particularly interested in specific DC programs, called *polyhedral DC programs*, where either g or h is polyhedral convex (i.e. the maximum of a finite family of affine functions), because that DC programs have interesting optimality and convergence properties. More precisely, we exploit the special structure of each objective function in an efficient way to propose a suitable DC decomposition. The ℓ_1-norm, ℓ_∞-norm and concave formulations of OBR are polyhedral DC programs where both DC components are polyhedral convex functions. The corresponding DCA enjoys, not only the finite convergence property, but also the local optimality of solutions. Furthermore, we study in a deep way the solution methods for the convex subproblems in DCA by deriving benefit from the specific structure of the convex subproblem in each DCA scheme. Actually, our techniques reduce to solving one linear program in ℓ_1-DCA, ℓ_∞-DCA, and cc-DCA, and one convex quadratic program in ℓ_2-DCA. Thus, these problems can be efficiently solved by standard softwares. On another hand, we investigate an efficient strategy for finding good starting points for all our DCA based algorithms.

Finally, we provide several numerical experiments of the proposed algorithms on two benchmarks - the stationary Garnet problem and the Gridworld problem, compared with two existing DCA schemes and two state-of-the-art RL algorithms.

The rest of the chapter is organized as follows. Different optimization formulations of the empirical OBR are described in Section 5.2. Section 5.3 first presents a short introduction of DC programming and DCA and then shows how to apply DC programming and DCA to these optimization problems. Section 5.4 reports the numerical results on several test problems. Finally, Section 5.5 concludes the chapter.

5.2 Optimization formulations of the empirical OBR via linear function approximation

Linear function approximators are the most commonly used in function approximation techniques (for instance, SARSA, Q-learning, and Least-Squares Policy Iteration LSPI). A tutorial on linear function approximators for DP and RL was recently given in [41]. The basic idea is that, the full set of states is projected into a lower dimensional space where the value function is represented as a linear function.

Given a basic function vector $\phi(\cdot) = (\phi_1(\cdot), \phi_2(\cdot)..., \phi_d(\cdot))^\top \in \mathbb{R}^d$ where $\phi_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, i = 1, \ldots, d$ are basic functions on $\mathcal{S} \times \mathcal{A}$. Then the linear approximation of the function Q, denoted by Q_θ, is characterized by the weight vector $\theta \in \mathbb{R}^d$ according to the relation

$$Q_\theta(s,a) := \langle \theta, \phi(s,a) \rangle, \forall (s,a) \in \mathcal{S} \times \mathcal{A}.$$

Let $\mathcal{F}$ denote the space of approximation functions, say

$$\mathcal{F} := \left\{ Q_\theta(s,a) = \langle \theta, \phi(s,a) \rangle | \theta \in \mathbb{R}^d \right\}.$$

5.2.1 ℓ_p-norm formulation ($p \geq 1$)

The ℓ_p-norm ($p \geq 1$) of OBR with the probability distribution μ is defined as (see e.g. [86])

$$J_{p,\mu}(Q) := ||B^*Q - Q||_{p,\mu}, \text{ where } ||Q||_{p,\mu} := \left(\sum_{(s,a) \in \mathcal{S} \times \mathcal{A}} \mu(s,a) |Q(s,a)|^p \right)^{\frac{1}{p}}.$$

Performance bounds of a policy greedy with respect to the state value function V in terms of the ℓ_p-norm ($1 \leq p \leq \infty$) of its OBR were given in [86], a similar result was stated later in [98] when $p \geq 1$ for the state-action value function Q. It is well known that Q^* is the optimal solution to $J_{p,\mu}$ and, obviously, if the minimum value of $J_{p,\mu}$ is near zero, the corresponding greedy policy is close to the optimal policy.

With the linear approximation, the ℓ_p-norm ($p \geq 1$) of the empirical OBR over the space of linear approximation functions $\mathcal{F}$ is

$$J_{p,\mu}(Q_\theta) = ||B^*Q_\theta - Q_\theta||_{p,\mu}. \tag{5.1}$$

In batch mode (Batch RL), estimating the optimal state-action value function rests on a fixed set of transition samples, i.e. the so-called sampling-based version of $J_{p,\mu}$ in [86] is used. Specifically, one first selects N samples $(S_i, A_i, S'_i)_{i=1,\ldots,N}$, then considers a nonbiased estimation of $B^*Q(S_i, A_i)$:

$$\widehat{B}^*Q(S_i, A_i) = \mathcal{R}(S_i, A_i) + \gamma \max_{a' \in \mathcal{A}} Q(S'_i, a') \tag{5.2}$$

and finally minimizes the empirical ℓ_p-norm of OBR [86, 98]:

$$J_{p,\mu_N}(Q) := ||\widehat{B}^*Q - Q||_{p,\mu_N}, \tag{5.3}$$

where μ_N is the empirical distribution given by $\mu_N(s,a) = \frac{1}{N} \sum_{i=1}^N \mathbb{1}_{\{(S_i,A_i)=(s,a)\}}(s,a)$ and $\mathbb{1}_X$ is the indicator function of X, i.e. $\mathbb{1}_X(x) = 1$ if $x \in X$, 0 otherwise.

The bound error between the minimized $J_{p,\mu}$ and J_{p,μ_N} in terms of μ_N and the capacity (the complexity) measure of the approximation function space (the Vapnik

consistency) was established in Statistical learning theory [123]. In particular, the Vapnik-consistency of (5.1) with respect to μ_N was deduced in [98] from the results of Theorem 5.3 of [123].

Using this sampling-based technique with linear approximation operator, we have to minimize the empirical ℓ_p-norm ($p \geq 1$) of the empirical OBR that is

$$\left(\sum_{i=1}^{N} \left| \widehat{B}^* Q_\theta(S_i, A_i) - Q_\theta(S_i, A_i) \right|^p \right)^{\frac{1}{p}}.$$

Finally the ℓ_p-norm formulation of the empirical OBR via linear approximation takes the form

$$\min_{\theta \in \mathbb{R}^d} \left\{ F_{p,\mu_N}(\theta) := \left(\sum_{i=1}^{N} \left| \mathcal{R}(S_i, A_i) + \gamma \max_{a' \in \mathcal{A}} \ \langle \theta, \phi(S_i', a') \rangle - \langle \theta, \phi(S_i, A_i) \rangle \right|^p \right) \right\}. \quad (5.4)$$

Let f_i, for $i = 1, \ldots, N$, denote the real function on $\mathbb{R}^d$ defined as

$$f_i(\theta) := \mathcal{R}(S_i, A_i) + \gamma \max_{a' \in \mathcal{A}} \ \langle \theta, \phi(S_i', a') \rangle - \langle \theta, \phi(S_i, A_i) \rangle. \quad (5.5)$$

Clearly, f_i is the maximal function of a finite family of affine functions and can be expressed as (remember that $\mathcal{A}$ is a finite space having $N_\mathcal{A}$ elements)

$$f_i(\theta) := \max_{j=1,\ldots,N_\mathcal{A}} \left\{ \langle \mathbf{A}^{(i,j)}, \theta \rangle + \mathbf{b}^{(i)} \right\}, \quad (5.6)$$

where $\mathbf{A}^{(i,j)} := \gamma \phi(S_i', a_j) - \phi(S_i, A_i),\ \mathbf{b}^{(i)} := \mathcal{R}(S_i, A_i),\ j = 1, \ldots, N_\mathcal{A}$.

Specifically, the ℓ_1-norm optimization formulation is expressed as follows

$$\min \left\{ F_1(\theta) = \sum_{i=1}^{N} |f_i(\theta)| : \theta \in \mathbb{R}^d \right\}, \quad (5.7)$$

and the ℓ_2-norm optimization formulation takes the form

$$\min \left\{ F_2(\theta) := \sum_{i=1}^{N} [f_i(\theta)]^2 : \theta \in \mathbb{R}^d \right\}. \quad (5.8)$$

As each f_i is a polyhedral convex function, it is obvious that F_1 and F_2 are DC functions. In [98], the authors highlighted one DC formulation for each of above DC programs (5.7), (5.8) and proposed two DCA schemes for solving them. In Section 5.3, we will propose more attractive DCA based algorithms for these same problems.

5.2.2 ℓ_∞-norm formulation

With the aim of intensively studying DCA based approaches for the empirical OBR, we consider the ℓ_∞-norm formulation of OBR, which is defined by

$$J_\infty(Q) = ||B^*Q - Q||_\infty, \quad (5.9)$$

where $||Q||_\infty = \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} |Q(s,a)|$. Evidently, the optimal state-action value function Q^* is the optimal solution to (5.9).

Similarly to ℓ_p-norm, after collecting N transition samples (S_i, A_i, S_i') and approximating the state-action value function $Q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ by the weight vector $\theta \in \mathbb{R}^d$ on the space $\mathcal{F}$, we obtain the ℓ_∞-norm optimization formulation of the empirical OBR on $\mathcal{F}$ as follows

$$\min_{\theta\in\mathbb{R}^d} \left\{ F_\infty(\theta) := \max_{i=1,\ldots,N} \left| \widehat{B}^* Q_\theta(S_i, A_i) - Q_\theta(S_i, A_i) \right| = \max_{i=1,\ldots,N} |f_i(\theta)| \right\}. \tag{5.10}$$

It is worth noting that the ℓ_∞-norm in Q is "equivalent" to the ℓ_p-norm on the finite state-action space $\mathcal{S}\times\mathcal{A}$ in the sense that for all $Q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, we have

$$\overline{C}^{\frac{1}{p}} ||Q||_\infty \leq ||Q||_{\mu,p} \leq ||Q||_\infty,$$

where $\overline{C} = \min\{\mu(s,a) : \mu(s,a) > 0, \forall (s,a) \in \mathcal{S}\times\mathcal{A}\}$. Thus, the bound of (5.1) can be derived from the value of (5.9) and moreover the Vapnik-consistency for the ℓ_∞-norm problem (5.9) can be guaranteed.

We will see later in Section 5.3 that (5.10) is also nonconvex, nonsmooth but a DC program.

5.2.3 New formulation: concave minimization under linear constraints

Recall that the main goal of MDP problems is to find the optimal state-action value function Q^* such that $B^*Q^*(s,a) - Q^*(s,a) = 0$ for all $(s,a) \in \mathcal{S}\times\mathcal{A}$, which implies

$$\sum_{(s,a)} [B^*Q^*(s,a) - Q^*(s,a)] = 0.$$

We are then suggested to consider the following optimization problem:

$$0 = \min_{Q\in\mathbb{R}^{\mathcal{S}\times\mathcal{A}}} \left\{ m(Q) := -\sum_{(s,a)} [B^*Q(s,a) - Q(s,a)] : B^*Q(s,a) - Q(s,a) \leq 0,\ \forall (s,a) \right\}. \tag{5.11}$$

The following result (whose proof is straightforward) justifies our formulation.

Proposition 5.1. *The optimal state-action value function Q^* is a unique optimal solution to* (5.11) *with the optimal objective value being zero.*

After collecting N transition samples (S_i, A_i, S_i') and approximating the state-action value function $Q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ by the weight vector $\theta \in \mathbb{R}^d$ on the space of approximation function $\mathcal{F}$, the problem (5.11) becomes

$$0 = \min_{\theta\in\mathbb{R}^d} \left\{ F_{\text{cc}}(\theta) := -\sum_{i=1}^{N} f_i(\theta) : f_i(\theta) \leq 0, \forall i = 1,\ldots,N \right\}. \tag{5.12}$$

This problem can be equivalently reformulated as

$$0 = \min_{\theta \in \mathbb{R}^d} \left\{ F_{\text{cc}}(\theta) := -\sum_{i=1}^{N} f_i(\theta) : f_i^+(\theta) \leq 0, \forall i = 1, \ldots, N \right\}, \tag{5.13}$$

where f_i^+ is the function defined by $f_i^+(\theta) := \max\{0, f_i(\theta)\}$.

As each f_i is a polyhedral convex function, the function F_{cc} is concave, and the problems (5.12) and (5.13) are concave minimization problems under polyhedral convex constraints (as will be seen later, they are in fact polyhedral DC programs). Obviously, the exact penalty holds for (5.13) (see [70]), i.e. there exists the parameter $\tau_0 > 0$ such that for all $\tau \geq \tau_0$, the problem (5.13) is equivalent to

$$0 = \min_{\theta \in \mathbb{R}^d} \left\{ -\sum_{i=1}^{N} f_i(\theta) + \tau f_{\max}^+(\theta) \right\},$$

where $f_{\max}$ is the function defined by $f_{\max}(\theta) := \max\{f_i(\theta) : i = 1, \ldots, N\}$.

In this work we focus on the problem (5.12) because it enjoys several advantages for DCA as will be shown in the next section. Let us denote $\mathcal{C}_\theta$ the feasible set of (5.12). According to (5.6) we have

$$\begin{aligned} \mathcal{C}_\theta &:= \left\{ \theta \in \mathbb{R}^d : \max_{j=1,\ldots,N_{\mathcal{A}}} \left\{ \langle \mathbf{A}^{(i,j)}, \theta \rangle + \mathbf{b}^{(i)} \right\} \leq 0, \ \forall i = 1, \ldots, N \right\} \\ &= \left\{ \theta \in \mathbb{R}^d : \langle \mathbf{A}^{(i,j)}, \theta \rangle + \mathbf{b}^{(i)} \leq 0, \ \forall i = 1, \ldots, N, \ \forall j = 1, \ldots, N_{\mathcal{A}} \right\}. \end{aligned}$$

$\mathcal{C}_\theta$ is so a polyhedral convex set defined by $N.N_{\mathcal{A}}$ linear constraints, and (5.12) can be now expressed as

$$0 = \min \left\{ F_{\text{cc}}(\theta) := -\sum_{i=1}^{N} f_i(\theta) : \langle \mathbf{A}^{(i,j)}, \theta \rangle + \mathbf{b}^{(i)} \leq 0, \ \forall i = 1, \ldots, N, \ \forall j = 1, \ldots, N_{\mathcal{A}} \right\} \tag{5.14}$$

which is a concave minimization problem under linear constraints.

Remark 5.1. *As $|f_i(\theta)| = -f_i(\theta)$ for all $\theta \in \mathcal{C}_\theta$, we have $F_1(\theta) = F_{\text{cc}}(\theta)$ for all $\theta \in \mathcal{C}_\theta$. Hence, the formulation (5.12) can be viewed as the ℓ_1-norm formulation restricted under $\mathcal{C}_\theta$, in case μ is the uniform distribution.*

From Remark 5.1 and Theorem 2 in [98], we have the Vapnik-consistency for the formulation (5.12) as showing the following lemma.

Lemma 5.1. *Let $\overline{\mathcal{F}} = \left\{ Q_\theta \in \mathcal{F} : ||Q_\theta||_\infty \leq \frac{||\mathcal{R}||_\infty}{1-\gamma}, B^* Q_\theta - Q_\theta \leq 0 \right\}$, $\eta \in (0,1)$ and consider the finite deterministic MDP, with probability at least $1-\eta$, we have:*

$$\forall Q_\theta \in \overline{\mathcal{F}}, \ \frac{1}{N_{\mathcal{S}}.N_{\mathcal{A}}} m(Q_\theta) \leq \frac{1}{N} F_{\text{cc}}(\theta) + \frac{2||\mathcal{R}||_\infty}{1-\gamma} \sqrt{\varepsilon(N)},$$

where $\varepsilon(N) = \dfrac{h\left(\ln\left(\frac{2N}{h}\right) + 1\right) + \ln\left(\frac{4}{\eta}\right)}{N}$ *and* $h = 2N_{\mathcal{A}}(d+1)$.

Proof. Based on the Vapnik-consistency for the formulation (5.1) from Theorem 2 in [98], this lemma is obtained in case $p = 1$, μ is the uniform distribution over the whole state-action space $\mathcal{S} \times \mathcal{A}$ and μ_N is the empirical distribution with respect to N transition samples. □

Remark 5.2. *The problem* (5.12) *is actually a concave minimization with linear constraints for which the use of DCA is worthy. In fact, DCA applied on linear constrained concave minimization, with the natural DC decomposition, consists of solving one linear program at each iteration, and has finite convergence property. We will show in Section 5.3.5 that* (5.12) *is more advantageous than the previous formulations and the corresponding DCA based algorithm is very efficient in terms of rapidity.*

We are going to present DC programming and DCA for solving the above problems (5.7), (5.8), (5.10) and (5.12).

5.3 Solution methods by DC programming and DCA

5.3.1 DCA for solving the ℓ_1-norm problem (5.7)

Remember that the problem (5.7) has the form

$$\min \left\{ F_1(\theta) = \sum_{i=1}^{N} |f_i(\theta)| : \theta \in \mathbb{R}^d \right\}, \text{ where } f_i(\theta) := \max_{j=1,\ldots,N_{\mathcal{A}}} \langle \theta, \mathbf{A}^{(i,j)} \rangle + \mathbf{b}^{(i)}.$$

As mentioned previously, the function f_i is polyhedral convex. By taking full advantage of this nice property, we construct a DC decomposition of $|f_i|$ as follows:

$$|f_i| = 2f_i^+ - f_i,$$

where $f_i^+ := \max\{0, f_i\}$ is polyhedral convex too. Let G_1 and H_1 be the functions defined by

$$G_1(\theta) := \sum_{i=1}^{N} 2f_i^+(\theta) \text{ and } H_1(\theta) := \sum_{i=1}^{N} f_i(\theta), \tag{5.15}$$

they are both polyhedral convex functions. Clearly,

$$F_1(\theta) = G_1(\theta) - H_1(\theta),$$

therefore the problem (5.7) can be now written in the form

$$\min \left\{ G_1(\theta) - H_1(\theta) : \theta \in \mathbb{R}^d \right\}, \tag{5.16}$$

which is a polyhedral DC program where both DC components are polyhedral convex.

According to the generic DCA scheme, DCA applied on (5.16) consists of, at each iteration k, computing one subgradient $w^k \in \partial H_1(\theta^k)$, and then solving the following convex program:

$$\min\left\{\sum_{i=1}^{N} 2f_i^+(\theta) - \langle w^k, \theta\rangle : \theta \in \mathbb{R}^d\right\}. \tag{5.17}$$

Thanks to the polyhedral convexity of f_i defined by (5.5), we can reformulate the last problem as a linear program of the form

$$\begin{cases} \min \sum_{i=1}^{N} 2t_i - \langle w^k, \theta\rangle, \\ \text{s.t. } \theta \in \mathbb{R}^d, \\ \quad t_i \geq 0, \forall i = 1, \ldots, N, \\ \quad t_i \geq \langle \mathbf{A}^{(i,j)}, \theta\rangle + \mathbf{b}^{(i)}, \forall i = 1, \ldots, N, \forall j = 1, \ldots, N_{\mathcal{A}}. \end{cases} \tag{5.18}$$

Compute ∂H_1: by the definition of H_1 and according to the rule of computing the subdifferential of a function being the maximum of a finite family of convex functions [101] we have

$$\begin{aligned} \partial H_1(\theta) &= \sum_{i=1}^{N} \partial f_i(\theta) = \sum_{i=1}^{N} \partial\left[\max_{j=1,\ldots,N_{\mathcal{A}}} \langle \mathbf{A}^{(i,j)}, \theta\rangle + \mathbf{b}^{(i)}\right] \\ &= \sum_{i=1}^{N} \text{co}\big\{\mathbf{A}^{(i,j_i)} : j_i \in I_i(\theta)\big\}, \\ I_i(\theta) &= \text{argmax}_{\ j=1,\ldots,N_{\mathcal{A}}} \langle \mathbf{A}^{(i,j)}, \theta\ \rangle. \end{aligned} \tag{5.19}$$

In particular, we can choose $w^k \in \partial H_1(\theta^k)$ as follows: for $i = 1, \ldots, N$, let $j_i \in I_i(\theta^k)$, then

$$w^k = \sum_{i=1}^{N} \mathbf{A}^{(i,j_i)}. \tag{5.20}$$

Define $t = (t_1, \ldots, t_N)^\top \in \mathbb{R}^N$. DCA applied to (5.7) can be summarized in Algorithm 5.1 (ℓ_1-DCA) below.

Algorithm 5.1 DCA for solving (5.16) (ℓ_1-DCA)

Initialization: Let ε be a sufficiently small positive number. Let $\theta^0 \in \mathbb{R}^d$. Set $k = 0$.
repeat
1. Compute $w^k \in \partial H_1(\theta^k)$ using (5.20).
2. Solve the linear program (5.18) to obtain (θ^{k+1}, t^{k+1}).
3. $k = k + 1$.

until $|F_1(\theta^k) - F_1(\theta^{k-1})| \leq \varepsilon(|F_1(\theta^{k-1})| + 1)$ or $||\theta^k - \theta^{k-1}||_2 \leq \varepsilon(1 + ||\theta^{k-1}||_2)$.

According to the convergence properties of DCA for polyhedral DC programs in Section 1.1, we deduce the following interesting convergence properties of ℓ_1-DCA.

Theorem 5.1. *Convergence properties of* ℓ_1-DCA
i) ℓ_1-DCA *generates the sequence* $\{\theta^k\}$ *such that the sequence* $\{F_1(\theta^k)\}$ *is decreasing.*
ii) The sequence $\{\theta^k\}$ *converges to a critical point* θ^* *of* (5.16) *after a finite number of iterations.*
iii) θ^* *is almost always a local minimizer to* (5.16)*. In particular, if* $I_i(\theta^*)$ *is a singleton for all* $i = 1, ..., N$*, then* θ^* *is a local minimizer to* (5.16)*.*

Proof. (i) and (ii) are, respectively, direct consequences of the convergence properties of general DC programs and polyhedral DC programs.
(iii) Since H_1 is a polyhedral convex function, the necessary local optimality condition $\partial H_1(\theta^*) \subset \partial G_1(\theta^*)$ is also sufficient. This inclusion holds when H_1 is differentiable at θ^*. As a polyhedral convex function is almost always differentiable, we deduce that θ^* is almost always a local minimizer to (5.16). In particular, from the computation of ∂H_1 we see that if $I_i(\theta^*)$ is a singleton for all $i = 1, ..., N$ then H_1 is differentiable. The proof is then complete. □

Remark 5.3. *Our* ℓ_1-DCA *algorithm is much more advantageous than* DCA_1 *in [98]:* ℓ_1-DCA *solves one linear program at each iteration while* DCA_1 *uses the iterative subgradient method to deal with nonsmooth convex subproblems.*

5.3.2 DCA for solving the ℓ_2-norm problem (5.8)

Recall that the ℓ_2-norm problem (5.8) is of the form:

$$\min_{\theta\in\mathbb{R}^d} \left\{ F_2(\theta) := \sum_{i=1}^{N} [f_i(\theta)]^2 \right\}.$$

Similarly, we thoroughly exploit the convexity of the function f_i to design DCA. For each $i = 1, \ldots, N$, with $\overline{a}_i \in \text{dom } \partial f_i$ and $\overline{b}_i \in \partial f_i(\overline{a}_i)$, we define the affine minorization of f_i as follows

$$l_i(\theta) = f_i(\overline{a}_i) + \langle \theta - \overline{a}_i, \overline{b}_i \rangle.$$

Let $l^-(\theta) = \max\{0, -l(\theta)\}$. For $\theta \in \mathbb{R}^d$, we have $f_i(\theta) = \left[f_i(\theta) + l_i^-(\theta)\right] - l_i^-(\theta)$, which implies

$$[f_i(\theta)]^2 = 2\left\{ \left[f_i(\theta) + l_i^-(\theta)\right]^2 + \left[l_i^-(\theta)\right]^2 \right\} - \left[f_i(\theta) + 2l_i^-(\theta)\right]^2.$$

Hence, we derive the following DC decomposition of F_2:

$$F_2(\theta) = G_2(\theta) - H_2(\theta), \tag{5.21}$$

where

$$G_2(\theta) = \sum_{i=1}^{N} 2\left\{ \left[f_i(\theta) + l_i^-(\theta)\right]^2 + \left[l_i^-(\theta)\right]^2 \right\},\ H_2(\theta) = \sum_{i=1}^{N} \left[f_i(\theta) + 2l_i^-(\theta)\right]^2. \tag{5.22}$$

It is evident that $f_i + l_i^-$ and l_i^- are nonnegative and convex on $\mathbb{R}^d$, so are G_2 and H_2. Hence (5.8) is a DC program of the form

$$\min\{G_2(\theta) - H_2(\theta) : \theta \in \mathbb{R}^d\}. \tag{5.23}$$

Applying DCA on (5.23) leads us to compute two sequences $\{w^k\}$ and $\{\theta^k\}$ such that $w^k \in \partial H_2(\theta^k)$ and θ^{k+1} solves the following convex program of the form (P_k)

$$\min\left\{\sum_{i=1}^{N} 2\left\{\left[f_i(\theta)+l_i^-(\theta)\right]^2+\left[l_i^-(\theta)\right]^2\right\}-\langle w^k,\theta\rangle : \theta\in\mathbb{R}^d\right\}.$$

The last problem can be reformulated as

$$\min\left\{\sum_{i=1}^{N}(2\tau_i^2)+\sum_{i=1}^{N}(2t_i^2)-\langle w^k,\theta\rangle : \begin{array}{l} t_i \geq l_i^-(\theta), \forall i=1,\ldots,N,\\ \tau_i \geq t_i+f_i(\theta), \forall i=1,\ldots,N\end{array}\right\}$$

which is in fact a quadratic program of the form

$$\left\{\begin{array}{l} \min \sum_{i=1}^{N}(2\tau_i^2)+\sum_{i=1}^{N}(2t_i^2)-\langle w^k,\theta\rangle,\\ \text{s.t. } \theta\in\mathbb{R}^d,\\ \quad t_i\geq 0, \forall i=1,\ldots,N,\\ \quad t_i\geq -l_i(\theta), \forall i=1,\ldots,N,\\ \quad \tau_i \geq t_i+\langle\theta,\mathbf{A}^{(i,j)}\rangle+\mathbf{b}^{(i)}, \forall i=1,\ldots,N, \forall j=1,\ldots,N_{\mathcal{A}}.\end{array}\right. \tag{5.24}$$

Compute ∂H_2 : from the definition of H_2 and l_i^-, we have

$$\partial H_2(\theta)=\sum_{i=1}^{N}\partial\left[f_i(\theta)+2l_i^-(\theta)\right]^2,$$

and for $i=1,\ldots,N$,

$$\partial l_i^-(\theta)=\left\{\begin{array}{ll}\{0\} & \text{if } l_i(\theta)>0,\\ \left[0,-\overline{b}_i\right] & \text{if } l_i(\theta)=0,\\ \{-\overline{b}_i\} & \text{if } l_i(\theta)<0.\end{array}\right.$$

Here $[0,-\overline{b}_i]$ is the line segment between 0 and $-\overline{b}_i$. Hence, we can take a subgradient $w^k\in\partial H_2(\theta^k)$ as follows

$$w^k=2\sum_{i=1}^{N}\left[f_i(\theta^k)+2l_i^-(\theta^k)\right]\left[\mathbf{A}^{(i,j_i)}-\overline{b}_i.\mathbb{1}_{\{l_i(\theta)<0\}}(\theta^k)\right], \tag{5.25}$$

where $j_i\in I_i(\theta^k),\ \forall i=1,\ldots,N$ and $I_i(\theta)$ is defined in (5.19).

Define $\overline{\tau}=(\tau_1,\ldots,\tau_N)^\top\in\mathbb{R}^N$. Finally, DCA applied to (5.23) can be described as Algorithm 5.2 (ℓ_2-DCA).

As direct consequences of convergence properties of DCA in Section 1.1, we have

Theorem 5.2. *Convergence properties of* ℓ_2-DCA

i) ℓ_2-DCA *generates the sequence* $\{\theta^k\}$ *such that the sequence* $\{F_2(\theta^k)\}$ *is decreasing.*

ii) If the optimal value of problem (5.23) *is finite then the sequence* $\{\theta^k\}$ *converges to* θ^* *that is a critical point of* (5.23).

Algorithm 5.2 DCA for solving (5.23) (ℓ_2-DCA)

Initialization: Let ε be a sufficiently small positive number. Let $\theta^0 \in \mathbb{R}^d$. Set $k = 0$.
repeat
1. Compute $w^k \in \partial H_2(\theta^k)$ using (5.25).
2. Solve the quadratic program (5.24) to obtain $(\theta^{k+1}, \overline{\tau}^{k+1}, t^{k+1})$.
3. $k = k + 1$.

until $|F_2(\theta^k) - F_2(\theta^{k-1})| \leq \varepsilon(|F_2(\theta^{k-1})| + 1)$ or $||\theta^k - \theta^{k-1}||_2 \leq \varepsilon(1 + ||\theta^{k-1}||_2)$.

Remark 5.4. *The DC decomposition (5.21)-(5.22) enjoys several advantages: firstly, the affine minorization l_i of f_i is easily computed, therefore the DC components G_2 and H_2 are defined in a simple way. Such a DC decomposition is possible thanks to the convexity of f_i. Secondly, the function l_i can be updated at each iteration to better approximate f_i, which means we can use a local DC decomposition of F_2 at each iteration. In our numerical experiments (Section 5.4), $\overline{a}_i$ and $\overline{b}_i$ are updated during iterations as follows: at iteration k of ℓ_2-DCA, we take $\overline{a}_i = \theta^k$ and $\overline{b}_i = \mathbf{A}^{(i,j_i)}$, where $j_i \in I_i(\theta^k), \forall i = 1, \ldots, N$ and I_i is defined as* (5.19). *Thirdly, the resulting convex subproblem is reformulated as a quadratic program that can be solved by standard softwares. Such a reformulation is feasible by dint of the polyhedral convexity of f_i. We thus see a good exploitation of the properties of f_i in the construction of DC decomposition and of the corresponding* DCA. *By the way, from a numerical point of view, our ℓ_2-*DCA *algorithm is more efficient than* DCA_2 *in [98] which uses the subgradient method to solve nonsmooth convex subproblems.*

5.3.3 DCA for solving the ℓ_∞-norm problem (5.10)

The ℓ_∞-norm problem (5.10) is written as follows.

$$\min \left\{ F_\infty(\theta) = \max_{i=1,\ldots,N} |f_i(\theta)| : \theta \in \mathbb{R}^d \right\}.$$

For $\theta \in \mathbb{R}^d$ and $i = 1, \ldots, N$, we have

$$|f_i(\theta)| = 2f_i^+(\theta) - f_i(\theta) = \left(2f_i^+(\theta) + \sum_{j=1, j\neq i}^{N} f_j(\theta)\right) - \left(\sum_{j=1}^{N} f_j(\theta)\right).$$

Let

$$G_3(\theta) := \max_{i=1,\ldots,N} \left(2f_i^+(\theta) + \sum_{j=1, j\neq i}^{N} f_j(\theta)\right).$$

Then G_3 is convex and therefore we get the following DC formulation of (5.10):

$$\min\{F_\infty(\theta) := G_3(\theta) - H_1(\theta) : \theta \in \mathbb{R}^d\}, \tag{5.26}$$

where both DC components are polyhedral convex. Hence, the problem (5.10) is a polyhedral DC program. Similarly, DCA applied to (5.26) consists of, at each iteration

k, first calculating a subgradient $w^k \in \partial H_1(\theta^k)$ and then solving the following convex program:

$$\min_{\theta\in\mathbb{R}^d}\left\{\max_{i=1,\ldots,N}\left(2f_i^+(\theta)+\sum_{j=1,j\neq i}^{N} f_j(\theta)\right)-\langle w^k,\theta\rangle\right\}. \tag{5.27}$$

The problem (5.27) can be equivalently reformulated as the next linear program:

$$\begin{cases} \min\ \hat{t}-\langle w^k,\theta\rangle, \\ \text{s.t. } \theta\in\mathbb{R}^d, \\ \quad \sum\limits_{j=1,j\neq i}^{N} t_j-\hat{t}\leq 0, \forall i=1,\ldots,N, \\ \quad 2t_i+\sum\limits_{j=1,j\neq i}^{N} t_j-\hat{t}\leq 0, \forall i=1,\ldots,N, \\ \quad \langle\theta,\mathbf{A}^{(i,j)}\rangle+\mathbf{b}^{(i)}\leq t_i, \forall i=1,\ldots,N, \forall j=1,\ldots,N_{\mathcal{A}}. \end{cases} \tag{5.28}$$

Finally, DCA applied to (5.26) is described in Algorithm 5.3 (ℓ_∞-DCA) below.

Algorithm 5.3 DCA for solving (5.26) (ℓ_∞-DCA)

Initialization: Let ε be a sufficiently small positive number. Let $\theta^0\in\mathbb{R}^d$. Set $k=0$.
repeat
1. Compute $w^k\in\partial H_1(\theta^k)$ using (5.20).
2. Solve the linear program (5.28) to obtain $(\theta^{k+1},\hat{t}^{k+1},t^{k+1})$.
3. $k=k+1$.

until $|F_\infty(\theta^k)-F_\infty(\theta^{k-1})|\leq\varepsilon(|F_\infty(\theta^{k-1})|+1)$ or $||\theta^k-\theta^{k-1}||_2\leq\varepsilon(1+||\theta^{k-1}||_2)$.

Similarly to Theorem 5.1, we have

Theorem 5.3. *Convergence properties of* ℓ_∞-DCA
i) ℓ_∞-DCA *generates the sequence* $\{\theta^k\}$ *such that the sequence* $\{F_\infty(\theta^k)\}$ *is decreasing.*
ii) The sequence $\{\theta^k\}$ *converges to a critical point* θ^* *of* (5.26) *after a finite number of iterations.*
iii) θ^* *is almost always a local minimizer to* (5.26)*. In particular, if* $I_i(\theta^*)$ *is a singleton for all* $i=1,...,N$*, then* θ^* *is a local minimizer to* (5.26)*.*

5.3.4 DCA applied on the new concave minimization formulation (5.14)

As (5.14) is a concave minimization problem with linear constraints, the following DC formulation is the most natural

$$\min\{F_{cc}(\theta):=\chi_{\mathcal{C}_\theta}(\theta)-H_1(\theta):\theta\in\mathbb{R}^d\}, \tag{5.29}$$

Obviously, the problem (5.29) is a polyhedral DC program where both DC components are polyhedral convex functions.

DCA applied to (5.29) consists in computing two sequences $\{w^k\}$ and $\{\theta^k\}$ such that

$$\begin{aligned} & w^k \in \partial H_1(\theta^k), \quad \theta^{k+1} \in \operatorname{argmin}\left\{\chi_{\mathcal{C}_\theta}(\theta) - \langle w^k, \theta\rangle : \theta \in \mathbb{R}^d\right\} \\ \Leftrightarrow \quad & \theta^{k+1} \in \operatorname{argmin}\left\{-\langle w^k, \theta\rangle : \langle \mathbf{A}^{(i,j)}, \theta\rangle + \mathbf{b}^{(i)} \leq 0,\ i = 1,\ldots,N,\ j = 1,\ldots,N\right\} \end{aligned} \tag{5.30}$$

Algorithm 5.4 DCA for solving (5.29) (cc-DCA)

Initialization: Let ε be a sufficiently small positive number. Let $\theta^0 \in \mathbb{R}^d$. Set $k = 0$.
repeat
1. Compute $w^k \in \partial H_1(\theta^k)$ using (5.20).
2. Solve the linear program (5.30) to obtain θ^{k+1}.
3. $k = k + 1$.

until $|F_{\text{cc}}(\theta^k) - F_{\text{cc}}(\theta^{k-1})| \leq \varepsilon(|F_{\text{cc}}(\theta^{k-1})| + 1)$ or $||\theta^k - \theta^{k-1}||_2 \leq \varepsilon(1 + ||\theta^{k-1}||_2)$.

As (5.29) is a polyhedral DC program where both DC components are polyhedral convex, we have

Theorem 5.4. *Convergence properties of* cc-DCA
i) cc-DCA *generates the sequence* $\{\theta^k\}$ *such that the sequence* $\{F_{cc}(\theta^k)\}$ *is decreasing.*
ii) The sequence $\{\theta^k\}$ *converges to a critical point* θ^* *of* (5.29) *after a finite number of iterations.*
iii) θ^* *is almost always a local minimizer to* (5.29)*. In particular, if* $I_i(\theta^*)$ *is a singleton for all* $i = 1, ..., N$*, then* θ^* *is a local minimizer to* (5.29)*.*

Remark 5.5. *Similarly to the* ℓ_1*-norm problem, our DC formulations for the* ℓ_∞*-norm and the concave minimization problems have the same advantages: both DC components are convex polyhedral, consequently the three algorithms* ℓ_1-DCA, ℓ_∞-DCA *and* cc-DCA *enjoy the same interesting convergence properties.*

5.3.5 Performance analysis on different DCA based algorithms

For the same ℓ_1-norm (resp. ℓ_2-norm) problem, our proposed algorithm ℓ_1-DCA (resp. ℓ_2-DCA) is more efficient than DCA_1 (resp. DCA_2) developed in [98] (see remarks 5.3 and 5.4 above). These advantages show once again the great effect of DC decompositions as well as the one of solution methods for convex subproblems in DCA. Here the convexity and the polyhedral structure of f_i are the key for designing ℓ_1-DCA and ℓ_2-DCA.

Among the four proposed DC formulations, those for the ℓ_1-norm, the ℓ_∞-norm and the concave minimization are similar: they have the same second DC component H_1 and both DC components are polyhedral convex. Consequently the three resulting ℓ_1-DCA, ℓ_∞-DCA and cc-DCA have the same convergence properties (almost always converge to local solutions after a finite number of iterations) and they require solving *one linear program* at each iteration. Hence, in terms of complexity, we can say that

these three algorithms are more efficient than ℓ_2-DCA which consists in solving iteratively one convex quadratic program. Note also that ℓ_2-DCA does not have the finite convergence property as ℓ_1-DCA, ℓ_∞-DCA and cc-DCA. Meanwhile, the DC decomposition (5.23) and the use of *local* DC decompositions in ℓ_2-DCA (see Remark 5.4) are quite interesting, which can result in higher quality of solutions. This observation is confirmed by numerical results in our experiments.

Since the three DC programs of the ℓ_1-norm, the ℓ_∞-norm and the concave minimization problems share the same DC component H_1, the complexity of the resulting DCAs differs from one of others by the complexity of linear subproblems. In this sense, cc-DCA is more advantageous than ℓ_1-DCA and ℓ_∞-DCA. In fact, the linear program in cc-DCA has the smallest size with d variables, $N_{\mathcal{A}}.N$ constraints; the one in ℓ_1-DCA comes next with $d+N$ variables, $(N_{\mathcal{A}}+1).N$ constraints and finally, the one in ℓ_∞-DCA has largest size with $d+N+1$ variables, $(N_{\mathcal{A}}+2).N$ constraints. Thus, cc-DCA should be the fastest, ℓ_1-DCA is the next and the last is ℓ_∞-DCA.

5.3.6 Starting points for DCA

Finding a good starting point is an important issue when designing DCA. For this purpose we convexify the concave function F_{cc} by ignoring the difficult term $\max_{a'\in\mathcal{A}} \langle \theta, \phi(S'_i, a')\rangle$ in each f_i and then consider the convex problem

$$\min \left\{ F_{cv}(\theta) := \sum_{i=1}^{N} Q_\theta(S_i, A_i) : f_i(\theta) \leq 0, \forall i = 1, \ldots, N \right\} \tag{5.31}$$

which can be reformulated as a linear program. When (5.31) admits an optimal solution, we take it as starting point for DCA. Instead, in case (5.31) has no optimal solution, we choose a feasible solution.

5.4 Numerical experiments

In the numerical experiments, we study the performance of the proposed DCA algorithms ℓ_1-DCA, ℓ_2-DCA, ℓ_∞-DCA, cc-DCA and compare them with two existing DCA algorithms DCA_1, DCA_2 [98], and two standard RL algorithms named Fitted Q-Iteration (FQI) [32] and Least Square Policy Iteration (LSPI) [52]. Table 5.1 summarizes the comparative algorithms.

Our experiment is composed of two parts. In the first experiment we study the efficiency of the six DCA based algorithms: the four proposed DCA and the two existing DCA_1, DCA_2 [98] on the Garnet problem [14] which has been tested in [98]. For a fair comparison we use the same procedure to generate Garnet instances as in [98]. We also study the effect of starting points on the proposed DCAs.

Table 5.1 – Summary of all comparative algorithms

Algorithm	Considered problem	Reference
ℓ_1-DCA	ℓ_1-norm (5.7)	our proposal
ℓ_2-DCA	ℓ_2-norm (5.8)	our proposal
ℓ_∞-DCA	ℓ_∞-norm (5.10)	our proposal
cc-DCA	concave (5.12)	our proposal
DCA_1	ℓ_1-norm (5.7)	[98]
DCA_2	ℓ_2-norm (5.8)	[98]
FQI	-	[32]
LSPI	-	[52]

In the second experiment we compare notable DCA algorithms (ℓ_1-DCA, ℓ_2-DCA, and cc-DCA) with the FQI, LSPI algorithms on Garnet problems and the Gridworld problem [1, 92, 118].

5.4.1 Description of Garnet and Gridworld problems

Let us first present the two problems concerning our computational experiments: Garnet and Gridworld.

■ **Garnet**: Stationary Garnet problems, a class of randomly constructed finite MDP (the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are finite), were introduced in [14]. The typical features of a stationary Garnet problem, denoted by Garnet($N_\mathcal{S}$,$N_\mathcal{A}$,N_B), are three parameters: the number of states $N_\mathcal{S}$, the number of actions $N_\mathcal{A}$ and the number of next states N_B for each state-action pair. We consider here a particular type of Garnet, given in [98], which is a topological structure relative to real dynamical systems. In this Garnet, the state space is considered as $\mathcal{S} = \{s_i\}_{i=1,\ldots,N_\mathcal{S}}$ where each state $s_i = (s_i^{(j)})_{j=1,\ldots,m}$ is an m-tuple ($m = 2$) and each component $s_i^{(j)}$ is chosen out of all integer numbers between 1 and x_i ($x_i = K, \forall i = 1, \ldots, N_\mathcal{S}$). Thus, the number of states is $N_\mathcal{S} = \prod_{i=1}^{m} x_i = K^2$. All parameters of Garnet problems are set similarly as in [98]. Specifically, for each state-action pair (s, a), the next states are randomly chosen via the multivariate normal distribution $\mathcal{N}_m(s, I_m)$ where I_m is the identity matrix of size m. The transition probability of going to each next state is generated by dividing the interval $[0, 1]$ at $(N_B - 1)$ random points. The reward for each state-action pair is generated uniformly between -1 and 1. The discount factor γ is set to 0.95. All algorithms use the exact tabular representation and thus, the number of basis functions is $d = N_\mathcal{S}.N_\mathcal{A}$. For each Garnet problem with the transition probability function and the reward function generated as above, the optimal policy π^* is obtained by the policy iteration algorithm (see, e.g., [118]). We test on N_G different Garnet($N_\mathcal{S}$, $N_\mathcal{A}$, N_B) problems $\{G_p\}_{1\leq p\leq N_G}$ and, for each Garnet G_p, construct N_{data} datasets $\{D^{p,q}\}_{1\leq q\leq N_{data}}$ of N transition samples $(S_i, A_i, S_i')_{i=1,\ldots,N}$ drawn uniformly and independently.

■ **Gridworld**: We use a 100×100 Gridworld with sparse rewards as described in

[1, 92, 118]. A cell of the grid corresponds to a state and thus the number of states is $N_{\mathcal{S}} = 10^4$. At each state, there are four possible actions: North, South, East, and West (thus, $N_{\mathcal{A}} = 4$), which make an agent move one cell in the corresponding direction on the grid with some probability p, or in each of three unintended directions with probability $\frac{1-p}{3}$. If the agent gets out of the grid, its place will be unchanged. We divide the grid into non-overlapping regions of 10×10 cells, called *macrocells*, and thus the number of macrocells is $N_{reg} = 100$. For each $i \in \{1, \ldots, N_{reg}\}$ and $j \in \{1, \ldots, N_{\mathcal{A}}\}$, $\phi_{i+N_{reg}(j-1)}(s, a) = 1$ if the state s resides in the ith macrocell and the action a coincides with a_j, 0 otherwise. In this case, the number of basis functions is $d = N_{reg}.N_{\mathcal{A}}$. The reward function is given by $\mathcal{R} = w^\top \phi$, where the weight $w \in \mathbb{R}^d$ is generated randomly as follows: for $i \in \{1, \ldots, d\}$, $w_i = 0$ with probability of 0.9 and w_i is sampled uniformly from $[0, 1]$ with probability of 0.1. Finally, w is renormalized such that $\sum_{i=1,\ldots,d} w_i = 1$. Instances with fewer than two nonzero entries in w are not interesting and discarded. The discounted factor γ and the probability p are set to 0.99 and 0.7, respectively. For each Gridworld problem, we also obtain the optimal policy π^* by the policy iteration algorithm (see, e.g., [118]). We run on N_G Gridworlds $\{Gr_p\}_{1 \leq p \leq N_G}$ and, for each Gr_p, build N_{data} datasets $\{D^{p,q}\}_{1 \leq q \leq N_{data}}$ of N transition samples $(S_i, A_i, S_i')_{i=1,\ldots,N}$ generated uniformly and independently.

5.4.2 Set up experiments

All experiments were implemented in MATLAB R2013b and performed on a PC Intel(R) Xeon(R) CPU E5-2630 v2, @ 2.60GHz of 32GB RAM. The software CPLEX 12.6 was used for solving linear and/or convex quadratic programs. The functions `lsqlin` and `mldivide` in MATLAB were utilized for solving constrained linear least-squares problems in FQI and the system of equations in LSPI, respectively.

The stopping criteria of all DCA based algorithms considered are either

$$||\theta^k - \theta^{k-1}||_2 \leq \varepsilon(1 + ||\theta^{k-1}||_2)$$

or

$$|F(\theta^k) - F(\theta^{k-1})| \leq \varepsilon(|F(\theta^{k-1})| + 1)$$

or the CPU time exceeds 1000 seconds for ℓ_1-norm/ℓ_∞-norm/concave problems and 2000 seconds for ℓ_2-norm problems. The default tolerance is $\varepsilon = 10^{-4}$.

The stopping criteria of the subgradient method for solving convex subproblems in DCA_1 and DCA_2 are either $\|s^k\|_2 \leq \varepsilon$ (s^k is a subgradient of the objective function at θ^k) or the number of iterations does not exceed M. A small value of M implies a premature stopping of subgradient algorithm, in such a case DCA_1 and DCA_2 are not stable in the sense that the sequence of objective function values may not be decreasing, and then the obtained solution may be worse. When M is large, the subgradient method runs very slowly. By compromising, we set M to 100.

We are interested in the following criteria to evaluate the effectiveness of the proposed algorithms: the error performance and its standard deviation (denoted by T_A and std_A

respectively), the number of iterations (denoted by Iter), the CPU time (in seconds) and the value of OBR (for ℓ_p-norm problems with $p = 1$ and $p = 2$). T_A and std_A are defined in [98] as follows.

$$T_A = \frac{1}{N_G.N_{data}} \sum_{p=1}^{N_G} \sum_{q=1}^{N_{data}} T_A^{p,q}, \quad T_A^{p,q} = \frac{\mathbb{E}_\rho[V^{\pi^*} - V^{\pi_A^{p,q}}]}{\mathbb{E}_\rho[|V^{\pi^*}|]},$$

where ρ is the uniform distribution over $\mathcal{S}$ and the policy $\pi_A^{p,q}$ is greedy with respect to the output Q_θ of the algorithm A and V^π is the state value function under policy π.

$$std_A = \frac{1}{N_G} \sum_{p=1}^{N_G} \left[\frac{1}{N_{data}} \sum_{q=1}^{N_{data}} \left(T_A^{p,q} - \frac{1}{N_{data}} \sum_{k=1}^{N_{data}} T_A^{p,k} \right)^2 \right]^{\frac{1}{2}}.$$

The value of OBR for ℓ_p-norm problems ($p = 1$ and $p = 2$) is given by

$$\text{OBR} = \left(\frac{1}{N} \sum_{i=1}^{N} |f_i(\theta)|^p \right)^{1/p}.$$

5.4.3 Experiment 1: Comparison between DCA based algorithms

5.4.3.1 Comparative results of the six versions of DCA

Two purposes of our experiments in this part are to, first, give a comparison of six versions of DCA and, second, compare carefully between ℓ_1-DCA (resp. ℓ_2-DCA) and DCA_1 (resp. DCA_2).

For the first purpose, we test our four DCA schemes (ℓ_1-DCA, ℓ_2-DCA, cc-DCA, ℓ_∞-DCA) and two existing ones (DCA_1, DCA_2) on 10 Garnet(100,5,5) problems $\{G_p\}_{1\leq p\leq 10}$ and 5 sample datasets $\{D^{p,q}\}_{1\leq q\leq 5}$ for each Garnet G_p as described in Section 5.4.1. The starting point of DCA_1, DCA_2 is set to zero as in [98]. The average results of T_A, std_A, Iter and CPU obtained by the algorithms on 50 different runs corresponding to 50 pairs of (G_p, $D^{p,q}$) with different values of N ($N \in \{500, 800, 1000, 1300, 1500, 1800, 2000, 2300, 2500\}$) are shown in Table 5.2. In Fig. 5.1 we plot the same results concerning T_A and CPU time. We also test the algorithms on 5 different Garnet($N_\mathcal{S}, 5, 5$) problems ($N_\mathcal{S} \in \{225, 324\}$) and 5 datasets of N transition samples for each Garnet with different values of N ($N \in \{4000, 5000, 6000, 7000\}$). The comparative results are given in Table 5.3.

For the second purpose, we perform four algorithms ℓ_1-DCA, ℓ_2-DCA and DCA_1, DCA_2 on 10 different Garnet(400,5,5) problems and 2 transition sample datasets for each Garnet with the same tolerance $\varepsilon = 10^{-6}$ and the same starting point as proposed in Section 5.3.6. The average results of T_A, std_A, Iter, CPU and OBR obtained by the algorithms with different numbers of transition samples N ($N \in \{5000, 6000, 7000, 8000\}$) are reported in Table 5.4.

Table 5.2 – Average results of T_A, std_A, Iter and CPU in seconds obtained by the six versions of DCA on 50 runs (corresponding to 10 different Garnet($N_{\mathcal{S}} = 100$,$N_{\mathcal{A}} = 5$,$N_B = 5$) problems and 5 transition sample datasets for each Garnet) with different numbers of transition samples N. Bold values indicate the best results.

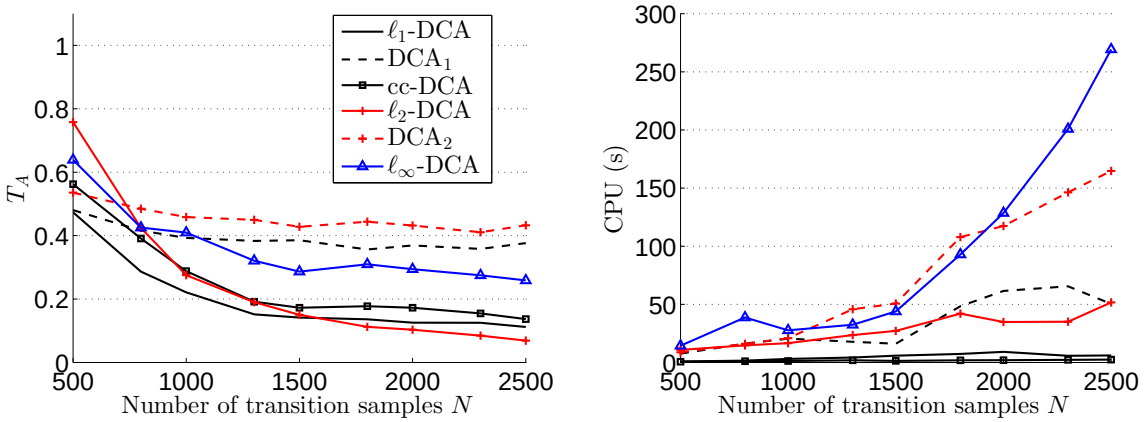
N		ℓ_1-DCA	ℓ_2-DCA	ℓ_∞-DCA	cc-DCA	DCA_1	DCA_2
500	T_A	**0.474**	0.758	0.639	0.562	0.481	0.536
	std_A	0.06	0.09	0.10	0.07	0.06	0.06
	CPU	0.84	10.8	14.4	**0.65**	7.58	8.72
	Iter	7.10	17.3	4.78	6.98	6.30	5.86
800	T_A	**0.286**	0.425	0.425	0.391	0.416	0.485
	std_A	0.05	0.09	0.08	0.07	0.05	0.07
	CPU	1.57	14.7	38.7	**1.01**	16.2	16.1
	Iter	5.78	14.6	3.66	5.72	7.40	5.92
1000	T_A	**0.221**	0.275	0.409	0.288	0.393	0.458
	std_A	0.05	0.07	0.07	0.07	0.05	0.08
	CPU	3.06	16.6	27.7	**1.27**	20.4	20.9
	Iter	5.26	12.5	3.98	4.64	7.60	5.68
1300	T_A	**0.152**	0.190	0.321	0.192	0.383	0.450
	std_A	0.03	0.05	0.05	0.04	0.06	0.08
	CPU	4.21	23.6	32.4	**1.80**	17.8	45.8
	Iter	4.68	10.9	3.56	4.08	7.64	5.84
1500	T_A	**0.141**	0.150	0.287	0.173	0.386	0.428
	std_A	0.03	0.04	0.06	0.04	0.06	0.07
	CPU	5.78	27.2	43.9	**1.36**	16.1	50.6
	Iter	4.18	10.5	3.50	4.10	7.22	6.24
1800	T_A	0.136	**0.112**	0.310	0.177	0.356	0.444
	std_A	0.03	0.03	0.09	0.04	0.04	0.09
	CPU	7.33	41.9	92.9	**1.79**	48.3	108
	Iter	5.08	9.90	3.82	3.80	7.58	5.82
2000	T_A	0.126	**0.103**	0.294	0.173	0.369	0.432
	std_A	0.03	0.02	0.06	0.04	0.06	0.08
	CPU	9.11	34.9	129	**1.90**	61.5	117
	Iter	5.34	9.82	3.26	3.48	8.18	6.06
2300	T_A	0.125	**0.084**	0.275	0.155	0.358	0.411
	std_A	0.03	0.02	0.07	0.03	0.06	0.07
	CPU	5.69	35.0	201	**2.25**	65.5	146
	Iter	5.06	9.50	3.58	3.26	8.00	5.92
2500	T_A	0.112	**0.069**	0.259	0.137	0.376	0.433
	std_A	0.03	0.02	0.07	0.04	0.06	0.08
	CPU	6.07	51.7	269	**2.44**	50.4	165
	Iter	5.12	9.36	4.08	2.86	7.48	6.38

Table 5.3 – Average results of T_A, std_A, Iter and CPU in seconds obtained by the six versions of DCA on 25 runs (corresponding to 5 different Garnet($N_{\mathcal{S}}$,5,5) problems ($N_{\mathcal{S}} \in \{225, 324\}$) and 5 transition sample datasets for each Garnet) with different numbers of transition samples N. Bold values indicate the best results.

Garnet $(N_{\mathcal{S}}, N_{\mathcal{A}}, N_B)$	N		ℓ_1-DCA	ℓ_2-DCA	ℓ_∞-DCA	cc-DCA	DCA_1	DCA_2
(225,5,5)	4000	T_A	0.225	**0.162**	0.572	0.360	0.369	0.370
		std_A	0.03	0.03	0.09	0.04	0.03	0.05
		CPU	38.0	77.8	793	**9.57**	216	361
		Iter	9.84	9.36	3.64	4.76	9.88	9.24
	5000	T_A	0.208	**0.145**	0.640	0.328	0.354	0.359
		std_A	0.02	0.03	0.08	0.05	0.03	0.06
		CPU	20.1	126	1175	**8.49**	250	374
		Iter	10.4	9.00	2.84	4.48	10.0	8.44
	6000	T_A	0.188	**0.145**	0.632	0.307	0.360	0.328
		std_A	0.03	0.03	0.13	0.04	0.04	0.06
		CPU	22.4	199	1136	**9.40**	204	505
		Iter	8.84	9.28	2.00	3.76	9.84	8.92
	7000	T_A	0.197	**0.135**	0.614	0.308	0.352	0.336
		std_A	0.03	0.02	0.10	0.05	0.03	0.07
		CPU	27.5	143	1765	**11.3**	298	585
		Iter	8.68	9.16	2.00	3.64	9.40	8.72
(324,5,5)	4000	T_A	0.271	**0.181**	0.697	0.425	0.353	0.325
		std_A	0.03	0.02	0.12	0.04	0.04	0.04
		CPU	65.2	153	773	**10.0**	753	1307
		Iter	11.2	13.4	4.44	6.80	12.0	9.6
	5000	T_A	0.249	**0.148**	0.654	0.397	0.355	0.346
		std_A	0.04	0.03	0.07	0.05	0.03	0.06
		CPU	93.9	231	994	**10.9**	748	1259
		Iter	10.3	12.4	3.04	5.48	10.6	8.52
	6000	T_A	0.241	**0.121**	0.730	0.384	0.345	0.295
		std_A	0.03	0.02	0.08	0.04	0.04	0.04
		CPU	116	210	1150	**11.0**	789	1377
		Iter	9.48	11.3	2.00	4.44	10.2	9.60
	7000	T_A	0.234	**0.106**	0.712	0.368	0.358	0.300
		std_A	0.03	0.02	0.07	0.04	0.03	0.05
		CPU	133	300	1830	**12.3**	842	1428
		Iter	9.96	11.0	2.00	4.08	9.44	10.0

Table 5.4 – Average results of T_A, std_A, Iter, CPU in seconds and OBR obtained by ℓ_1-DCA and DCA_1 (resp. ℓ_2-DCA and DCA_2) on 20 runs (corresponding to 10 Garnet(400,5,5) problems and 2 transition sample datasets for each Garnet) with different numbers of transition samples N. Bold values indicate the best results in the same ℓ_1-norm (resp. ℓ_2-norm) problem.

N		ℓ_1-DCA	DCA_1	ℓ_2-DCA	DCA_2
5000	T_A	**0.294**	0.459	**0.217**	0.404
	std_A	0.02	0.02	0.02	0.03
	CPU	**100**	400	**155**	1813
	Iter	13.8	12.6	15.6	11.6
	OBR	**0.070**	0.118	**0.211**	0.318
6000	T_A	**0.294**	0.480	**0.175**	0.398
	std_A	0.02	0.02	0.01	0.02
	CPU	**172**	395	**261**	1902
	Iter	14.9	10.8	16.4	11.1
	OBR	**0.073**	0.129	**0.205**	0.299
7000	T_A	**0.298**	0.480	**0.145**	0.400
	std_A	0.02	0.02	0.01	0.02
	CPU	**236**	378	**406**	1741
	Iter	12.6	8.60	15.6	13.7
	OBR	**0.073**	0.132	**0.215**	0.292
8000	T_A	**0.280**	0.488	**0.139**	0.395
	std_A	0.02	0.03	0.01	0.03
	CPU	**319**	431	**541**	1901
	Iter	12.6	8.75	15.0	13.1
	OBR	**0.074**	0.138	**0.220**	0.304



(a) Error performance T_A (b) CPU time

Figure 5.1 – Average results of T_A and CPU in seconds obtained by the six versions of DCA on 50 runs (corresponding to 10 different Garnet($N_\mathcal{S} = 100, N_\mathcal{A} = 5, N_B = 5$) problems and 5 transition sample datasets for each Garnet) with different numbers of transition samples N.

Comments on computational results:

a) Comparison between the six versions of DCA

i) In terms of the quality of solutions:

• For the medium Garnet problems ($N_\mathcal{S} = 100$) in Table 5.2, ℓ_1-DCA and ℓ_2-DCA are the two best algorithms, cc-DCA is the third, ℓ_∞-DCA is the next and DCA$_1$, DCA$_2$ are the worst. In particular, ℓ_1-DCA is the best in 5/9 cases (when $N \leq 1500$) while ℓ_2-DCA is the best in other four cases. The ratio of gain on T_A of ℓ_1-DCA versus cc-DCA, ℓ_∞-DCA, DCA$_1$, DCA$_2$ and ℓ_2-DCA varies from 15.6% to 27.1%, from 25.8% to 57.1%, from 1.45% to 70.2%, from 11.5% to 74.1% and from 6% to 37.4%, respectively. As for ℓ_2-DCA, it outperforms cc-DCA, ℓ_∞-DCA, DCA$_1$ in 7/9 cases (except when $N = 500$ and $N = 800$) and DCA$_2$ in 8/9 cases (except when $N = 500$) – the gain versus cc-DCA, ℓ_∞-DCA, DCA$_1$, DCA$_2$ varies from 1.04% to 49.6%, from 32.7% to 73.3%, from 30% to 81.6%, from 12.3% to 84%, respectively. cc-DCA is more efficient than DCA$_1$, DCA$_2$ in 8/9 cases (except when $N = 500$) and ℓ_∞-DCA in all cases – the gain versus DCA$_1$, DCA$_2$, ℓ_∞-DCA varies from 6.01% to 63.5%, from 19.3% to 68.3%, from 8% to 47.1%, respectively. Finally, ℓ_∞-DCA also outperforms DCA$_1$ (resp. DCA$_2$) in 6/9 cases (resp. 8/9 cases) – the gain varies from 13.9% to 31.1% (resp. from 10.6% to 40.1%).

• For the larger Garnet problems ($N_\mathcal{S} \in \{225, 324\}$ and $N \geq 4000$) in Table 5.3, the quality of the algorithms is classified in the following order: ℓ_2-DCA, ℓ_1-DCA, cc-DCA, DCA$_1$, DCA$_2$, ℓ_∞-DCA. Particularly, ℓ_2-DCA gives the best T_A in all cases – the ratio of gain versus ℓ_1-DCA, cc-DCA, ℓ_∞-DCA, DCA$_1$, DCA$_2$ varies from 22.8% to 54.7%, from 52.7% to 71.1%, from 71.6% to 85.1%, from 48.7% to 70.3%, from 44.3% to 64.6%, respectively. As for ℓ_1-DCA, the gain

on T_A versus cc-DCA, ℓ_∞-DCA, DCA_1, DCA_2 varies from 36% to 38.7%, from 60.6% to 70.2%, from 23.2% to 47.7%, from 16.6% to 42.6%, respectively. The T_A of cc-DCA is better than that of both DCA_1 and DCA_2 in 4/8 cases with the ratio of gain from 2.43% to 14.7% and larger in other four cases with the ratio of loss from 2.71% to 23.5%. ℓ_∞-DCA seems to be inefficient in these large problems.

ii) Concerning CPU time:

cc-DCA is the fastest algorithm, ℓ_1-DCA and ℓ_2-DCA come next, and then DCA_1, DCA_2 and finally ℓ_∞-DCA. For the medium Garnet problems with $N_{\mathcal{S}} = 100$ (see Table 5.2), the ratio of gain of cc-DCA versus ℓ_1-DCA, ℓ_2-DCA, ℓ_∞-DCA varies, respectively, from 1.29 to 4.79, from 13.1 to 23.4, from 18 to 110.2 times. The gain is more important versus DCA_1 (resp. DCA_2): it varies from 9.88 to 32.3 (resp. from 13.4 to 67.6) times. ℓ_1-DCA is also fast – the gain versus DCA_1 varies from 2.78 to 11.5 times. Not surprisingly, DCA_2, ℓ_2-DCA and ℓ_∞-DCA consume much more time due to the complexity of convex subproblems, but ℓ_2-DCA runs faster than DCA_2 and ℓ_∞-DCA in most cases – the gain of ℓ_2-DCA versus DCA_2 varies from 1.09 to 4.17 times in 8/9 cases (when $N \geq 800$). For the larger Garnet problems with $N_{\mathcal{S}} \in \{225, 324\}$ (see Table 5.3), the ratio of gain of cc-DCA is larger – the gain versus ℓ_1-DCA, ℓ_2-DCA, ℓ_∞-DCA, DCA_1, DCA_2 varies, respectively, from 2.36 to 10.8, from 8.13 to 24.3, from 82.8 to 156.1, from 21.7 to 75.3, from 37.7 to 131 times. As for ℓ_1-DCA, the gain versus ℓ_2-DCA, ℓ_∞-DCA, DCA_1, DCA_2 varies, respectively, from 1.81 to 8.88, from 9.91 to 64.1, from 5.68 to 12.4, from 9.5 to 22.5 times. ℓ_2-DCA runs faster than ℓ_∞-DCA, DCA_1, DCA_2 with the ratio of gain from 4.3 to 12.3, from 1.02 to 4.92, from 2.53 to 8.54 times respectively. Finally, DCA_1, ℓ_∞-DCA, DCA_2 run slowly and sometimes exceed the maximum CPU time.

b) ℓ_1-DCA versus DCA_1 (Table 5.4)

i) Regarding the quality of solutions: ℓ_1-DCA is much more efficient than DCA_1 for all Garnet problems in terms of both T_A and std_A. The ratio of gain on T_A varies from 35.9% to 42.6%. Moreover, the OBR of ℓ_1-DCA is always smaller than that of DCA_1.

ii) As for CPU time: ℓ_1-DCA runs faster than DCA_1 in all considered Garnet problems – the ratio of gain varies from 1.35 to 4 times.

c) ℓ_2-DCA versus DCA_2 (Table 5.4)

i) ℓ_2-DCA enhances T_A and OBR more significantly than DCA_2 in all cases. Particularly, the ratio of gain on T_A of ℓ_2-DCA versus DCA_2 varies from 46.2% to 64.8%.

ii) The CPU time of DCA_2 usually exceeds the maximum time (2000 seconds) while that of ℓ_2-DCA is about 155 seconds (when $N = 5000$) – the gain varies from 3.51 to 11.7 times.

In summary, we can say that for these medium Garnet problems, ℓ_1-DCA could be the algorithm realizing the best trade-off between the quality and the rapidity while ℓ_2-DCA could be the best choice when considering the larger (in terms of the state space

and sample datasets) Garnet problems. Moreover, our DCA algorithm ℓ_1-DCA (resp. ℓ_2-DCA) is more efficient than the existing one DCA_1 (resp. DCA_2) in all criteria.

5.4.3.2 Effect of starting points on our four DCA algorithms

In this experiment, we study the effect of our starting point $\theta_{(1)}$ (as in Section 5.3.6) and the zero starting point $\theta_{(2)}$ on our algorithms ℓ_1-DCA, ℓ_2-DCA, ℓ_∞-DCA and cc-DCA in terms of the error performance T_A on 10 different Garnet(100,5,5) problems and 5 transition sample datasets for each Garnet ($N \in \{200, 400, 600, 800, 1000\}$). The comparative results are shown in Table 5.5.

Table 5.5 – Comparative results of the proposed starting point $\theta_{(1)}$ and the zero starting point $\theta_{(2)}$ for our algorithms ℓ_1-DCA, ℓ_2-DCA, ℓ_∞-DCA and cc-DCA in terms of T_A on 50 runs (corresponding to 10 different Garnet(100,5,5) problems and 5 transition sample datasets for each Garnet) with different numbers of transition samples N. Bold values indicate the best results in each algorithm.

N	ℓ_1-DCA		ℓ_2-DCA		ℓ_∞-DCA		cc-DCA	
	$\theta_{(1)}$	$\theta_{(2)}$	$\theta_{(1)}$	$\theta_{(2)}$	$\theta_{(1)}$	$\theta_{(2)}$	$\theta_{(1)}$	$\theta_{(2)}$
200	**0.847**	0.862	**0.972**	0.990	**0.864**	0.877	**0.850**	0.859
400	**0.576**	0.627	**0.837**	0.860	**0.747**	0.816	**0.648**	0.734
600	**0.408**	0.537	**0.607**	0.640	**0.587**	0.741	**0.508**	0.783
800	**0.286**	0.513	**0.443**	0.459	**0.425**	0.694	**0.391**	0.812
1000	**0.221**	0.487	**0.269**	0.300	**0.409**	0.740	**0.288**	0.832

From Table 5.5 we see that, for all the algorithms, the starting point $\theta_{(1)}$ is more efficient than $\theta_{(2)}$ in all cases – the ratio of gain on T_A of ℓ_1-DCA, ℓ_2-DCA, ℓ_∞-DCA, cc-DCA varies from 1.74% to 54.6%, from 1.81% to 10.3%, from 1.48% to 44.7%, from 1.04% to 65.3%, respectively. This is an illustration of the effectiveness of our proposed starting point.

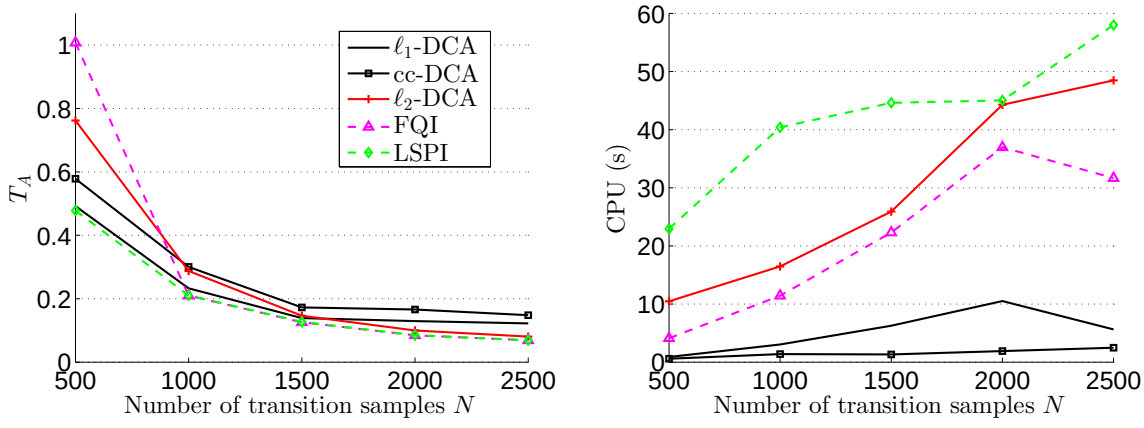
5.4.4 Experiment 2: Comparison with LSPI, FQI

5.4.4.1 Garnet problems

This experiment aims to make a comparison between our notable DCA algorithms ℓ_1-DCA, cc-DCA, ℓ_2-DCA and the standard RL algorithms FQI, LSPI on Garnet problems. We test the algorithms on 50 different Garnet(100,5,5) problems and 10 datasets of N transition samples for each Garnet ($N \in \{500, 1000, 1500, 2000, 2500\}$). The comparative results of these algorithms in terms of T_A, std_A and CPU are presented in Table 5.6. In Fig. 5.2 we plot the same results concerning T_A and CPU time.

Table 5.6 – Average results of T_A, std_A and CPU in seconds obtained by ℓ_1-DCA, ℓ_2-DCA, cc-DCA and LSPI, FQI on 500 runs (corresponding to 50 different Garnet(100,5,5) problems and 10 transition sample datasets for each Garnet) with different numbers of transition samples N. Bold values indicate the best results.

N		ℓ_1-DCA	ℓ_2-DCA	cc-DCA	FQI	LSPI
500	T_A	0.492	0.762	0.578	1.007	**0.478**
	std_A	0.07	0.09	0.09	0.00	0.08
	CPU	0.89	10.5	**0.58**	4.10	22.9
1000	T_A	0.233	0.288	0.301	**0.210**	0.211
	std_A	0.05	0.07	0.06	0.05	0.05
	CPU	3.04	16.5	**1.38**	11.4	40.3
1500	T_A	0.139	0.146	0.173	**0.126**	0.127
	std_A	0.03	0.04	0.05	0.04	0.04
	CPU	6.26	25.9	**1.32**	22.3	44.6
2000	T_A	0.130	0.100	0.166	**0.085**	**0.085**
	std_A	0.03	0.03	0.04	0.02	0.03
	CPU	10.5	44.3	**1.89**	37.0	45.0
2500	T_A	0.122	0.081	0.148	**0.069**	**0.069**
	std_A	0.03	0.02	0.04	0.02	0.02
	CPU	5.63	48.5	**2.48**	31.7	58.0



(a) Error performance T_A (b) CPU time

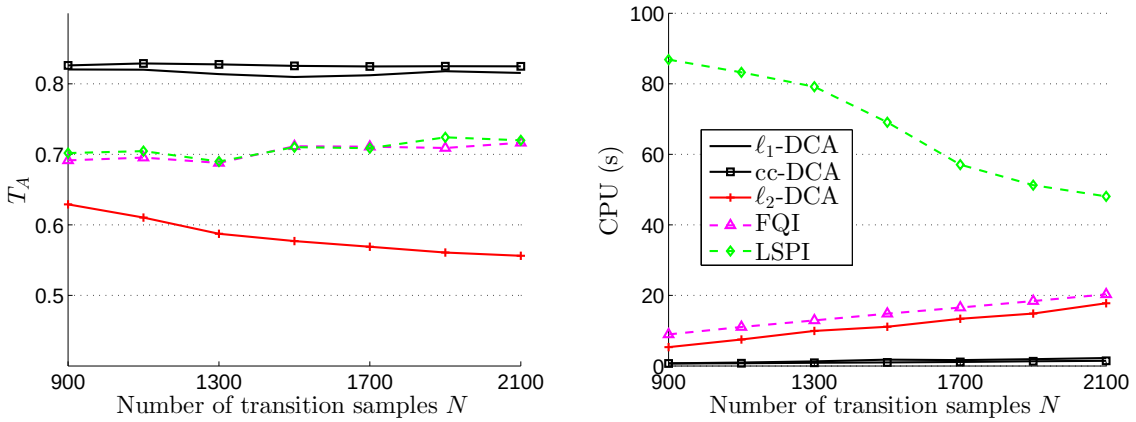
Figure 5.2 – Average results of T_A and CPU in seconds obtained by ℓ_1-DCA, ℓ_2-DCA, cc-DCA and LSPI, FQI on 500 runs (corresponding to 50 different Garnet(100,5,5) problems and 10 transition sample datasets for each Garnet) with different numbers of transition samples N.

Comments on numerical results

For medium Garnet problems (Table 5.6 and Fig. 5.2), LSPI and FQI give the best error performance T_A, except for the case $N = 500$ where FQI completely fails to find a good solution and its T_A is the largest versus all algorithms. Meanwhile, to reach these results LSPI and FQI need much more CPU time than our DCAs. The T_A furnished by ℓ_1-DCA is close to the ones of LSPI but ℓ_1-DCA is much faster than LSPI. The ratio of gain varies from 4.3 to 25.7 times. Similarly, ℓ_1-DCA is faster than FQI, from 3.5 to 5.6 times. As for ℓ_2-DCA and cc-DCA, their error performances T_A are comparable with ℓ_1-DCA. More precisely, ℓ_2-DCA gives the smaller T_A when $N = 2000$ and $N = 2500$ and the slightly larger T_A in other three cases, but it consumes more time than ℓ_1-DCA (from 4.1 to 11.8 times). The T_A given by cc-DCA is slightly larger than that of ℓ_1-DCA but it is faster from 1.5 to 5.5 times. Thus, not surprisingly, cc-DCA is the fastest algorithm, in particular, the ratio of gain versus LSPI (resp. FQI) varies from 23.4 to 39.4 (resp. from 7 to 19.6) times.
In summary, ℓ_1-DCA could be the algorithm realizing the best trade-off between the quality and the rapidity.

5.4.4.2 Gridworld problems

In this experiment, we study the performance of ℓ_1-DCA, ℓ_2-DCA, cc-DCA in comparing with LSPI, FQI on 10 different Gridworld problems and 10 random datasets of N transition samples for each Gridworld ($N \in \{900, 1100, \ldots, 2100\}$) as described in Section 5.4.1. The comparative results in terms of T_A, std_A and CPU time are reported in Table 5.7 and Fig. 5.3.



(a) Error performance T_A (b) CPU time

Figure 5.3 – Average results of T_A and CPU in seconds obtained by ℓ_1-DCA, ℓ_2-DCA, cc-DCA and LSPI, FQI on 100 runs (corresponding to 10 different Gridworld problems and 10 transition sample datasets for each Gridworld) with different numbers of transition samples N.

Table 5.7 – Average results of T_A, std_A and CPU in seconds obtained by ℓ_1-DCA, ℓ_2-DCA, cc-DCA and LSPI, FQI on 100 runs (corresponding to 10 different Gridworld problems and 10 transition sample datasets for each Gridworld) with different numbers of transition samples N. Bold values indicate the best results.

N		ℓ_1-DCA	cc-DCA	ℓ_2-DCA	FQI	LSPI
900	T_A	0.820	0.826	**0.629**	0.691	0.702
	std_A	0.02	0.02	0.06	0.06	0.07
	CPU	0.75	**0.69**	5.34	8.94	86.9
1100	T_A	0.820	0.829	**0.610**	0.695	0.705
	std_A	0.02	0.01	0.06	0.07	0.06
	CPU	0.98	**0.78**	7.50	11.0	83.3
1300	T_A	0.814	0.827	**0.587**	0.688	0.690
	std_A	0.02	0.01	0.05	0.06	0.06
	CPU	1.31	**0.90**	9.94	12.9	79.2
1500	T_A	0.810	0.826	**0.577**	0.712	0.710
	std_A	0.03	0.00	0.04	0.06	0.06
	CPU	1.77	**1.01**	11.1	14.8	69.1
1700	T_A	0.812	0.825	**0.569**	0.711	0.708
	std_A	0.02	0.00	0.04	0.06	0.07
	CPU	1.65	**1.12**	13.4	16.6	57.1
1900	T_A	0.818	0.825	**0.561**	0.710	0.724
	std_A	0.02	0.00	0.03	0.06	0.06
	CPU	1.93	**1.33**	14.9	18.4	51.3
2100	T_A	0.815	0.825	**0.556**	0.716	0.720
	std_A	0.02	0.00	0.02	0.06	0.07
	CPU	2.24	**1.44**	17.8	20.3	48.1

Comments on numerical results (Table 5.7 and Fig. 5.3)

In terms of quality, ℓ_2-DCA gives the best error performance T_A in all cases, LSPI and FQI come next, and afterward ℓ_1-DCA, then cc-DCA. In particular, the gain on T_A of ℓ_2-DCA versus LSPI (resp. FQI) varies from 10.3% to 22.7% (resp. from 8.97% to 22.3%). As for the rapidity, cc-DCA is always the fastest algorithm, especially the ratio of gain versus LSPI (resp. FQI) varies from 33.4 to 125.9 (resp. from 12.96 to 14.8) times. In all cases, LSPI consumes much more time than other algorithms. Among three DCA based algorithms ℓ_2-DCA is the slowest, meanwhile it is faster than LSPI (resp. FQI) from 2.7 to 16.27 (resp. from 1.14 to 1.67) times. In sum, we can say that for these Gridworld problems ℓ_2-DCA is the most efficient algorithm which realizes the best trade-off between the quality and the rapidity.

5.5 Conclusions

We have intensively investigated DC programming and DCA for Batch RL by finding the zero of the empirical Optimal Bellman Residual via linear approximation. Besides the existing ℓ_p-norm formulations of OBR ($p \in \{1, 2\}$) we have considered the ℓ_∞-norm, and also proposed the new concave minimization formulation without using the ℓ_p-norm of OBR. These optimization formulations are reformulated as DC programs for which four DCA schemes have been developed. Exploiting the special structure of the considered problems we have carefully addressed the two key issues of DCA – the nice effect of DC decompositions and the solution methods of resulting convex subproblems so as to design efficient DCA based algorithms. It turns out that, for the same ℓ_1-norm (resp. ℓ_2-norm) formulation, the proposed ℓ_1-DCA (resp. ℓ_2-DCA) is much more efficient than the existing DCA_1 (resp. DCA_2) on both quality and rapidity. Among the four proposed DCA, the cc-DCA (corresponding to the new concave formulation) is the fastest, and the three ℓ_1-DCA, cc-DCA and ℓ_∞-DCA enjoy interesting convergence properties of polyhedral DC programs while ℓ_2-DCA is of good quality thanks to the update of DC decompositions during the algorithm. Comparing with the two standard approaches for Batch RL, ℓ_1-DCA (resp. ℓ_2-DCA) realizes the best trade-off between the quality and the rapidity for Garnet problems (resp. Gridworld problems).

Chapter 6

Online DCA for Reinforcement Learning[1]

Abstract: *This chapter focuses on developing reinforcement learning (RL) techniques in online mode via the optimal Bellman residual (OBR) minimization approach. Based upon the transition sample collected from the interaction at each step, an agent tries to improve the policy via the state-action value function Q by optimizing the sampled OBR functions with ℓ_p-norm ($p \geq 1$). In fact, each ℓ_p-norm optimization formulation of OBR is a DC (Difference of Convex functions) program. In this chapter, we investigate Online DCA (DC Algorithm) for solving the ℓ_2-norm formulations of OBR via value-function linear approximation. Exploiting an appropriate DC decomposition, we develop a corresponding Online DCA based algorithm which enjoys the online stability property. We also indicate that the well-known residual gradient algorithm for RL is a special case of our Online DCA algorithm. We propose an alternating version of this algorithm where the value function at each step is alternatively updated. Numerical experiments on two benchmarks – mountain car and pole balancing problems – show the effectiveness of our approaches in comparison with some standard RL algorithms (Q-learning, SARSA).*

1. The material of this chapter is based on the following work:
[1]. Vinh Thanh Ho, Hoai An Le Thi. Online DCA for Reinforcement Learning. *Submitted* to ALT 2018: 29th International Conference on Algorithmic Learning Theory.

6.1 Our contributions

We develop RL techniques in online mode based on Online DC programming and Online DCA by addressing the ℓ_p-norm of empirical OBR via value-function linear approximation. Firstly, we consider an ℓ_2-norm optimization formulation of OBR for which Online DC programming is investigated. Exploiting an appropriate DC decomposition for DC function at each step, we develop the corresponding Online DCA based algorithm, which enjoys the online stability property. Secondly, we suggest an alternating version of our algorithm where the update rule of value function is computed alternatively. Finally, we provide several numerical experiments of the proposed algorithms on two benchmarks – mountain car problems and pole balancing problems – in comparison with two well-known RL algorithms, namely Q-learning and SARSA.

The rest of the chapter is organized as follows. Optimization formulations of the empirical OBR is described in Section 6.2. Section 6.3 first presents an introduction of Online DC programming and Online DCA and then shows how to investigate Online DC programming and Online DCA for RL. Section 6.4 reports the numerical results on several test problems which is followed by some conclusions in Section 6.5.

6.2 Optimization formulations

Similarly to Section 5.2, we briefly restate the optimization formulation of ℓ_p-norm ($p \geq 1$) of OBR, in particular when $p = 2$. Given a basic function vector $\phi(\cdot) = (\phi_1(\cdot), \phi_2(\cdot), ..., \phi_d(\cdot))^\top \in \mathbb{R}^d$ where $\phi_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, i = 1, \ldots, d$ are basic functions on $\mathcal{S} \times \mathcal{A}$. Then the linear approximation of the function Q, denoted by Q_θ, is characterized by the weight vector $\theta \in \mathbb{R}^d$ according to the relation

$$Q_\theta(s, a) := \langle \theta, \phi(s, a) \rangle, \forall (s, a) \in \mathcal{S} \times \mathcal{A}.$$

Let $\mathcal{F}$ denote the space of approximation functions, say

$$\mathcal{F} := \left\{ Q_\theta(s, a) = \langle \theta, \phi(s, a) \rangle | \theta \in \mathbb{R}^d \right\}.$$

The ℓ_p-norm ($p \geq 1$) of OBR with the probability distribution μ is defined as (see e.g. [86])

$$J_{p,\mu}(Q) := ||B^*Q - Q||_{p,\mu}, \text{ where } ||Q||_{p,\mu} := \left(\sum_{(s,a) \in \mathcal{S} \times \mathcal{A}} \mu(s, a) |Q(s, a)|^p \right)^{\frac{1}{p}}.$$

Thus, with the linear approximation, the ℓ_p-norm ($p \geq 1$) of the empirical OBR over the space of linear approximation functions $\mathcal{F}$ is

$$J_{p,\mu}(Q_\theta) = ||B^*Q_\theta - Q_\theta||_{p,\mu}.$$

In RL, estimating the optimal state-action value function at each step is based on the transition sample through the interaction with the environment. Specifically, at the step t, one first collects a sample (S_t, A_t, S'_t, r_t) via ε-greedy strategy, then considers a nonbiased estimation of $B^*Q(S_t, A_t)$:

$$\widehat{B}^*Q(S_t, A_t) = r_t + \gamma \max_{a' \in \mathcal{A}} Q(S'_t, a')$$

Finally, at the final step T, one minimize the corresponding empirical ℓ_p-norm of OBR: $J_{p,T}(Q) := \left[\sum_{t=1}^{T} \left(\widehat{B}^*Q(S_t, A_t) - Q(S_t, A_t)\right)^p\right]^{1/p}$.

Using this sampling-based technique with linear approximation operator, we have to minimize the empirical ℓ_p-norm ($p \geq 1$) of the empirical OBR that is

$$\left(\sum_{t=1}^{T} \left|\widehat{B}^*Q_\theta(S_t, A_t) - Q_\theta(S_t, A_t)\right|^p\right)^{\frac{1}{p}}$$

and finally the ℓ_p-norm formulation of the empirical OBR via linear approximation takes the form

$$\min_{\theta \in \mathbb{R}^d} \left\{ F_{p,T}(\theta) := \left(\sum_{t=1}^{T} \left| r_t + \gamma \max_{a' \in \mathcal{A}} \langle \theta, \phi(S'_t, a')\rangle - \langle \theta, \phi(S_t, A_t)\rangle \right|^p\right) \right\}. \tag{6.1}$$

Let p_t denote the real-value function on $\mathbb{R}^d$ defined as

$$p_t(\theta) := r_t + \gamma \max_{a' \in \mathcal{A}} \langle \theta, \phi(S'_t, a')\rangle - \langle \theta, \phi(S_t, A_t)\rangle.$$

Clearly, p_t is the maximal function of a finite family of affine functions, expressed as

$$p_t(\theta) = \max_{j=1,\ldots,N_{\mathcal{A}}} \left\{\langle \mathbf{A}^{(t,j)}, \theta\rangle + \mathbf{b}^{(t)}\right\}$$

where $\mathbf{A}^{(t,j)} := \gamma\phi(S'_t, a_j) - \phi(S_t, A_t)$, $\mathbf{b}^{(t)} := r_t$, $j = 1, \ldots, N_{\mathcal{A}}$.

In this chapter, we consider the ℓ_2-norm optimization formulation at the final step T taking the form

$$\min \left\{ F_{2,T}(\theta) := \sum_{t=1}^{T} [p_t(\theta)]^2 : \theta \in \mathbb{R}^d \right\}. \tag{6.2}$$

As each p_t is a polyhedral convex function, $F_{2,T}$ is a DC function.

6.3 Solution methods by Online DC programming and Online DCA

6.3.1 Online DCA for solving the ℓ_2-norm problem (6.2)

Recall that the ℓ_2-norm problem (6.2) at final step T is of the form:

$$\min \left\{ F_{2,T}(\theta) := \sum_{t=1}^{T} [p_t(\theta)]^2 : \theta \in \mathbb{R}^d \right\}.$$

According to Online DC programming, at the step t, we have the DC function

$$f_t(\theta) = [p_t(\theta)]^2 .$$

Taking $\overline{a}_t \in \text{dom } \partial p_t$, $\overline{b}_t \in \partial p_t(\overline{a}_t)$, we define the affine minorization of the polyhedral convex function p_t as follows

$$l_t(\theta) = p_t(\overline{a}_t) + \langle \theta - \overline{a}_t, \overline{b}_t \rangle.$$

Let $l^-(\theta) := \max\{0, -l(\theta)\}$. For $\theta \in \mathbb{R}^d$, we have

$$p_t(\theta) = \left[p_t(\theta) + l_t^-(\theta)\right] - l_t^-(\theta),$$

which implies that

$$f_t(\theta) = g_t(\theta) - h_t(\theta),$$

where

$$g_t(\theta) = 2\left\{ \left[p_t(\theta) + l_t^-(\theta)\right]^2 + \left[l_t^-(\theta)\right]^2 \right\}$$

and

$$h_t(\theta) = \left[p_t(\theta) + 2l_t^-(\theta)\right]^2 .$$

It is evident that $p_t + l_t^-$ and l_t^- are nonnegative and convex on $\mathbb{R}^d$, so are g_t and h_t.

According to a particular Online DCA based scheme in Section 1.2.3, at the step t, we solve the subproblem (1.11) by approximating by one iteration of subgradient method. Thus, θ^{t+1} will be updated as

$$\theta^{t+1} = \theta^t - \frac{\eta_t}{2} v^t \tag{6.3}$$

where η_t is a step size, the vector $v^t \in (\partial g_t(\theta^t) - z^t)$, $z^t \in \partial h_t(\theta^t)$.

■ *Compute* $\partial h_t(\theta)$: from the definition of h_t and l_t^-, we have

$$\partial h_t(\theta) = \partial \left[p_t(\theta) + 2l_t^-(\theta)\right]^2 \supset 2\left[p_t(\theta) + 2l_t^-(\theta)\right]\left[\partial p_t(\theta) + 2\partial(l_t^-)(\theta)\right],$$

$$\partial l_t^-(\theta) = \begin{cases} \{0\} & \text{if } l_t(\theta) > 0, \\ \left[0, -\overline{b}_t\right] & \text{if } l_t(\theta) = 0, \\ \{-\overline{b}_t\} & \text{if } l_t(\theta) < 0, \end{cases}$$

and

$$\begin{aligned} \partial p_t(\theta) &= \partial \left[\max_{j=1,\ldots,N_{\mathcal{A}}} \langle \mathbf{A}^{(t,j)}, \theta \rangle + \mathbf{b}^{(t)} \right] \\ &= \text{co}\{\mathbf{A}^{(t,j_t)} : j_t \in I_t(\theta)\}, \\ I_t(\theta) &= \text{argmax}_{\ j=1,\ldots,N_{\mathcal{A}}} \langle \mathbf{A}^{(t,j)}, \theta \ \rangle. \end{aligned} \tag{6.4}$$

Hence, we can take a subgradient $z^t \in \partial h_t(\theta^t)$ as the following procedure.

+ Let j_t be an index in $I_t(\theta^t)$ defined by (6.4), then $b_t = \mathbf{A}^{(t,j_t)}$.
+ Let $\overline{j}_t$ be an index in $I_t(\overline{a}_t)$, then $\overline{b}_t = \mathbf{A}^{(t,\overline{j}_t)}$.

+ Let $\tau_{1,t}$ be a real number in $[-1, 0]$. We choose $\overline{c}_{1,t} \in \partial(l_t^-)(\theta^t)$ as

$$\overline{c}_{1,t} = \begin{cases} \{0\} & \text{if } l_t(\theta^t) > 0, \\ \{\tau_{1,t}\overline{b}_t\} & \text{if } l_t(\theta^t) = 0, \\ \{-\overline{b}_t\} & \text{if } l_t(\theta^t) < 0. \end{cases}$$

+ z^t is computed by

$$z^t = 2\left[p_t(\theta^t) + 2l_t^-(\theta^t)\right]\left[b_t + 2\overline{c}_{1,t}\right]. \tag{6.5}$$

■ *Compute* $\partial g_t(\theta)$: from the definition of g_t, we have

$$\begin{aligned} \partial g_t(\theta) &= 2\partial\left\{\left[p_t(\theta) + l_t^-(\theta)\right]^2 + \left[l_t^-(\theta)\right]^2\right\} \\ &\supset 4\left[p_t(\theta) + l_t^-(\theta)\right]\left[\partial p_t(\theta) + \partial(l_t^-)(\theta)\right] + 4l_t^-(\theta)\partial(l_t^-)(\theta). \end{aligned}$$

Similarly, we can take a subgradient $u^t \in \partial g_t(\theta^t)$ as the following procedure.

+ Let $\tau_{2,t}$ be a real number in $[-1, 0]$. We choose $\overline{c}_{2,t} \in \partial(l_t^-)(\theta^t)$ as

$$\overline{c}_{2,t} = \begin{cases} \{0\} & \text{if } l_t(\theta^t) > 0, \\ \{\tau_{2,t}\overline{b}_t\} & \text{if } l_t(\theta^t) = 0, \\ \{-\overline{b}_t\} & \text{if } l_t(\theta^t) < 0. \end{cases}$$

+ u^t is computed by

$$u^t = 4\left[p_t(\theta^t) + l_t^-(\theta^t)\right]\left[b_t + \overline{c}_{2,t}\right] + 4l_t^-(\theta^t)\overline{c}_{2,t}. \tag{6.6}$$

From (6.5) and (6.6), $v^t \in (\partial g_t(\theta^t) - z^t)$ is chosen as

$$\begin{aligned} v^t &= u^t - z^t \\ &= 4\left[p_t(\theta^t) + l_t^-(\theta^t)\right]\left[b_t + \overline{c}_{2,t}\right] + 4l_t^-(\theta^t)\overline{c}_{2,t} - 2\left[p_t(\theta^t) + 2l_t^-(\theta^t)\right]\left[b_t + 2\overline{c}_{1,t}\right] \\ &= \begin{cases} 2p_t(\theta^t)b_t & \text{if } l_t(\theta^t) \neq 0 \\ 2p_t(\theta^t)\left[b_t + 2(\tau_{2,t} - \tau_{1,t})\overline{b}_t\right] & \text{if } l_t(\theta^t) = 0 \end{cases} \qquad (6.7) \\ &= 2\left[\langle \mathbf{A}^{(t,j_t)}, \theta^t\rangle + \mathbf{b}^{(t)}\right]\left[\mathbf{A}^{(t,j_t)} + 2\delta_t\mathbf{A}^{(t,\overline{j}_t)}\mathbb{1}_{\langle \mathbf{A}^{(t,\overline{j}_t)},\theta\rangle + \mathbf{b}^{(t)}=0}(\theta^t)\right], \qquad (6.8) \end{aligned}$$

where $\delta_t = \tau_{2,t} - \tau_{1,t} \in [-1, 1]$.

At last, Online DCA applied to (6.2) can be described as Algorithm 6.1 (ODCA).

Algorithm 6.1 Online DCA for solving (6.2) (ODCA)

Initialization: let $\{\eta_t\}$ be the sequence of step sizes, θ^1 be the initial point
for $t = 1, 2, \ldots$ **do**
1. Choose $j_t \in I_t(\theta^t)$ and $\overline{j}_t \in I_t(\overline{a}_t)$ using (6.4).
2. Compute $v^t \in (\partial g_t(\theta^t) - z^t)$ with $z^t \in \partial h_t(\theta^t)$ using (6.8).
3. Compute θ^{t+1} using (6.3).

end for

The proposed ODCA algorithm enjoys an interesting property as stated Theorem 6.1. The proof technique of this theorem rests partly on the works [104].

Theorem 6.1. *(Online stability of* ODCA*)*

Assume that ODCA *generates the sequence* $\{\theta^t\}_{t=1,\ldots,T+1}$ *with the step size* $\eta_t = \eta$ *for all* t *and the sequence* $\{v^t\}$ *is upper bounded by* V*, i.e.* $\|v^t\| \leq V$ *for all* t*. If* $\eta = 1/\sqrt{T}$*, then the* ODCA *algorithm is online stable, that is to say,*

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \|\theta^t - \theta^{t+1}\| = 0.$$

Proof. We know that the update rule (6.3) is equivalent to

$$\theta^{t+1} \in \arg\min_{\theta\in\mathbb{R}^d} \sum_{i=1}^{t} \langle \theta, v^i \rangle + \frac{1}{\eta}\|\theta\|^2.$$

Since the function $\|\cdot\|^2$ is a strongly convex function on $\mathbb{R}^d$, we derive from (6.3.1) that

$$\left(\sum_{i=1}^{t} \langle \theta^t, v^i \rangle + \frac{1}{\eta}\|\theta^t\|^2\right) - \left(\sum_{i=1}^{t} \langle \theta^{t+1}, v^i \rangle + \frac{1}{\eta}\|\theta^{t+1}\|^2\right) \geq \frac{1}{\eta}\|\theta^t - \theta^{t+1}\|^2. \tag{6.9}$$

Similarly to the step $t-1$, we have

$$\left(\sum_{i=1}^{t-1} \langle \theta^{t+1}, v^i \rangle + \frac{1}{\eta}\|\theta^{t+1}\|^2\right) - \left(\sum_{i=1}^{t-1} \langle \theta^t, v^i \rangle + \frac{1}{\eta}\|\theta^t\|^2\right) \geq \frac{1}{\eta}\|\theta^{t+1} - \theta^t\|^2. \tag{6.10}$$

The sum, side by side, of (6.9) and (6.10) yields

$$\frac{2}{\eta}\|\theta^{t+1} - \theta^t\|^2 \leq \langle \theta^t - \theta^{t+1}, v^t \rangle \leq \|\theta^{t+1} - \theta^t\|\|v^t\|, \ \forall t.$$

Thus, we have

$$\mathsf{S}^T_{\mathrm{ODCA}} = \frac{1}{T} \sum_{t=1}^{T} \|\theta^t - \theta^{t+1}\| \leq \frac{\eta V}{2},$$

which concludes the proof. □

Remark 6.1. *By choosing the suitable parameters, we see that the class of residual gradient* (RG) *algorithms in [5] is a special case of* ODCA*. Indeed, if* $\delta_t = 0$ *or* $\overline{a}_t = \theta^t$ *for all* t*, then*

$$\theta^{t+1} = \theta^t - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)}\right] \mathbf{A}^{(t,j_t)},$$

which is exactly the same as the update step in RG *algorithms, which is summarized in* Algorithm 6.2 (ODCA1).

Algorithm 6.2 ODCA when $\overline{a}_t = \theta^t$ for all t (ODCA1)

Initialization: let $\{\eta_t\}$ be the sequence of step sizes, θ^1 be the initial point
for $t = 1, 2, \ldots, T$ **do**
1. Choose $j_t \in I_t(\theta^t)$ using (6.4).
2. Compute $v^t \in (\partial g_t(\theta^t) - z^t)$ with $z^t \in \partial h_t(\theta^t)$:

$$v^t = 2\left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)}\right] \mathbf{A}^{(t,j_t)}.$$

2. Compute θ^{t+1} using (6.3).
end for

6.3.2 Alternating Online DCA versions

To exploit the knowledge of transition sample (S_t, A_t, S'_t, r_t) for each update step (6.3), we propose alternating versions of ODCA. Indeed, the computation of v^t (6.8) can be rewritten explicitly as

$$\begin{aligned} v_t &= 2\left[\langle \mathbf{A}^{(t,j_t)}, \theta_t \rangle + \mathbf{b}^{(t)}\right] \left[\mathbf{A}^{(t,j_t)} + 2\delta_t \mathbf{A}^{(t,\overline{j}_t)} \mathbb{1}_{\langle \mathbf{A}^{(t,\overline{j}_t)}, \theta \rangle + \mathbf{b}^{(t)} = 0}(\theta_t)\right] \qquad (6.11) \\ &= 2\left[\langle \mathbf{A}^{(t,j_t)}, \theta_t \rangle + \mathbf{b}^{(t)}\right] \Big\{\gamma\phi(S'_t, a_{j_t}) - \phi(S_t, A_t) + \\ &\quad 2\delta_t \left[\gamma\phi(S'_t, a_{\overline{j}_t}) - \phi(S_t, A_t)\right] \mathbb{1}_{\langle \mathbf{A}^{(t,\overline{j}_t)}, \theta \rangle + \mathbf{b}^{(t)} = 0}(\theta_t)\Big\}. \qquad (6.12) \end{aligned}$$

When we change the transition at the step t that is $(S'_t, a) \neq (S_t, A_t)$, the procedure to update θ^{t+1} can be performed alternatively in terms of $\phi(S_t, A_t)$, $\phi(S'_t, a_{j_t})$, $\phi(S'_t, a_{\overline{j}_t})$ as Figure 6.1. In particular, the vector $\overline{\theta}^{t+1}$ for the current state-action (S_t, A_t) is used to update θ^{t+1} for the next state-action (S'_t, a). Thus, the quality of θ^{t+1} can be improved.

Consequently, we derive an alternating version of ODCA (resp. ODCA1) given by Algorithm 6.3 (AODCA) (resp. Algorithm 6.4 (AODCA1)).

Algorithm 6.3 Alternating Online DCA for solving (6.2) (AODCA)

Initialize: let $\{\eta_t\}$ be the sequence of step sizes, θ^1 be the initial point
for $t = 1, 2, \ldots, T$ **do**
1. Choose $j_t \in I_t(\theta^t)$ and $\overline{j}_t \in I_t(\overline{a}_t)$ using (6.4).
2. Compute θ^{t+1} using Procedure 1.
end for

Input: sample (S_t, A_t, S'_t, r_t), basic function ϕ, weight θ^t, indices $j_t, \bar{j}_t$, and $\delta_t \in [-1, 1]$
Output: weight θ^{t+1}

if $\langle \mathbf{A}^{(t,\bar{j}_t)}, \theta^t \rangle + \mathbf{b}^{(t)} \neq 0$ **then**

  if $(S'_t, a_{j_t}) = (S_t, A_t)$ **then**
    $\theta^{t+1} = \theta^t - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)} \right] (\gamma - 1)\phi(S_t, A_t)$
  else
    $\overline{\theta}^{t+1} = \theta^t + \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)} \right] \phi(S_t, A_t)$
    $\theta^{t+1} = \overline{\theta}^{t+1} - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \overline{\theta}^{t+1} \rangle + \mathbf{b}^{(t)} \right] \gamma\phi(S'_t, a_{j_t})$
  end if
else if $a_{\bar{j}_t} = a_{j_t}$ **then**

  if $(S'_t, a_{j_t}) = (S_t, A_t)$ **then**
    $\theta^{t+1} = \theta^t - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)} \right] (\gamma - 1)(1 + 2\delta_t)\phi(S_t, A_t)$
  else
    $\overline{\theta}^{t+1} = \theta^t + \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)} \right] (1 + 2\delta_t)\phi(S_t, A_t)$
    $\theta^{t+1} = \overline{\theta}^{t+1} - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \overline{\theta}^{t+1} \rangle + \mathbf{b}^{(t)} \right] \gamma(1 + 2\delta_t)\phi(S'_t, a_{j_t})$
  end if
else

  if $(S'_t, a_{j_t}) = (S_t, A_t)$ **then**
    $\overline{\theta}^{t+1} = \theta^t - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)} \right] (\gamma - 1 - 2\delta_t)\phi(S_t, A_t)$
    $\theta^{t+1} = \overline{\theta}^{t+1} - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \overline{\theta}^{t+1} \rangle + \mathbf{b}^{(t)} \right] 2\gamma\delta_t\phi(S'_t, a_{\bar{j}_t})$
  else if $(S'_t, a_{\bar{j}_t}) = (S_t, A_t)$ **then**
    $\overline{\theta}^{t+1} = \theta^t + \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)} \right] (1 + 2\delta_t(1 - \gamma))\phi(S_t, A_t)$
    $\theta^{t+1} = \overline{\theta}^{t+1} - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \overline{\theta}^{t+1} \rangle + \mathbf{b}^{(t)} \right] \gamma\phi(S'_t, a_{j_t})$
  else
    $\overline{\theta}^{t+1} = \theta^t + \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)} \right] (1 + 2\delta_t)\phi(S_t, A_t)$
    $\overline{\overline{\theta}}_{t+1} = \overline{\theta}^{t+1} - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \overline{\theta}^{t+1} \rangle + \mathbf{b}^{(t)} \right] \gamma\phi(S'_t, a_{j_t})$
    $\theta_{t+1} = \overline{\overline{\theta}}_{t+1} - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \overline{\overline{\theta}}_{t+1} \rangle + \mathbf{b}^{(t)} \right] 2\delta_t\gamma\phi(S'_t, a_{\bar{j}_t})$
  end if
end if

Figure 6.1 – Procedure 1: compute θ^{t+1} in the alternating version of ODCA

Algorithm 6.4 AODCA when $\overline{a}_t = \theta^t$ for all t (AODCA1)

Initialize: let $\{\eta_t\}$ be the sequence of step sizes, θ^1 be the initial point
for $t = 1, 2, \ldots, T$ **do**
1. Choose $j_t \in I_t(\theta^t)$ using (6.4).
2. Compute θ^{t+1} by the following procedure

if $(S'_t, a_{j_t}) = (S_t, A_t)$ **then**
$\theta^{t+1} = \theta^t - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)}\right](\gamma - 1)\phi(S_t, A_t)$.
else
$\overline{\theta}^{t+1} = \theta^t + \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \theta^t \rangle + \mathbf{b}^{(t)}\right]\phi(S_t, A_t)$.
$\theta^{t+1} = \overline{\theta}^{t+1} - \eta_t \left[\langle \mathbf{A}^{(t,j_t)}, \overline{\theta}^{t+1} \rangle + \mathbf{b}^{(t)}\right]\gamma\phi(S'_t, a_{j_t})$.
end if
end for

6.4 Numerical experiments

In the numerical experiments, we describe the performance of the proposed online algorithms ODCA, ODCA1 and their alternating versions AODCA, AODCA1, and compare our notable algorithm with some standard RL algorithms: Q-learning [118, 126], SARSA [114, 117]. Our experiment consists of two well-known benchmarks: mountain car problems [118] and pole balancing problems [6, 84] described as follows.

6.4.1 Descriptions of mountain car and pole balancing problems

■ Mountain car problem: a continuous control task of driving an underpowered car up a steep mountain road [118]. The state of the car at step t is characterized by its position, denoted by q_t, and its velocity, denoted by $\dot{q}_t$. The simplified physics which the car moves according to is described in detail in [118]. In this case, the state space is continuous and $\mathcal{S}_c = \{(q, \dot{q}) : \text{bl}_q \leq q \leq \text{br}_q, \text{bl}_{\dot{q}} \leq \dot{q} \leq \text{br}_{\dot{q}}\}$ where the left bounds $\text{bl}_q = -1.5$, $\text{bl}_{\dot{q}} = -0.07$ and the right bounds $\text{br}_q = 0.45$, $\text{br}_{\dot{q}} = 0.07$. If the next position $q_{t+1} \leq \text{bl}_q$, then the next velocity $\dot{q}_{t+1}$ is set to 0. When $q_{t+1} \geq \text{br}_q$, the goal at the top of the mountain is reached. Each episode of the task starts from "the foot of the mountain" (i.e. $q = -0.5$, $\dot{q} = 0$) and ends when either the goal or the maximum number of steps per episode is reached. The action space is $\mathcal{A} = \{-1, 0, 1\}$, corresponding to three actions: full throttle forward, full throttle reverse and zero throttle. The reward r_t is set to 100 if the goal is reached, -1 otherwise. In our experiment, we discretize the continuous state space $\mathcal{S}_c$ into a discrete state space denoted by $\mathcal{S}$. In order to convert a continuous state $s \in \mathcal{S}_c$ into $\mathcal{S}$, we check which state in $\mathcal{S}$ is the closest to s and return it as a state discretization of s. As for basis functions, for each $(s_i, a_j) \in \mathcal{S} \times \mathcal{A}, i = 1, \ldots, N_{\mathcal{S}}$ and $j = 1, \ldots, N_{\mathcal{A}}$, we define

$$\phi(s_i, a_j) = (\phi_k(s_i, a_j))_{k=1,\ldots,N_{\mathcal{S}}.N_{\mathcal{A}}} \tag{6.13}$$

where

$$\phi_k(s_i, a_j) = \begin{cases} 1 & \text{if } k = i + (j-1).N_{\mathcal{S}}, \\ 0 & \text{otherwise,} \end{cases} \tag{6.14}$$

and thus, the number of basis functions is $d = N_{\mathcal{S}}.N_{\mathcal{A}}$.

■ Pole balancing problem: a control task which consists in stabilizing a pole hinged to a cart by applying forces to the cart [6, 84, 118]. In this problem, the state space $S_c = \{(q, \dot{q}, x, \dot{x}) \in \mathbb{R}^4\}$ is continuous, where $q, \dot{q}$ are respectively the vertical angle (in radians), the angular velocity (in radians per second) of the pole, and $x, \dot{x}$ are respectively the cart position (in meters) from the center and its velocity (in meters per second). The general physics of the cart and pole is presented in [6]. In our experiments, each episode starts from the "zero" state (i.e. $q = \dot{q} = x = \dot{x} = 0$) and ends when the pole falls (i.e. $|q| > \pi/4$) or the cart moves off the track (i.e. $|x| > 4$). The action space is $\mathcal{A} = \{-10, -9, \ldots, 9, 10\}$, corresponding to the forces (in newtons) to the cart. At step t, the reward is defined as $r_t = -10000 - 50|q_t| - 100|x_t|$ if the corresponding episode is terminated, and $r_t = 10 - 1000x^2 - 5|q_t| - 10\dot{x}_t$ otherwise. Similarly to mountain car problem, we also consider a discrete state space $\mathcal{S}$ and the similar basis functions.

6.4.2 Set up experiments

All experiments were implemented in MATLAB R2013b and performed on a PC Intel(R) Xeon(R) CPU E5-2630 v2, @ 2.60GHz of 32GB RAM. All parameters for the simulation of both above problems are set the same as in the implementation by José Antonio Martín H. and the source codes of Q-learning, SARSA are available on his homepage [2]. The initial point for all algorithms is $\theta^1 = 0 \in \mathbb{R}^d$. As mentioned in Section 4.1, we here consider the ε-greedy method to make the trade-off between exploitation and exploration. In particular, the exploration schedule is $\varepsilon_t = \varepsilon_0.\varepsilon^{\text{epi}(t)}$ where the decay rate $\varepsilon = 0.99$, the initial exploration probability $\varepsilon_0 = 0.01$ and the function epi(t) returns the index of episode which the step t belongs to. The discount factor for mountain car problems (resp. pole balancing problems) is $\gamma = 0.97$ (resp. $\gamma = 0.95$). The maximum number of episodes is set to 1000. The maximum number of steps per episode is set to 1000. The learning rate is $\eta_t = \eta$ for all t where η is chosen from the set $\{0.1, 0.2, \ldots, 2\}$ such that it gives the smallest (resp. largest) averaged number of steps at the last episode for mountain car problems (resp. pole balancing problems). As for the ODCA, AODCA algorithms, the value $\delta_t = -0.5$ and $\overline{a}_t = \theta^1$ for all t. For mountain car problems, the criteria to evaluate the effectiveness of the proposed algorithms are the average number of steps at the last episode and CPU time (in seconds), while for pole balancing problems we are interested in the total number of steps of all episodes and the CPU time per step (in seconds).

2. `https://jamh-web.appspot.com/download.htm#Reinforcement_Learning:`

6.4.3 Computational results

Two purposes of our experiments in this chapter are to, first, give a comparison of four algorithms: ODCA, ODCA1, AODCA, AODCA1 in mountain car problems and, second, compare between the notable algorithms AODCA1 and Q-learning, SARSA in both problems.

Concerning the first purpose, we tune the parameter η for each algorithm over 5 different runs and, with the best value of η, we test the algorithm over 50 runs. Fig. 6.2 shows the average number of steps of four ODCA based algorithms over 50 runs in the number of episodes in mountain car problems. In addition, each bar in Fig. 6.2 represents the value of standard deviation number of steps over 50 runs divided by 8 (this makes the figures clearer). Table 6.1 reports the average/standard deviation number of steps at the last episode and CPU time (in seconds) of the four algorithms.

Table 6.1 – Average/standard deviation number of steps at the last episode and CPU time (in seconds) of two ODCA based algorithms and their alternating version over 50 runs in mountain car problems. Bold values indicate the best results.

Algorithms	Number of steps at the last episode	CPU
ODCA	295.4.2 ± 111.3	52.6
ODCA1	219.1 ± 54.1	42.5
AODCA	278.9 ± 90.84	43.3
AODCA1	**182.3** ± 39.1	**25.5**

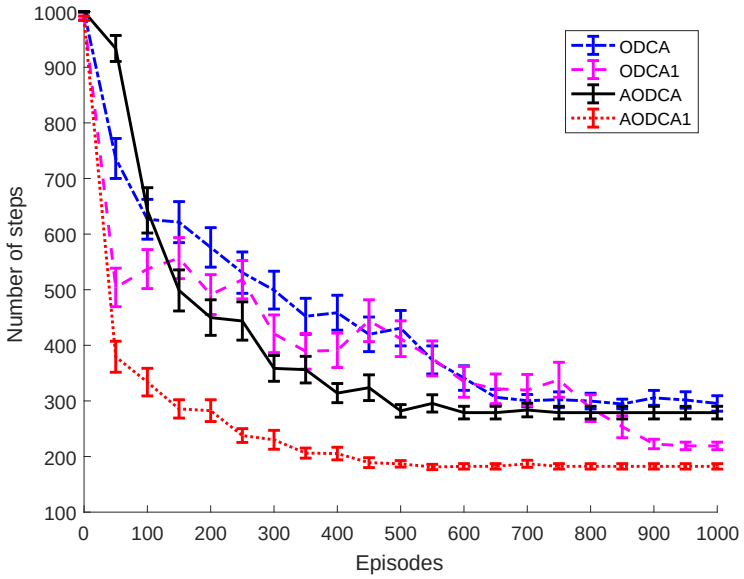


Figure 6.2 – Average/standard deviation number of steps of two ODCA based algorithms and their alternating version in the number of episodes over 50 runs in mountain car problems. Each bar represents the value of standard deviation number of steps divided by 8. The lower the curve, the better the performance.

For the second purpose, similarly, we tune the parameter over 10 different runs and run each algorithm over 100 runs in both mountain car and pole balancing problems. The average/standard deviation results of AODCA1 and Q-learning, SARSA over these 100 runs are reported in Fig. 6.3 and Fig. 6.4. These results at the last episode and CPU time for mountain car problems are given in Table 6.2. Table 6.3 presents the total number of steps of all episodes, CPU time (in seconds) and the ratio of CPU time to the total number of steps of the three algorithms in pole balancing problems.

Table 6.2 – Average/standard deviation number of steps at the last episode and CPU time (in seconds) of AODCA1 and Q-learning, SARSA over 100 runs in mountain car problems. Bold values indicate the best results.

Algorithms	Number of steps at the last episode	CPU
SARSA	210.2 ± 54.2	**24.1**
Q-learning	222.1 ± 55.7	26.1
AODCA1	**184.8** ± 35.0	26.8

Table 6.3 – Total number of steps of all episodes and CPU time (in seconds) and the ratio of CPU time to the total number of steps (denoted by CPU/step) of AODCA1 and Q-learning, SARSA over 100 runs in pole balancing problems.

Algorithms	Total number of steps	CPU	CPU/step
SARSA	637 899	**26.0**	4.0e-5
Q-learning	780 088	37.3	4.7e-5
AODCA1	**791 213**	37.9	4.7e-5

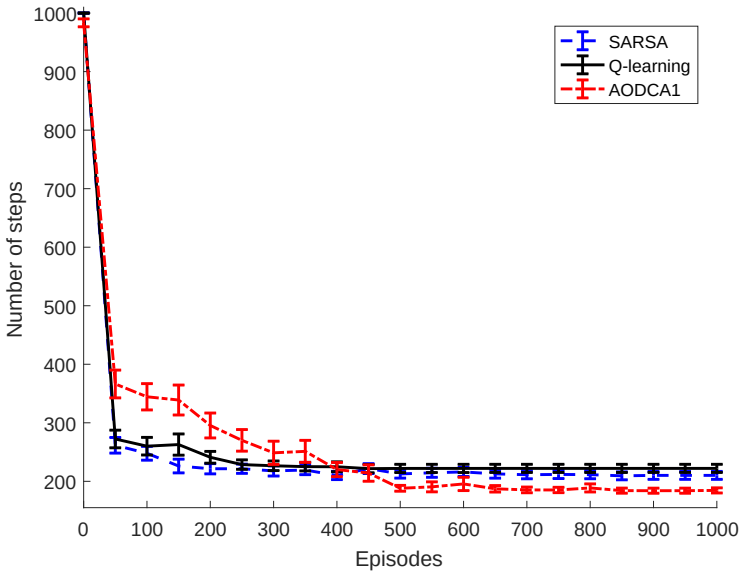


Figure 6.3 – Average/standard deviation number of steps of AODCA1 and Q-learning, SARSA in the number of episodes over 100 runs in mountain car problems. The lower the curve, the better the performance.

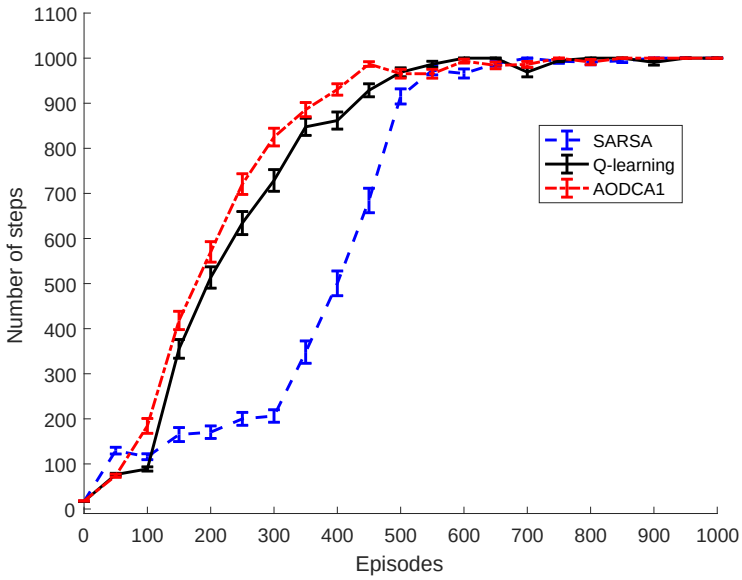


Figure 6.4 – Average/standard deviation number of steps of AODCA1 and Q-learning, SARSA in the number of episodes over 100 runs in pole balancing problems. The upper the curve, the better the performance.

Comments on computational results:

Concerning mountain car problems, the smaller the number of steps, the better the algorithm (due to the fact that one needs to drive the car such that it can move up to the top of the mountain as soon as possible). From Fig. 6.2 and Table 6.1, we see that the alternating versions of ODCA algorithms are more efficient than their original versions. Among our four proposed algorithms, AODCA1 is the best in both quality of solutions and rapidity. From Fig. 6.3 and Table 6.2, our online algorithm AODCA1 gives the average number of steps better than SARSA, Q-learning algorithms when the number of episodes is greater than 400, and more precisely, the policy generated by AODCA1 is more efficient, stable than SARSA and Q-learning. Although the complexity of AODCA1 at each iteration is larger than that of SARSA and Q-learning, the CPU time of AODCA1 is comparable with SARSA and Q-learning (see Table 6.2). This can be explained by the fact that the number of steps obtained by AODCA1 is smaller than SARSA and Q-learning.

For pole balancing problems, from the fact that one wants to stabilize the pole as long as possible, it derives that the larger the number of steps, the better the algorithm. From Fig. 6.4 and Table 6.3, we observe that AODCA1 keeps the pole balancing longer than SARSA and Q-learning. Specifically, when the number of episodes is less than 500, AODCA1 is the best in terms of number of steps, SARSA is the worst and when the number of steps is greater than or equal to 500, the three algorithms are comparable. Concerning CPU time per step, AODCA1 is fairly similar to both SARSA and Q-learning although the complexity of AODCA1 at each step is larger. This fact is because the number of steps of AODCA1 is the best in most cases.

In summary, we can say that the alternating versions of ODCA based algorithms are more efficient than the original versions in all criteria. In addition, for both considered problems, AODCA1 could be the algorithm realizing the best trade-off between the quality and the rapidity.

6.5 Conclusions

We have investigated Online DC programming and Online DCA for RL in online mode via the OBR minimization approach. Considering the ℓ_2-norm formulations of OBR, we have developed an Online DCA based algorithm (ODCA) which enjoys the online stability property. We have indicated that the classic residual gradient algorithm for RL is a special case of our algorithm. To exploit the knowledge of transition samples, we have proposed its alternating versions. Comparing with the two standard RL algorithms (Q-learning and SARSA) on two benchmarks – mountain car and pole balancing problems, our ODCA based approach is proven to be effective due to its best trade-off between the quality and the rapidity. In the future, we plan to extend our results to develop more efficient RL techniques, for example, RL with eligibility traces.

Chapter 7

Applications to Stochastic Shortest Path problems: DCA Approaches via Cardinality Minimization and Reinforcement Learning[1]

Abstract: *The chapter concerns the Stochastic Shortest Path (SSP) problem for a single independent vehicle on a road network using the probability tail model-based criterion. The SSP problem aims to search an optimal path that maximizes the probability of reaching destination before a particular deadline. There exist two different reformulations of this problem: first, a cardinality minimization formulation (cardinality of a vector is the number of nonzero elements in that vector) and second, an RL formulation. For the first formulation, the maximization problem is reformulated as a cardinality minimization problem with zero-one variables. Recently, some algorithms have been proposed by approximating the cardinality term due to its discontinuity, however, without treating zero-one variables. In this chapter, we develop a DC programming and DCA based approach for solving this cardinality problem. We investigate first a DC approximation approach for the cardinality term and then an exact penalty technique for the zero-one variables. The resulting optimization reformulation can be expressed as a DC program for which DCA is applied. We propose a DCA based algorithm, namely Card-DCA, for solving this SSP problem. Numerical experiments in an artificial road network with different given deadlines show the efficiency of the proposed algorithm in terms of both quality and rapidity when compared with the existing algorithms. For the RL formulation, we take into account the ℓ_1-norm optimization problem in which the given set of samples is defined based on the travel time data on the road network and hence a DCA based algorithm, namely RL-DCA, is proposed for these SSP problems. In comparison with Card-DCA, our proposed algorithm RL-DCA is capable of improving the probability of reaching destination.*

1. The material of this chapter is developed from the following work:
[1]. Vinh Thanh Ho, Hoai An Le Thi. A DCA Approach for the Stochastic Shortest Path Problem in Vehicle Routing. *Accepted* by IESM 2017: 7th International Conference on Industrial Engineering and Systems Management.

7.1 Introduction

The Stochastic Shortest Path (SSP) problem in vehicle routing on a given road network is the problem of finding the optimal path in which the length of road links is random [35, 112, 82]. As one fundamental problem in network studies, the SSP problem has attracted attention from researchers in many areas of, e.g., transportation engineering, computer science and operations research. In this chapter, we consider the SSP problem for a single independent vehicle where the length of road links is represented by their travel time. In fact, the randomness of travel time is due to many uncertain traffic conditions, e.g., the quality of vehicle, traffic jam, weather and so on. In such a stochastic environment, there are many different criteria for an optimal path, e.g., least expected travel [85], mean-risk model [89], path optimality index [112], probability tail (PT) model [35, 90, 34], which are reviewed in detail in [18]. Because of perspicuity for drivers and solving daily issues, we are interested in SSP problems using the PT model-based criterion where the optimal path is the one that maximizes the probability of reaching destination before a given deadline, which are known as the arriving-on-time problems. An optimization formulation of PT model-based SSP problems on a road network can be expressed as the following. In this chapter, we use bold letters to denote matrices and vectors, normal letter to denote scalars.

7.1.1 An optimization formulation of PT model-based SSP problems

A road network can be modeled as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A}_r)$ where $\mathcal{V} = \{1, \ldots, n\}$ represents the set of nodes and $\mathcal{A}_r \subseteq \{(v_1, v_2) : v_1, v_2 \in \mathcal{V}, v_1 \neq v_2\}$ represents the set of arcs with the size of m, (v_1, v_2) represents an arc from node v_1 to node v_2. As mentioned above, the goal of the PT model-based SSP problem is to find the path $\mathbf{x}$ that maximizes the probability of arriving at the destination d from the origin o not later than a given deadline T (more precisely, T is the remaining time to deadline) based on the distributed travel time samples $\mathbf{w}$. The corresponding optimization formulation can be written as follows [26].

$$\max_{\mathbf{x}} \ \mathbb{P}(\mathbf{w}^\top \mathbf{x} \leq T) \tag{7.1}$$

s.t. $\forall v_1 \in \mathcal{V}$:

$$\sum_{\substack{v_2 \in \mathcal{V}, \\ (v_1, v_2) \in \mathcal{A}_r}} x(v_1, v_2) - \sum_{\substack{v_2 \in \mathcal{V}, \\ (v_2, v_1) \in \mathcal{A}_r}} x(v_2, v_1) = \begin{cases} 1 & \text{if } v_1 = o, \\ -1 & \text{if } v_1 = d, \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathbf{x} \in \{0, 1\}^m$ such that its each component $x(v_1, v_2)$ refers to an arc $(v_1, v_2) \in \mathcal{A}_r$ and $x(v_1, v_2) = 1$ if and only if (v_1, v_2) is on the concerned path. This formulation is equivalent to the following problem:

$$\begin{aligned} & \min_{\mathbf{x}} \ \mathbb{P}(\mathbf{w}^\top \mathbf{x} > T) \\ \text{s.t.} \quad & \mathbf{M}\mathbf{x} = \mathbf{b}, \ \mathbf{x} \in \{0, 1\}^m, \end{aligned}$$

where $\mathbf{M} \in \mathbb{R}^{n\times m}$ is a node-arc incidence matrix (see, e.g., [2] for more details), $\mathbf{b} \in \mathbb{R}^n$ whose elements are zero except the two elements corresponding to node o and node d, which are 1 and -1, respectively.

7.1.2 Related works

Most related studies for solving the problem (7.2) are based on the common assumptions about specific distributions of travel times, correlations between travel times on different road links, deadlines (see, e.g., [90, 77, 35, 34, 88] for more details). However, on real road networks, these assumptions are hard to be satisfied. Without working on these assumptions, Cao et al. [19] proposed a data-driven approach which is only based on the travel time samples of all road links. In this chapter, to solve the problem (7.2), we consider two reformulations of these PT model-based SSP problems based on the data-driven approach: cardinality minimization and Markov Decision Process (MDP). In the following, we present the related works of these two reformulations.

7.1.2.1 Cardinality minimization reformulation

By reformulating (7.2), a cardinality minimization problem with the zero-one variables is described as follows (see [19]).

$$\min \left\{ \texttt{Card}(C(\mathbf{x})) : \mathbf{x} \in \{0,1\}^m,\ \mathbf{M}\mathbf{x} = \mathbf{b} \right\}, \tag{7.2}$$

where $\texttt{Card}(\mathbf{z})$ is the number of nonzero elements in $\mathbf{z}$, vector $C(\mathbf{x}) = \left([\mathbf{w}_1^\top \mathbf{x} - T]^+, \ldots, [\mathbf{w}_S^\top \mathbf{x} - T]^+\right)^\top$, $\{\mathbf{w}_i\}_{i=1,\ldots,S}$ denotes the set of travel time samples on all arcs with the size of S and $[\cdot]^+ = \max\{0, \cdot\}$. There exists a double difficulty in solving this cardinality minimization problem: how to treat, first, the cardinality term and, second, the zero-one variables. In [19], the authors approximated the cardinality term by ℓ_p-norm ($0 < p \leq 1$) functions, the logarithmic approximation function and their combination, and thus, proposed the so-called ℓ_1-norm algorithm and reweighted-ℓ_1 algorithms. The ℓ_1-norm algorithm only solves one Mixed Integer Linear Program (MILP), while the reweighted-ℓ_1 algorithms require solving successive MILPs. There is one limitation of this work: expensive computations in solving MILPs. In [20], the authors avoided solving MILPs by relaxing them to linear programs as well as using partial Lagrange multiplier techniques and a property of incidence matrix $\mathbf{M}$ to guarantee that their obtained solutions are zero-one. Concerning the choice of good approximation of the cardinality term, it is known that there exist approximation functions in [68] which have been proved to be more efficient than those in [19, 20].

In the optimization and machine learning community, the cardinality minimization problem (7.2) with the continuous variables instead of zero-one variables is known as an ℓ_0-norm problem (see, e.g, [68]) where ℓ_0-norm is defined as $\texttt{Card}$. A variety of works relating to ℓ_0-norm problems have been recently reviewed in the seminal work of Le Thi et al. [68]. According to the way to treat the ℓ_0-norm, they are divided into three

main categories: convex approximation, nonconvex approximation and nonconvex exact reformulation. Le Thi et al. [68] developed a unified DC (Difference of Convex functions) approximation and DCA (DC Algorithm) based approach for the ℓ_0-norm problem and indicated that corresponding DCA schemes cover all standard algorithms in nonconvex approximation approaches as special versions. Although there are many studies for solving the ℓ_0-norm problems, these problems are actually hard to solve due to ℓ_0-norm. In this chapter, we follow the continuous DC approximation function in [68] to tackle the discontinuous `Card` term.

7.1.2.2 MDP reformulation

In the literature, there are many SSP problems which are analyzed by means of the MDP framework (see e.g. [9, 11, 12]). These SSP problems can be seen as a well-known class of MDP problems where their different MDP formulations are considered (see [10, 21, 87, 100]). Concerning the PT model-based SSP problem, its MDP reformulation has been proposed in [21]. In particular, the state space, denoted by $\mathcal{S}$, is defined as the set of the pair of intersections (or nodes in the graph) and time-to-deadlines:

$$\mathcal{S} := \{s = (v, \tau) : v \in \mathcal{V}, \tau \in \Gamma\}$$

where Γ is the set of time-to-deadlines. Let us assume that the set Γ is finite and denote by $N_{\mathcal{S}}$, N_Γ respectively the size of the space $\mathcal{S}$, the set Γ. Thus, one has $N_{\mathcal{S}} = n.N_\Gamma$. The action space, denoted by $\mathcal{A}$, is defined as the set of directions:

$$\mathcal{A} := \{a_i\}_{i=1,\ldots,N_{\mathcal{A}}}.$$

The state transition probability $\mathcal{P}(s' = (v', \tau')|s = (v, \tau), a)$ represents the distribution of the random travel time, denoted by $t_{v,v'}$, on the road link (or arc) $(v, v') \in \mathcal{A}_r$ by the direction $a \in \mathcal{A}$ that takes the vehicle (or the agent) from the intersection v with the time-to-deadline to the next intersection v' with the remaining time-to-deadline τ' computed by $\tau' = \tau - t_{v,v'}$. The reward function, denoted by $\mathcal{R}$, at each state-action transition (s, a) represents the expected immediate reward received after taking the direction a for the intersection s with the time-to-deadline τ. The discount factor is denoted by γ.

In this chapter, we concern these SSP problems in which the full knowledge of MDP in terms of $\mathcal{P}$ and $\mathcal{R}$ is not given in advance, and thus the aforementioned RL approach via state-action value functions will be investigated. By setting γ to be 1 and the immediate reward to be 1 if the action moves the vehicle from the current intersection to the deadline before the deadline, and 0 otherwise, the optimal state-action value function defined by (4.1) and (4.2) is exactly the same as the probability of arriving on time defined by (7.1). In the recent work [21], the authors use Q-learning to approximate the optimal state-action value function for discrete/continuous deadlines via a neural network approximation where the sample at each step is chosen through e.g. the Softmax strategy. In this chapter, we focus on estimating the probability of arriving on time by reinforcement learning (RL) techniques in batch mode (Batch RL), i.e. based on the given fixed set of transition samples (more precisely the travel time samples on arcs) with discrete deadlines.

7.1.3 Our contributions

Our contributions are multiple, which can be classified into two parts corresponding to two reformulation problems just mentioned previously. Concerning the cardinality problem (7.2), we develop a DCA approach for solving it via two following main steps with respect to the `Card` term and the zero-one variables. First, we use a DC approximation of the `Card` term, which results in approximating the problem (7.2) to a DC minimization problem with zero-one variables. Second, we exploit an exact penalty technique in DC programming for treating these zero-one variables, which has been widely studied in, e.g., [70, 69, 97]. Consequently, the resulting combinatorial optimization problem is equivalently reformulated as a continuous problem of the standard form of a DC program. Thus, a DCA based algorithm, namely Card-DCA is designed for solving the considered SSP problem. In addition, as an illustrative experiment for the arriving on time problem, we demonstrate the efficiency of our approach for the problem (7.2) in terms of both the quality of obtained paths and the rapidity on a road network in comparison with the ℓ_1-norm algorithm and the reweighted-ℓ_1 algorithm. As for the Batch RL based SSP problem, we take into account the ℓ_1-norm optimization problem in which the given samples are defined based on the travel time data on the network and hence another DCA based algorithm, namely RL-DCA, is proposed for these SSP problems. Several numerical experiments on the artificial road networks are conducted in order to compare two DCA approaches for these problems, in particular Card-DCA and RL-DCA algorithms.

The rest of the chapter is organized as follows. In Section 7.2, we present two DCA approaches for SSP problems in which we show how to express the cardinality minimization problem as well as the ℓ_1-norm optimization problem as DC programs, and then design DCA based algorithms for the resulting optimization reformulation. Section 7.3 reports the numerical results on artificial road networks which is followed by some conclusions in Section 7.4.

7.2 DCA Approaches for the reformulations of PT model-based SSP problems

7.2.1 The first reformulation: cardinality problem (7.2)

A DC formulation of the cardinality problem (7.2)

By introducing the slack variable $\mathbf{y}$, the following proposition (whose proof is evident) will justify our considered optimization formulation (7.3) whose feasible set is bounded.

Proposition 7.1. *If $\mathbf{x}^*$ is an optimal solution to the problem (7.2), then $(\mathbf{x}^*, \mathbf{y}^*)$ is an optimal solution to the following problem*

$$\min \left\{ \texttt{Card}(\mathbf{y}) : \begin{array}{l} \mathbf{Mx} = \mathbf{b},\ \mathbf{x} \in \{0,1\}^m, \mathbf{y} \geq 0, \\ y_i \geq \mathbf{w}_i^\top \mathbf{x} - T,\ \forall i = 1, \ldots, S, \\ y_i \leq [||\mathbf{w}_i||_1 - T]^+,\ \forall i = 1, \ldots, S \end{array} \right\}, \tag{7.3}$$

where $\mathbf{y} = (y_1, \ldots, y_S) \in \mathbb{R}^S$, $y_i^* = [\mathbf{w}_i^\top \mathbf{x}^* - T]^+,\ \forall i = 1, \ldots, S$. *Conversely, if* $(\mathbf{x}^*, \mathbf{y}^*)$ *is an optimal solution to* (7.3)*, then* $\mathbf{x}^*$ *is an optimal solution to* (7.2).

Note that the upper bounds for the variable $\mathbf{y}$ are added in (7.3) due to the fact that for all $\mathbf{x} \in \{0,1\}^m$, $0 \leq [\mathbf{w}_i^\top \mathbf{x} - T]^+ \leq [||\mathbf{w}_i||_1 - T]^+,\ \forall i = 1, \ldots, S$. This boundedness of the feasible set is useful for the exact penalty techniques as well as the nonconvex exact reformulation approach for the problem (7.2) as discussed in Section 7.1.2.

In this chapter, we use a continuous DC approximation function of the `Card` term, specifically the exponential concave approximation [16], which is proven to be efficient in pratice (see, e.g., [57, 16, 68, 72]), as follows.

$$\texttt{Card}(\mathbf{y}) \approx \sum_{i=1}^{S} r_\eta(y_i),$$

where the function $r_\eta(t) = 1 - e^{-\eta|t|}$ for $t \in \mathbb{R}$ and the positive approximation parameter η. Thus, we derive the following approximate problem

$$\alpha := \min \left\{ \sum_{i=1}^{S} r_\eta(y_i) : \begin{array}{l} \mathbf{Mx} = \mathbf{b},\ \mathbf{x} \in \{0,1\}^m, \mathbf{y} \geq 0, \\ y_i \geq \mathbf{w}_i^\top \mathbf{x} - T,\ \forall i = 1, \ldots, S, \\ y_i \leq [||\mathbf{w}_i||_1 - T]^+, \forall i = 1, \ldots, S \end{array} \right\}. \tag{7.4}$$

In fact, solving the problem (7.4) is still difficult due to zero-one variables. Using the exact penalty techniques to treat these variables leads to an exact continuous reformulation of (7.4) which is of the standard form of a DC program for which DCA is applied. Let $p : [0,1]^m \to \mathbb{R}$ be the penalty function defined by $p(\mathbf{x}) := \sum_{i=1}^m \min\{x_i, 1 - x_i\}$. The problem (7.4) can be rewritten as follows:

$$\alpha = \min \left\{ f(\mathbf{x}, \mathbf{y}) := \sum_{i=1}^{S} r_\eta(y_i) : (\mathbf{x}, \mathbf{y}) \in \mathcal{K},\ p(\mathbf{x}) \leq 0 \right\},$$

where

$$\mathcal{K} := \left\{ (\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{m+S} : \begin{array}{l} \mathbf{Mx} = \mathbf{b},\ \mathbf{x} \in [0,1]^m, \mathbf{y} \geq 0, \\ y_i \geq \mathbf{w}_i^\top \mathbf{x} - T, \forall i = 1, \ldots, S, \\ y_i \leq [||\mathbf{w}_i||_1 - T]^+, \forall i = 1, \ldots, S \end{array} \right\}$$

is the bounded polyhedral convex set defined by $n + S$ linear constraints and upper/lower bound constraints. It leads to the corresponding penalized problem (τ being the positive penalty parameter)

$$\alpha(\tau) := \min \left\{ f(\mathbf{x}, \mathbf{y}) + \tau p(\mathbf{x}) : (\mathbf{x}, \mathbf{y}) \in \mathcal{K} \right\}. \tag{7.5}$$

It follows from Theorem 7 in [70] that there exists $\tau_0 > 0$ such that for all $\tau > \tau_0$, the two problems (7.4) and (7.5) are equivalent, in the sense that they have the same optimal value and the same solution set.

Considering the problem (7.5) with a sufficiently large number τ, we use the following DC decomposition of r_η [68]:

$$r_\eta(t) = g(t) - h(t),\ \forall t \in \mathbb{R}, \tag{7.6}$$

where $g(t) = \eta|t|$ and $h(t) = \eta|t| - 1 + e^{-\eta|t|}$. Thus, a DC reformulation of the problem (7.5) can be expressed as follows.

$$\min\left\{F(\mathbf{x},\mathbf{y}) = G(\mathbf{x},\mathbf{y}) - H(\mathbf{x},\mathbf{y}) : (\mathbf{x},\mathbf{y}) \in \mathbb{R}^{m+S}\right\}, \tag{7.7}$$

where $G(\mathbf{x},\mathbf{y}) = \chi_{\mathcal{K}}(\mathbf{x},\mathbf{y}) + \sum_{i=1}^{S} g(y_i)$ and $H(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{S} h(y_i) - \tau p(\mathbf{x})$. Obviously, (7.7) is a polyhedral DC program where the first DC component $G(\mathbf{x},\mathbf{y})$ is polyhedral convex.

DCA for solving the polyhedral DC program (7.7)

According to the generic DCA scheme, at each iteration k, after computing one subgradient $(\overline{\mathbf{x}}^k, \overline{\mathbf{y}}^k) \in \partial H(\mathbf{x}^k, \mathbf{y}^k)$, the calculation of $(\mathbf{x}^{k+1}, \mathbf{y}^{k+1})$ is reduced to solve the following convex program

$$\min\left\{G(\mathbf{x},\mathbf{y}) - \left\langle(\overline{\mathbf{x}}^k, \overline{\mathbf{y}}^k), (\mathbf{x},\mathbf{y})\right\rangle : (\mathbf{x},\mathbf{y}) \in \mathbb{R}^{m+S}\right\},$$

which is equivalent to a linear program

$$\min\left\{\left\langle(-\overline{\mathbf{x}}^k, \eta\mathbf{e} - \overline{\mathbf{y}}^k), (\mathbf{x},\mathbf{y})\right\rangle : (\mathbf{x},\mathbf{y}) \in \mathcal{K}\right\}, \tag{7.8}$$

where $\mathbf{e}$ is the vector of ones in the appropriate vector space.

Let $q(\mathbf{y}) := \sum_{i=1}^{S} h(y_i)$. We present how to compute a subgradient $(\overline{\mathbf{x}}^k, \overline{\mathbf{y}}^k) \in \partial H(\mathbf{x}^k, \mathbf{y}^k)$. Since $\mathbf{y}^k \geq 0$, we can choose $\overline{\mathbf{x}}^k \in \tau\partial(-p)(\mathbf{x}^k)$ and $\overline{\mathbf{y}}^k \in \partial q(\mathbf{y}^k)$ as follows. For $i = 1, \ldots, m$,

$$\overline{x}_i^k = \begin{cases} -\tau & \text{if } x_i^k < 1/2, \\ 0 & \text{if } x_i^k = 1/2, \\ \tau & \text{if } x_i^k > 1/2, \end{cases} \tag{7.9}$$

and for $j = 1, \ldots, S$,

$$\overline{y}_j^k = \begin{cases} \eta(1 - e^{-\eta y_j^k}) & \text{if } y_j^k > 0, \\ 0 & \text{if } y_j^k = 0. \end{cases} \tag{7.10}$$

DCA applied to (7.7) can be given by Algorithm 7.1 (Card-DCA).

Algorithm 7.1 DCA for solving the cardinality problem (7.7) (Card-DCA)

Initialization: Let $(\mathbf{x}^0, \mathbf{y}^0) \in \mathbb{R}^{m+S}$, $k = 0$.

repeat

1. Compute $(\overline{\mathbf{x}}^k, \overline{\mathbf{y}}^k) \in \partial H(\mathbf{x}^k, \mathbf{y}^k)$ using (7.9)-(7.10).
2. Compute $(\mathbf{x}^{k+1}, \mathbf{y}^{k+1})$, an optimal solution to (7.8).
3. $k = k + 1$.

until $||\mathbf{x}^k - \mathbf{x}^{k-1}|| + ||\mathbf{y}^k - \mathbf{y}^{k-1}|| \leq \varepsilon(1 + ||\mathbf{x}^{k-1}|| + ||\mathbf{y}^{k-1}||)$ or $|F(\mathbf{x}^k, \mathbf{y}^k) - F(\mathbf{x}^{k-1}, \mathbf{y}^{k-1})| \leq \varepsilon(1 + F(\mathbf{x}^{k-1}, \mathbf{y}^{k-1}))$

According to the convergence properties of the generic DCA scheme for polyhedral DC programs in Section 1.1, we deduce the following convergence properties of Card-DCA.

Theorem 7.1. *Convergence properties of* Card-DCA

i) Card-DCA *generates the sequence* $\{(\mathbf{x}^k, \mathbf{y}^k)\}$ *containing finitely many elements such that the sequence* $\{F(\mathbf{x}^k, \mathbf{y}^k)\}$ *is decreasing.*

ii) The sequence $\{(\mathbf{x}^k, \mathbf{y}^k)\}$ *converges to a critical point* $(\mathbf{x}^*, \mathbf{y}^*)$ *of* (7.7) *after a finite number of iterations.*

iii) For a sufficiently large number τ*, if there exists an iteration* $\overline{k}$ *such that* $\mathbf{x}^{\overline{k}} \in \{0,1\}^m$*, then* $\mathbf{x}^l \in \{0,1\}^m$ *for all* $l > \overline{k}$*.*

Proof. The properties *i)* and *ii)* are direct consequences of the convergence properties of DCA for a polyhedral DC program. The property *iii)* is proved similarly as Theorem 1 in [59] and Theorem 2 in [69]. □

7.2.2 The second reformulation: Batch RL problem

Remember that for the MDP reformulation as described in Section 7.1.2.2, the different optimization formulations of Optimal Bellman Residual (OBR) which were thoroughly studied in Chapter 5 can be made use of for solving these SSP problems. In this chapter, we concentrate on the ℓ_1-norm optimization formulation, which is briefly presented as follows.

ℓ_1-norm optimization formulation

In batch mode, we collect N samples $(S_i, A_i, S_i', R_i)_{i=1,\ldots,N}$ where R_i is the immediate reward received by taking the action A_i which moves the agent from the current state S_i to the next state S_i' for $i = 1, \ldots, N$. In this case, we have the ℓ_1-norm optimization formulation expressed as

$$\min \left\{ \overline{F}(\theta) := \sum_{i=1}^{N} |f_i(\theta)| : \theta \in \mathbb{R}^l \right\}, \tag{7.11}$$

where the real function

$$f_i(\theta) := \max_{j=1,\ldots,N_{\mathcal{A}}} \left\{ \langle \mathbf{A}^{(i,j)}, \theta \rangle + \mathbf{c}^{(i)} \right\},$$

the matrix $\mathbf{A}$ and the vector $\mathbf{c}$ are defined as

$$\mathbf{A}^{(i,j)} := \gamma \phi(S_i', a_j) - \phi(S_i, A_i), \ \mathbf{c}^{(i)} := R_i$$

for $i = 1, \ldots, N$, $j = 1, \ldots, N_{\mathcal{A}}$, ϕ is an l-dimensional basic function vector.

DCA for solving the ℓ_1-norm problem (7.11)

The problem (7.11) is a DC polyhedral program written in the form

$$\min\left\{\overline{G}(\theta) - \overline{H}(\theta) : \theta \in \mathbb{R}^l\right\}, \tag{7.12}$$

where two DC components defined by

$$\overline{G}(\theta) := \sum_{i=1}^{N} 2f_i^+(\theta) \text{ and } \overline{H}(\theta) := \sum_{i=1}^{N} f_i(\theta)$$

are polyhedral convex.

DCA applied to (7.11) is described as Algorithm 7.2 (RL-DCA) below.

Algorithm 7.2 DCA for solving (7.12) (RL-DCA)

Initialization: Let ε be a sufficiently small positive number. Let $\theta^0 \in \mathbb{R}^l$. Set $k = 0$.
repeat

1. Compute $w^k \in \partial\overline{H}(\theta^k)$:

$$w^k = \sum_{i=1}^{N} \mathbf{A}^{(i,j_i)}, \tag{7.13}$$

where $j_i \in \text{argmax}_{j=1,\ldots,N_{\mathcal{A}}}\langle \mathbf{A}^{(i,j)}, \theta^k\rangle$ for $i = 1,\ldots,N$.

2. Solve the linear program

$$\begin{cases} \min \sum_{i=1}^{N} 2t_i - \langle w^k, \theta\rangle, \\ \text{s.t. } \theta \in \mathbb{R}^l, \\ \quad t_i \geq 0, \forall i = 1,\ldots,N, \\ \quad t_i \geq \langle \mathbf{A}^{(i,j)}, \theta\rangle + \mathbf{c}^{(i)}, \forall i = 1,\ldots,N, \forall j = 1,\ldots,N_{\mathcal{A}}, \end{cases} \tag{7.14}$$

to obtain $(\theta^{k+1}, t_1^{k+1}, \ldots, t_N^{k+1}) \in \mathbb{R}^{l+N}$.

3. $k = k + 1$.

until $|\overline{F}(\theta^k) - \overline{F}(\theta^{k-1})| \leq \varepsilon(|\overline{F}(\theta^{k-1})| + 1)$ or $||\theta^k - \theta^{k-1}|| \leq \varepsilon(1 + ||\theta^{k-1}||)$.

In addition, RL-DCA enjoys the following convergence properties.

Theorem 7.2. *Convergence properties of* RL-DCA
i) RL-DCA *generates the sequence* $\{\theta^k\}$ *such that the sequence* $\{\overline{F}(\theta^k)\}$ *is decreasing.*
ii) The sequence $\{\theta^k\}$ *converges to a critical point* θ^* *of* (7.12) *after a finite number of iterations.*
iii) θ^* *is almost always a local minimizer to* (7.12)*. In particular, if* $I_i(\theta^*)$ *is a singleton for all* $i = 1, ..., N$*, then* θ^* *is a local minimizer to* (7.12)*.*

Remark 7.1. *Comparing the* RL-DCA *algorithm with the* Card-DCA *algorithm in terms of the complexity, we see that both algorithms require solving one linear program at each iteration. In particular, the linear program in* RL-DCA *has* $l + N$ *variables,*

$(N_{\mathcal{A}}+1).N$ constraints, and the one in Card-DCA *has $m+S$ variables, $n+2m+3S$ constraints where l is the number of the basic functions, N is the number of batch samples, $N_{\mathcal{A}}$ is the number of actions, m is the number of arcs, n is the number of nodes, S is the number of travel time data for each link.*

7.3 Numerical experiments

In this section, we study the performance of the proposed DCA algorithms RL-DCA, Card-DCA and compare them with existing algorithms for solving these SSP problems which will be listed in each experiment, on the arriving-on-time problem (i.e. maximizing the probability of arriving at the destination d from the origin o not later than the deadline T) for a single vehicle on road networks.

Our experiments consist of two parts. In the first experiment, we only concern the cardinality problem, in particular we consider the efficiency of Card-DCA in comparison with two existing algorithms in [19]: ℓ_1-norm, reweighted ℓ_1-norm minimization algorithms (we call ℓ_1, re-ℓ_1 respectively for short). The ℓ_1 algorithm replaces the `Card` term by the ℓ_1-norm function and solves one MILP. The re-ℓ_1 algorithm uses the combination of the ℓ_p-norm ($0 < p < 1$) and logarithmic approximation functions and requires solving successive MILPs. In fact, the approximation in the re-ℓ_1 algorithm is a special version of our DC approximation approach (see [68] for more details). It has been shown in [19] that when comparing between re-ℓ_1 and ℓ_1, in most cases, re-ℓ_1 (resp. ℓ_1) is better than the other in terms of the quality of solutions (resp. the rapidity). In the second experiment, we aim to compare between two DCA approaches for these PT model-based SSP problems, in particular RL-DCA and Card-DCA on artificial networks.

7.3.1 Experiment 1: Card-DCA for the cardinality problem

This experiment aims to make a comparison between the proposed Card-DCA algorithm and the existing algorithms (ℓ_1, re-ℓ_1) for the cardinality problem, which is implemented on a road network with 123 road links and 65 intersections of roads as in [19], corresponding to a directed graph with 123 arcs and 65 nodes respectively as Fig. 7.1 and the length of arcs is represented by the travel time of road links (in seconds).

Data set: Numerical experiments are conducted on the data sets (including the training set of size S_1 and test set of size S_2) of travel time samples on each link generated by some independent random distribution functions (e.g., normal, gamma and lognormal distributions). The (o, d) pair is chosen randomly from the given set. The deadline T is defined by $T = T_1 + \kappa(T_2 - T_1)$ where κ is a deadline coefficient, T_2 is the minimum longest travel time for all paths of (o,d) and T_1 is the shortest travel time with respect to the same path (see [19] for more details).

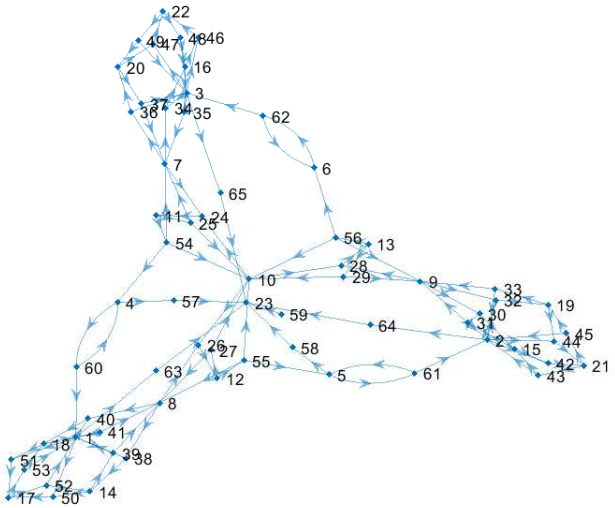


Figure 7.1 – A road network with 123 road links and 65 intersections [19]

Set up experiments: The parameters S_1, S_2, κ are set as follows: $S_1 \in \{100, 500, 1000\}$, $S_2 = 5000$, $\kappa \in \{0.2, 0.4, 0.6, 0.8, 1.0, 1.2\}$. The given set of (o,d) pairs is $\{(3,7), (7,3), (1,8), (8,1), (10,2), (2,10), (1,10), (10,1), (3,10), (10,3)\}$. Three random distribution functions are used with different parameters chosen randomly, specifically, normal distribution with mean in $\{5, 10, 15, 20\}$ and standard deviation in $\{10, 30, 50, 70\}$, gamma distribution with shape and scale in $\{1, 2, 3, 4, 5\}$, lognormal distribution with mean in $\{5, 10, 15, 20\}$ and variance in $\{10, 30, 50, 70\}$. This experiment is run 100 times and performed in MATLAB R2016a on a Laptop Intel(R) Core(TM) i7-2720QM CPU @ 2.20GHz of 4GB RAM. The software CPLEX 12.6 is used for solving linear programs in our algorithm, and solving MILPs in the existing algorithms. Card-DCA and re-ℓ_1 use the same zero starting point, their approximation parameter η is set to 0.01. For re-ℓ_1, p =0.5. The default tolerance is $\varepsilon = 10^{-4}$. The penalty parameter τ will be updated at each iteration.

Comparison criteria of algorithms: We are interested in the following aspects: accuracy (in %) and CPU time (in seconds). The accuracy of each algorithm is computed on the test set by the percentage of the probability of arriving on time of the path obtained by the algorithm relative to the maximum probability of all possible paths by the enumeration method.

Descriptions of tables of results: The average accuracy and average CPU time obtained by three algorithms Card-DCA, ℓ_1 and re-ℓ_1 on 100 runs with the different deadline coefficients κ and random distributions are reported in Tables 7.1, 7.2. The values in 2nd-4th columns of Table 7.1 represent the algorithms' accuracy in case the size of training set $S_1 = 100, 500, 1000$, respectively. These descriptions are similar to Table 7.2. Bold values in Tables 7.1, 7.2 are the best results in each column.

Table 7.1 – Accuracy (%) of Card-DCA and ℓ_1-norm algorithm (ℓ_1), reweighted-ℓ_1 algrorithm (re-ℓ_1) on 100 runs with the different deadline coefficients κ and random distributions

Normal distribution

	$S_1 = 100$			$S_1 = 500$			$S_1 = 1000$		
κ	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1
0.2	**99.16**	99.14	99.14	**99.60**	**99.60**	**99.60**	**99.60**	99.59	99.59
0.4	**99.54**	99.53	99.53	**99.90**	99.89	99.89	**99.90**	99.89	99.89
0.6	**99.73**	99.72	99.72	**99.98**	**99.98**	**99.98**	**99.98**	**99.98**	**99.98**
0.8	**99.81**	99.80	99.80	**100**	**100**	**100**	**100**	**100**	**100**
1.0	**99.87**	99.87	99.87	**100**	**100**	**100**	**100**	**100**	**100**
1.2	**99.91**	99.90	99.90	**100**	**100**	**100**	**100**	**100**	**100**

Gamma distribution

	$S_1 = 100$			$S_1 = 500$			$S_1 = 1000$		
κ	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1
0.2	**99.77**	99.71	99.71	**99.87**	99.61	99.61	**99.87**	99.83	99.83
0.4	**99.82**	99.78	99.78	**99.75**	99.71	99.71	**99.92**	99.71	99.71
0.6	**99.90**	99.85	99.85	**99.84**	99.80	99.80	**99.84**	99.80	99.80
0.8	**99.94**	99.90	99.90	**99.90**	99.86	99.86	**99.90**	99.86	99.86
1.0	99.86	**99.92**	**99.92**	**99.96**	99.92	99.92	**99.96**	99.92	99.92
1.2	**100**	99.94	99.94	**99.98**	99.94	99.94	**99.98**	99.94	99.94

Lognormal distribution

	$S_1 = 100$			$S_1 = 500$			$S_1 = 1000$		
κ	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1
0.2	**95.98**	95.89	95.89	**96.48**	95.66	95.66	**97.84**	96.99	96.99
0.4	**96.43**	96.21	96.22	**97.01**	96.32	96.32	**98.18**	97.41	97.41
0.6	**96.88**	96.65	96.67	**97.50**	96.80	96.80	**98.51**	97.72	97.72
0.8	**97.32**	97.06	97.07	**97.74**	97.15	97.15	**98.05**	97.99	97.99
1.0	96.82	97.30	**97.31**	97.37	**97.48**	**97.48**	98.19	**98.24**	**98.24**
1.2	**97.28**	96.47	96.48	**97.63**	97.28	97.28	**98.38**	97.91	97.91

* S_1 is the size of training set on each arc. Bold values are the best results in each column.

Table 7.2 – CPU time (in seconds) of enumeration method, Card-DCA, ℓ_1 and re-ℓ_1 on 100 runs with different random distributions in case $\kappa = 1.2$

Normal distribution

	$S_1 = 100$			$S_1 = 500$			$S_1 = 1000$		
enumeration	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1
1.81	**0.01**	0.06	0.07	**0.03**	0.35	0.45	**0.10**	1.03	1.20

Gamma distribution

	$S_1 = 100$			$S_1 = 500$			$S_1 = 1000$		
enumeration	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1
2.11	**0.01**	0.10	0.12	**0.04**	0.49	0.43	**0.05**	0.81	0.93

Lognormal distribution

	$S_1 = 100$			$S_1 = 500$			$S_1 = 1000$		
enumeration	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1	Card-DCA	ℓ_1	re-ℓ_1
2.16	**0.04**	0.22	0.37	**0.14**	0.57	1.21	**0.72**	2.14	5.21

Comments on numerical results (Tables 7.1, 7.2): we observe from experiments that with suitable penalty parameters, our Card-DCA algorithm always furnishes the zero-one solutions, which is an advantage when solving the continuous problems. Moreover, in terms of the accuracy, our Card-DCA algorithm is almost always better than the existing algorithms with the different deadline coefficients and different random distributions. The quality of our solution is very high with the accuracy of above 95% and as for normal/gamma distribution, even above 99%, that is to say, our algorithm usually finds the optimal path as the enumeration method. Moreover, in most cases of our Card-DCA, the larger the training set is, the more the accuracy is improved, however for normal/gamma distribution, the accuracy in case $S_1 = 500$ is sufficiently high. Regarding CPU time, Card-DCA is faster than the existing algorithms ℓ_1, re-ℓ_1, respectively, up to 16.2, 18.6 times, which can be explained by the fact that our Card-DCA algorithm only requires solving successive linear programs, while the existing algorithms solve MILPs. In addition, with the good obtained accuracy, Card-DCA runs much faster than the enumeration method up to 211 times.

7.3.2 Experiment 2: Comparison between DCA approaches

In this experiment, we make a comparison between two DCA approaches, namely RL-DCA and Card-DCA, on an artificial road network with discrete deadlines which is described in [21]. In particular, the road network is a grid with $\overline{n} \times \overline{n}$ intersections and thus, the number of intersections (or nodes) is $n = \overline{n}^2$ and the number of links (or arcs) is $m = 2\overline{n}(\overline{n} - 1)$. At each intersection, there are four possible travel directions: east, west, south and north (thus, $N_{\mathcal{A}} = 4$).

Data set: For each link, we randomly generate S travel time instances with mean μ and standard deviation $\sigma = 0.3\mu$ where the value of μ is randomly generated with mean of 15 and standard deviation of 3. The origin o and the destination d are chosen from the set of intersections $\mathcal{V}$. The deadline T is defined by $T = \beta T_e$ where β is a deadline parameter ($\beta > 1$), T_e is the least expected travel time from o to d with the shortest path of length l_e. All values of travel time, deadline are rounded to the nearest integer. In batch mode, the samples are collected from the origin o along any path of length l_e based on all S travel time data. Thus, the number of batch samples is $N = S.l_e$.

Set up experiments: The parameters $\overline{n}$, S, β are set as follows: $\overline{n} \in \{20, 30, 40\}$, $S \in \{200, 500\}$, $\beta \in \{1.05, 1.10, 1.15\}$. The number of ($o$,$d$) pairs selected is 100. The experiment was implemented in MATLAB R2016a on a PC Intel(R) Xeon(R) CPU E5-2630 v2, @ 2.6GHz of 32GB RAM. The software CPLEX 12.6 was used for solving linear programs. The default tolerance is $\varepsilon = 10^{-4}$. The parameters of Card-DCA are set the same as the previous experiment. Let us define the basic function vector ϕ as follows: for any state $s = (v, \tau)$ and any action a, $i = 1, \ldots, n$, $j = -1, \ldots, T$, $k = 1, \ldots, N_{\mathcal{A}}$, we have the basic function $\phi_{i+(j+1)n+n(k-1)(N_\Gamma+2)}(s, a) = 1$ if the action a, the node v coincide respectively $a_k \in \mathcal{A}$, i in $\mathcal{V}$ and the time-to-deadline τ satisfies that $\tau < 0$ when $j = -1$, $\tau = j$ when $j = 0, \ldots, T-1$, $\tau \geq T$ when $j = T$, and 0 otherwise. Thus, the number of basic functions is $l = nN_{\mathcal{A}}(T + 2)$. The starting point for RL-DCA is a zero vector in $\mathbb{R}^d$.

Comparison criteria of algorithms: In our experiment, for any (o,d) pair, we cannot guarantee to find the path which has the maximum number of times of not being late by enumerating all the possible paths during a reasonable time (e.g. more than 3600 seconds). Thus, here we are interested in the following aspects: accuracy (in %) and CPU time (in seconds). For each (o,d) pair and each value of β, we compute the accuracy of each algorithm by the probability of arriving on time of the path obtained by the algorithm on the data set.

Descriptions of tables of results: The average accuracy and average CPU time obtained by two algorithms RL-DCA, Card-DCA on 100 (o,d) pairs with the different grids $\overline{n} \times \overline{n}$, the different number of samples on each link S, the different deadline parameters β are reported in Tables 7.3, 7.4. Bold values in these tables represent the best results.

Comments on numerical results (Tables 7.3, 7.4): for different grids $\overline{n} \times \overline{n}$ and number of samples S, the probability of arriving on time of RL-DCA is always larger than that of Card-DCA in all cases of deadline parameters β – the gain of RL-DCA versus Card-DCA varies from 0.08% to 5.85%. The CPU time of both algorithms is small in all data sets: less than 12 seconds – in particular, the Card-DCA is faster than RL-DCA when $\overline{n} = 20, 30$ and moderately comparable when $\overline{n} = 40$. In our experiments, we can summarize that within an acceptable time, RL-DCA gives the better accuracy than Card-DCA.

Table 7.3 – Accuracy (%) of RL-DCA and Card-DCA on 100 (o,d) pairs with the different grids $\overline{n} \times \overline{n}$, the different number of samples on each link S, the different deadline parameters β. Bold values are the best results.

	$\overline{n} = 20$, $S = 200$		$\overline{n} = 30$, $S = 200$		$\overline{n} = 40$, $S = 200$	
β	RL-DCA	Card-DCA	RL-DCA	Card-DCA	RL-DCA	Card-DCA
1.05	**75.23**	74.78	**84.29**	83.34	**83.19**	77.34
1.10	**88.85**	88.59	**97.14**	93.60	**97.22**	97.14
1.15	**95.46**	92.70	**99.50**	99.0	**99.64**	99.243

	$\overline{n} = 20$, $S = 500$		$\overline{n} = 30$, $S = 500$		$\overline{n} = 40$, $S = 500$	
β	RL-DCA	Card-DCA	RL-DCA	Card-DCA	RL-DCA	Card-DCA
1.05	**80.69**	77.43	**78.88**	78.03	**80.48**	76.38
1.10	**95.66**	93.05	**95.65**	91.82	**96.45**	94.04
1.15	**99.20**	98.11	**99.32**	98.13	**99.38**	98.65

Table 7.4 – CPU time (in seconds) of RL-DCA and Card-DCA on 100 (o,d) pairs with the different grids $\overline{n} \times \overline{n}$, the different number of samples on each link S, the different deadline parameters β. Bold values are the best results.

	$\overline{n} = 20$, $S = 200$		$\overline{n} = 30$, $S = 200$		$\overline{n} = 40$, $S = 200$	
β	RL-DCA	Card-DCA	RL-DCA	Card-DCA	RL-DCA	Card-DCA
1.05	1.35	**0.17**	3.53	**1.67**	**9.96**	10.9
1.10	1.33	**0.18**	3.68	**1.25**	10.1	**9.36**
1.15	1.37	**0.16**	3.74	**0.83**	10.3	**6.23**

	$\overline{n} = 20$, $S = 500$		$\overline{n} = 30$, $S = 500$		$\overline{n} = 40$, $S = 500$	
β	RL-DCA	Card-DCA	RL-DCA	Card-DCA	RL-DCA	Card-DCA
1.05	3.91	**0.82**	**5.17**	5.28	**11.0**	11.7
1.10	4.05	**0.69**	5.29	**4.28**	11.3	**9.16**
1.15	4.21	**0.50**	5.45	**2.97**	11.7	**6.72**

7.4 Conclusions

We have studied nonconvex, nonsmooth programming approaches based on DC programming and DCA for solving a class of SSP problems in vehicle routing for a single independent vehicle. In fact, the considered SSP problems can be reformulated as a cardinality optimization problem with zero-one variables and an RL problem. As for

the cardinality problem, we have used a DC approximation approach for the cardinality term. Exploiting an exact penalty technique in DC programming, we have dealt with the zero-one variables and obtained an exact reformulation of the corresponding approximate problem. Employing DCA for the resulting reformulation problem, we have developed a fast, efficient DCA scheme for the considered cardinality problem. Concerning the RL problem, we have explored the DCA based RL techniques in batch mode for solving these SSP problems. The numerical results on artificial road networks show that our approach for the cardinality problem is efficient in comparison with the existing algorithms in terms of the quality and the rapidity whereas the RL techniques can obtain the effectiveness to improve the accuracy.

Chapter 8

Conclusions

In the dissertation, we have developed the machine learning techniques in both theoretical and computational aspects. The backbones of our approaches are DC (Difference of Convex functions) and DCA (DC Algorithm), and their online version, which are best known as powerful nonsmooth, nonconvex optimization tools. Over the last three decades, DCA has been thoroughly studied and has enjoyed a lot of great success in a variety of domains in applied science. Thanks to DCA, we have recently developed its online version in the dissertation. This new version allows to encounter most of the large-scale optimization problems nowadays.

In the first part of the dissertation, we have intensively developed online learning (OL) techniques for a class of online problems where the loss function at each step is nonconvex and/or nonsmooth. By the DC approach, we have proposed a specific online version of DCA, named ODCA, where each subproblem is solved by approximating by one iteration of classical subgradient method. We have thoroughly studied the analysis of ODCA in the terms of regret – ODCA enjoys the sublinear/logarithmic regret. As an application, we have considered online binary linear classification. In particular, we have designed three corresponding ODCA based algorithms, all of which enjoy the logarithmic regret. Through the numerical experiments on various benchmark classification datasets, the effectiveness of our algorithms in terms of the quality and the rapidity has been demonstrated by comparing with the five state-of-the-art online classification algorithms.

Continuing the previous works, we have developed another class of OL techniques, namely prediction with expert advice, where the prediction at each step is made based on the basis of experts' predictions. We have exploited different DC approximation functions, which results in two particular Online DCA based schemes. Each convex subproblem in the schemes is solved based on two variants of subgradient method. Several analyses of both schemes in terms of regret have been studied thoroughly. The performance of our approach in efficiency, rapidity and scalability respects has been verified when compared with three existing standard algorithms on a variety of benchmark datasets.

Next, we have been interested in reinforcement learning (RL) techniques in both online mode and batch mode. As for batch RL, we have intensively investigated DC programming and DCA for the problem of finding the zero of the empirical optimal Bellman residual via linear approximation. We have considered four different DC optimization formulations for which four corresponding DCA schemes have been developed. To design efficient DCA algorithms, we have carefully addressed the three key issues of DCA by exploiting the special structure of the considered problems. It points out that, our proposed algorithm are more efficient than the existing DCA based algorithms on both quality and rapidity. Comparing with the two standard approaches for Batch RL, our algorithm realizes the best trade-off between the quality and the rapidity in several numerical experiments. Concerning RL in online mode, we have proposed a particular online DCA based algorithm which enjoys the online stability property. Moreover, we have suggested efficient alternating versions of the proposed online algorithm. Some experiment results on the classic mountain car and pole balancing problems have shown the efficiency of our approach.

As an application, we have studied DCA approaches for solving a class of stochastic shortest path (SSP) problems in vehicle routing. These SSP problems can be expressed as a cardinality optimization problem with binary variables as well as an RL problem. Regarding the cardinality problem, we have employed a DC approximation approach and an exact penalty technique in DC programming to develop a fast, efficient DCA scheme for the resulting problem. As for the RL problem, we have explored the DCA based RL techniques in batch mode for solving these problems. The numerical results on artificial road networks show that our DCA approaches give the best accuracy.

In our ongoing works, it is necessary to address some following issues.

Concerning OL, it is known that OL plays a very important role among machine learning techniques for solving the problems which are encountered more and more in various day-to-day applications. In this context, it is essential to create new, efficient tools for the development of the novel OL techniques. Our research on the online version of DC programming and DCA has just started but from the encouraging results, it promises to be an effective tool. In the future, we are making progress in further development of Online DC programming and Online DCA in theoretical and algorithmic respects. In particular, we plan to explore the more efficient DC approximation functions, exploit the fast and scalable solution methods for solving convex subproblems in the Online DCA scheme, study the thorough analysis of the corresponding scheme. We continue investigating this tool to solve a variety of problems in OL.

Regarding RL, our study on RL in batch/online mode suggests that DCA is an efficient approach. That motivates us to develop DC programming and DCA for applications of RL in several areas. Moreover, we intend to extend our results to develop more efficient RL techniques, for example, RL with eligibility traces.

As for SSP problems, we plan to explore the property of incident matrix in the cardinality formulation to guarantee the binary property of our obtained solutions. In addition, the first study on the way to treat the cardinality optimization problem with

binary variables based on DC programming and DCA opens up promising directions: not only how to develop efficient DCA schemes, but also the consistency between the approximate problem and the original problem should be investigated. On another hand, we are able to investigate these SSP problems based on the nonconvex exact reformulation approach for the cardinality term using the exact penalty techniques. In regard to RL approach for these SSP problems, it will be interesting to know whether our RL techniques can deal with this cardinality optimization problem or not. In the scope of the dissertation, we have considered linear approximation with simple basic function vectors, so the effect of basic functions on the performance of the proposed algorithms as well as how to develop DCA for RL with different approximation functions (e.g. neural network) should also be addressed in the future.

Bibliography

[1] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, ICML, New York, NY, USA. ACM.

[2] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[3] Antos, A., Szepesvári, C., and Munos, R. (2008). Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129.

[4] Azoury, K. and Warmuth, M. (2001). Relative loss bounds for on-line density estimation with the exponential family of distributions. *Machine Learning*, 43(3):211–246.

[5] Baird, L. C. I. (1995). Residual algorithms: Reinforcement learning with function approximation. In Prieditis, A. and Russell, S., editors, *Machine Learning Proceedings 1995*, pages 30–37. Morgan Kaufmann, San Francisco (CA).

[6] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846.

[7] Bellman, R. (1957). A Markovian decision process. *Indiana Univ. Math. J.*, 6(4):679–684.

[8] Bellman, R. and Dreyfus, S. (1959). Functional approximation and dynamic programming. *Math. Tables and other Aids Comp*, 13:247–251.

[9] Bertsekas, D. P., editor (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[10] Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific, 1st edition.

[11] Bertsekas, D. P. and Shreve, S. E. (1978). *Stochastic Optimal Control: The Discrete Time Case*. Mathematics in Science and Engineering 139. Academic Press, 1 edition.

[12] Bertsekas, D. P. and Tsitsiklis, J. N. (1991). An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595.

[13] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, 1st edition.

[14] Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M. (2009). Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482.

[15] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA.

[16] Bradley, P. S. and Mangasarian, O. L. (1998). Feature Selection via Concave Minimization and Support Vector Machines. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 82–90, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[17] Buşoniu, L., Babuska, R., Schutter, B. D., and Ernst, D. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

[18] Cao, Z. (2017). *Maximizing the Probability of Arriving on Time : A Stochastic Shortest Path Problem*. PhD thesis, Interdisciplinary Graduate School (IGS).

[19] Cao, Z., Guo, H., Zhang, J., Niyato, D., and Fastenrath, U. (2016a). Finding The Shortest Path in Stochastic Vehicle Routing: A Cardinality Minimization Approach. *IEEE Transactions on Intelligent Transportation Systems*, 17(6):1688–1702.

[20] Cao, Z., Guo, H., Zhang, J., Niyato, D., and Fastenrath, U. (2016b). Improving The Efficiency of Stochastic Vehicle Routing: A Partial Lagrange Multiplier Method. *IEEE Transactions on Vehicular Technology*, 65(6):3993–4005.

[21] Cao, Z., Guo, H., Zhang, J., Oliehoek, F., and Fastenrath, U. (2017). Maximizing the Probability of Arriving on Time: A Practical Q-Learning Method. In *31th AAAI Conference on Artificial Intelligence (AAAI)*, pages 4481–4487, San Francisco, California, USA.

[22] Cesa-Bianchi, N. (1999). Analysis of Two Gradient-Based Algorithms for On-Line Regression. *Journal of Computer and System Sciences*, 59(3):392–411.

[23] Cesa-Bianchi, N., Freund, Y., Haussler, D., Helmbold, D. P., Schapire, R. E., and Warmuth, M. K. (1997). How to use expert advice. *J. ACM*, 44(3):427–485.

[24] Cesa-Bianchi, N. and Lugosi, G. (2006). *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA.

[25] Chambolle, A., Devore, R. A., Lee, N. Y., and Lucier, B. J. (1998). Nonlinear wavelet image processing: Variational problems, compression, and noise removal through wavelet shrinkage. *IEEE Trans Image Process*, 7:319–335.

[26] Cheng, J., Kosuch, S., and Lisser, A. (2012). Stochastic Shortest Path Problem with Uncertain Delays. In *Proceedings of the 1st International Conference on Operations Research and Enterprise Systems*, pages 256–264.

[27] Collobert, R., Sinz, F., Weston, J., and Bottou, L. (2006a). Large scale transductive SVMs. *Journal of Machine Learning Research*, 7:1687–1712.

[28] Collobert, R., Sinz, F., Weston, J., and Bottou, L. (2006b). Trading convexity for scalability. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 201–208, New York, NY, USA. ACM.

[29] Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

[30] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39:1–38.

[31] DeSantis, A., Markowsky, G., and Wegman, M. N. (1988). Learning probabilistic prediction functions. In *Proceedings of the 1st Annual Workshop on Computational Learning Theory*, COLT'88, pages 312–328, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[32] Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556.

[33] Ertekin, S., Bottou, L., and Giles, C. L. (2011). Nonconvex Online Support Vector Machines. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(2):368–381.

[34] Fan, Y. Y., Kalaba, R. E., and Moore, J. E. (2005). Arriving on Time. *Journal of Optimization Theory and Applications*, 127(3):497–513.

[35] Frank, H. (1969). Shortest Paths in Probabilistic Graphs. *Oper. Res.*, 17(4):583–599.

[36] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

[37] Gasso, G., Pappaioannou, A., Spivak, M., and Bottou, L. (2011). Batch and online learning algorithms for nonconvex neyman-pearson classification. *ACM Trans. Intell. Syst. Technol.*, 2(3):28:1–28:19.

[38] Geist, M. and Pietquin, O. (2013). Algorithmic Survey of Parametric Value Function Approximation. *IEEE Transactions on Neural Networks and Learning Systems*, 24(6):845–867.

[39] Gentile, C. (2002). A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242.

[40] Gentile, C. (2003). The robustness of the p-norm algorithms. *Machine Learning*, 53(3):265–299.

[41] Geramifard, A., Walsh, T. J., Tellex, S., Chowdhary, G., Roy, N., and How, J. P. (2013). A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends® in Machine Learning*, 6(4):375–451.

[42] Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *INFORMS J. on Computing*, 21(2):178–192.

[43] Grötschel, M., Krumke, S. O., and Rambau, J., editors (2001). *Online Optimization of Large Scale Systems*. Springer.

[44] Haussler, D., Kivinen, J., and Warmuth, M. (1995). Tight worst-case loss bounds for predicting with expert advice. In Vitányi, P., editor, *Computational Learning Theory*, volume 904 of *Lecture Notes in Computer Science*, pages 69–83. Springer Berlin Heidelberg.

[45] Hazan, E. (2016). Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325.

[46] Ho, V. T., Le Thi, H. A., and Bui, D. C. (2016). Online DC optimization for online binary linear classification. In Nguyen, T. N., Trawiński, B., Fujita, H., and Hong, T.-P., editors, *Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Proceedings, Part II*, pages 661–670, Berlin, Heidelberg. Springer Berlin Heidelberg.

[47] Hoi, S. C. H., Wang, J., and Zhao, P. (2014). Libol: A library for online learning algorithms. *Journal of Machine Learning Research*, 15(1):495–499.

[48] Jaillet, P. and Wagner, M. R. (2014). *Online Optimization - An Introduction*, chapter 6, pages 142–152. INFORMS, Maryland, USA.

[49] Kalai, A. and Vempala, S. (2005). Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307.

[50] Kivinen, J. and Warmuth, M. K. (1997). Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63.

[51] Kivinen, J. and Warmuth, M. K. (2001). Relative loss bounds for multidimensional regression problems. *Machine Learning*, 45(3):301–329.

[52] Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149.

[53] Lange, S., Gabel, T., and Riedmiller, M. (2012). Batch Reinforcement Learning. In Wiering, M. and van Otterlo, M., editors, *Reinforcement Learning*, volume 12, chapter 2, pages 45–73. Springer Berlin Heidelberg, Hillsdale, NJ.

[54] Le, H. M., Le Thi, H. A., and Nguyen, M. C. (2015). Sparse semi-supervised support vector machines by DC programming and DCA. *Neurocomputing*, 153(Supplement C):62 – 76.

[55] Le Thi, H. A. (1997). Contribution à l'optimisation non convexe et l'optimisation globale: Théorie, algorithmes et applications. *Habilitation à Diriger des Recherches*, Université de Rouen.

[56] Le Thi, H. A. (2005). *DC Programming and DCA*. `http://www.lita.univ-lorraine.fr/~lethi/`.

[57] Le Thi, H. A., Le, H. M., Nguyen, V. V., and Pham Dinh, T. (2008). A DC Programming Approach for Feature Selection in Support Vector Machines Learning. *Advances in Data Analysis and Classification*, 2(3):259–278.

[58] Le Thi, H. A., Le, H. M., and Pham Dinh, T. (2014). New and efficient DCA based algorithms for minimum sum-of-squares clustering. *Pattern Recognition*, 47(1):388–401.

[59] Le Thi, H. A., Le, H. M., and Pham Dinh, T. (2015a). Feature selection in machine learning: an exact penalty approach using a difference of convex function algorithm. *Machine Learning*, 101(1–3):163–186.

[60] Le Thi, H. A., Le, H. M., Pham Dinh, T., and Huynh, V. N. (2013). Binary classification via spherical separator by DC programming and DCA. *Journal of Global Optimization*, 56(4):1393–1407.

[61] Le Thi, H. A. and Moeini, M. (2014). Long-Short Portfolio Optimization Under Cardinality Constraints by Difference of Convex Functions Algorithm. *Journal of Optimization Theory and Applications*, 161(1):199–224.

[62] Le Thi, H. A. and Nguyen, M. C. (2014). Self-organizing maps by difference of convex functions optimization. *Data Mining and Knowledge Discovery*, 28(5-6):1336–1365.

[63] Le Thi, H. A. and Nguyen, M. C. (2017). DCA based algorithms for feature selection in multi-class support vector machine. *Annals of Operations Research*, 249(1):273–300.

[64] Le Thi, H. A., Nguyen, M. C., and Pham Dinh, T. (2014a). A DC programming approach for finding communities in networks. *Neural Computation*, 26(12):2827–2854.

[65] Le Thi, H. A. and Pham Dinh, T. (1997). Solving a Class of Linearly Constrained Indefinite Quadratic Problems by D.C. Algorithms. *Journal of Global Optimization*, 11(3):253–285.

[66] Le Thi, H. A. and Pham Dinh, T. (2005). The DC (Difference of Convex Functions) Programming and DCA Revisited with DC Models of Real World Nonconvex Optimization Problems. *Annals of Operation Research*, 133(1-4):23–46.

[67] Le Thi, H. A. and Pham Dinh, T. (2016). DC programming and DCA: thirty years of developments. *Mathematical Programming Series B*. Submitted.

[68] Le Thi, H. A., Pham Dinh, T., Le, H. M., and Vo, X. T. (2015b). DC approximation approaches for sparse optimization. *European Journal of Operational Research*, 244(1):26–46.

[69] Le Thi, H. A., Pham Dinh, T., and Le Dung, M. (1999). Exact Penalty in DC Programming. *Vietnam Journal of Mathematics*, 27(2):169–178.

[70] Le Thi, H. A., Pham Dinh, T., and Ngai, H. V. (2012). Exact penalty and error bounds in DC programming. *Journal of Global Optimization*, 52(3):509–535.

[71] Le Thi, H. A., Pham Dinh, T., and Thiao, M. (2016a). Efficient approaches for ℓ_2-ℓ_0 regularization and applications to feature selection in svm. *Applied Intelligence*, 45(2):549–565.

[72] Le Thi, H. A. and Phan, D. N. (2016). DC Programming and DCA for Sparse Optimal Scoring Problem. *Neurocomputing*, 186:170 –181.

[73] Le Thi, H. A. and Phan, D. N. (2017). DC programming and DCA for sparse fisher linear discriminant analysis. *Neural Computing and Applications*, 28(9):2809–2822.

[74] Le Thi, H. A., Vo, X. T., and Pham Dinh, T. (2014b). Feature selection for linear SVMs under uncertain data: Robust optimization based on difference of convex functions algorithms. *Neural Networks*, 59:36–50.

[75] Le Thi, H. A., Vo, X. T., and Pham Dinh, T. (2016b). Efficient nonnegative matrix factorization by DC programming and DCA. *Neural Computation*, 28(6):1163–1216.

[76] Li, Y. and Long, P. (2002). The relaxed online maximum margin algorithm. *Machine Learning*, 46(1-3):361–387.

[77] Lim, Sejoon and Balakrishnan, Hari and Gifford, David and Madden, Samuel and Rus, Daniela (2011). Stochastic Motion Planning and Applications to Traffic. *The International Journal of Robotics Research*, 30(6):699–712.

[78] Littlestone, N. and Warmuth, M. (1994). The weighted majority algorithm. *Information and Computation*, 108(2):212–261.

[79] Liu, Y. and Shen, X. (2006). Multicategory ψ-learning. *Journal of the American Statistical Association*, 101:500–509.

[80] Luce, R. D. (1959). *Individual Choice Behavior: A theoretical analysis*. Wiley.

[81] Maillard, O. A., Munos, R., Lazaric, A., and Ghavamzadeh, M. (2010). Finite sample analysis of Bellman residual minimization. In Sugiyama, M. and Yang, Q., editors, *Asian Conference on Machine Learpning. JMLR: Workshop and Conference Proceedings*, volume 13, pages 309–324.

[82] Martin, J. J. (1965). Distribution of The Time Through A Directed, Acyclic Network. *Oper. Res.*, 13(1):46–66.

[83] Mason, L., Baxter, J., Bartlett, P., and Frean, M. (1999). Boosting algorithms as gradient descent. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 512–518, Cambridge, MA, USA. MIT Press.

[84] Michie, D. and Chambers, R. A. (1968). BOXES: An experiment in adaptive control. In Dale, E. and Michie, D., editors, *Machine Intelligence*. Oliver and Boyd, Edinburgh, UK.

[85] Miller-Hooks, E. D. and Mahmassani, H. S. (2000). Least Expected Time Paths in Stochastic, Time-Varying Transportation Networks. *Transportation Science*, 34(2):198–215.

[86] Munos, R. (2007). Performance Bounds in L_p norm for Approximate Value Iteration. *SIAM Journal on Control and Optimization*.

[87] Neu, G., Gyorgy, A., and Szepesvari, C. (2012). The adversarial stochastic shortest path problem with unknown transition probabilities. In Lawrence, N. D. and Girolami, M., editors, *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 805–813, La Palma, Canary Islands. PMLR.

[88] Nie, Y. M. and Wu, X. (2009). Shortest Path Problem Considering On-Time Arrival Probability. *Transportation Research Part B: Methodological*, 43(6):597–613.

[89] Nikolova, E. (2010). Approximation Algorithms for Reliable Stochastic Combinatorial Optimization. In Serna, M., Shaltiel, R., Jansen, K., and Rolim, J., editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques: 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, pages 338–351. Springer Berlin Heidelberg, Berlin, Heidelberg.

[90] Nikolova, E., Kelner, J. A., Brand, M., and Mitzenmacher, M. (2006). Stochastic Shortest Paths via Quasi-convex Maximization. In Azar, Y. and Erlebach, T., editors, *Algorithms – ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings*, pages 552–563. Springer Berlin Heidelberg, Berlin, Heidelberg.

[91] Novikoff, A. B. (1963). On convergence proofs for perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622.

[92] Pashenkova, E., Rish, I., and Dechter, R. (1996). Value iteration and policy iteration algorithms for markov decision problem.

[93] Pham Dinh, T. and El Bernoussi, S. (1986). Algorithms for Solving a Class of Nonconvex Optimization Problems. Methods of Subgradients. *North-Holland Mathematics Studies*, 129:249 – 271. Fermat Days 85: Mathematics for Optimization.

[94] Pham Dinh, T., Le, H. M., Le Thi, H. A., and Lauer, F. (2014). A difference of convex functions algorithm for switched linear regression. *Automatic Control, IEEE Transactions on*, 59(8):2277–2282.

[95] Pham Dinh, T. and Le Thi, H. A. (1997). Convex analysis approach to DC programming: theory, algorithms and applications. *Acta Mathematica Vietnamica*, 22(1):289–355.

[96] Pham Dinh, T. and Le Thi, H. A. (1998). DC optimization algorithms for solving the trust region subproblem. *SIAM Journal of Optimization*, 8(2):476–505.

[97] Pham Dinh, T. and Le Thi, H. A. (2014). Recent Advances in DC Programming and DCA. In Nguyen, N. T. and Le Thi, H. A., editors, *Transactions on Computational Intelligence XIII*, volume 8342, pages 1–37. Springer Berlin Heidelberg.

[98] Piot, B., Geist, M., and Pietquin, O. (2014). Difference of Convex Functions Programming for Reinforcement Learning. In *Advances in Neural Information Processing Systems (NIPS 2014)*.

[99] Puterman, M. L., editor (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA.

[100] Randour, M., Raskin, J.-F., and Sankur, O. (2015). Variations on the stochastic shortest path problem. In D'Souza, D., Lal, A., and Larsen, K. G., editors, *Verification, Model Checking, and Abstract Interpretation: 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, pages 1–18, Berlin, Heidelberg. Springer Berlin Heidelberg.

[101] Rockafellar, R. T. (1970). *Convex analysis*. Princeton Mathematical Series. Princeton University Press, Princeton, N. J.

[102] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

[103] Ross, S. and Bagnell, J. A. (2011). Stability conditions for online learnability. *arXiv:1108.3154*, pages 1–16.

[104] Saha, A., Jain, P., and Tewari, A. (2012). The interplay between stability and regret in online learning. *arXiv:1211.6158*, pages 1–19.

[105] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229.

[106] Scherrer, B. (2010). Should one compute the Temporal Difference fix point or minimize the Bellman residual? The unified oblique projection view. In *27th International Conference on Machine Learning - ICML 2010*, Haïfa, Israel.

[107] Schweitzer, P. and Seidmann, A. (1985). Generalized polynomial approximations in markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110(2):568–582.

[108] Shalev-Shwartz, S. (2007). *Online Learning: Theory, Algorithms, and Applications*. PhD thesis, The Hebrew University of Jerusalem.

[109] Shalev-Shwartz, S. (2012). Online learning and online convex optimization. *Found. Trends Mach. Learn.*, 4(2):107–194.

[110] Shalev-Shwartz, S. and Singer, Y. (2007). A primal-dual perspective of online learning algorithms. *Machine Learning*, 69(2-3):115–142.

[111] Shor, N. Z. (1985). *Minimization Methods for Non-Differentiable Functions*. Springer Series in Computational Mathematics 3. Springer-Verlag Berlin Heidelberg, 1 edition.

[112] Sigal, C. E., Pritsker, A. A. B., and Solberg, J. J. (1980). The Stochastic Shortest Route Problem. *Oper. Res.*, 28(5):1122–1129.

[113] Sigaud, O. and Buffet, O., editors (2010). *Markov Decision Processes in Artificial Intelligence*. Wiley-IEEE Press.

[114] Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308.

[115] Singh, S. P., Jaakkola, T., and Jordan, M. I. (1995). Reinforcement learning with soft state aggregation. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 361–368. MIT Press, San Mateo.

[116] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

[117] Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press.

[118] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

[119] Szepesvári, C. (2010). *Algorithms for Reinforcement Learning*. Morgan & Claypool.

[120] Szepesvári, C. and Smart, W. D. (2004). Interpolation-based Q-learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML'04, pages 791–798, New York, NY, USA. ACM.

[121] Van Der Malsburg, C. (1986). Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. In Palm, G. and Aertsen, A., editors, *Brain Theory*, pages 245–248. Springer Berlin Heidelberg.

[122] van Otterlo, M. and Wiering, M. (2012). Reinforcement Learning and Markov Decision Processes. In Wiering, M. and van Otterlo, M., editors, *Reinforcement Learning*, volume 12, chapter 1, pages 3–42. Springer Berlin Heidelberg, Hillsdale, NJ.

[123] Vapnik, V. N., editor (1998). *Statistical learning theory*. Wiley.

[124] Vovk, V. G. (1990). Aggregating strategies. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, COLT'90, pages 371–386, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[125] Wang, W. and Carreira-Perpiñán, M. Á. (2013). Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application.

[126] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK.

[127] Wiering, M. and van Otterlo, M., editors (2012). *Reinforcement Learning: State-of-the-Art*, volume 12 of *Adaptation, Learning, and Optimization*. Springer-Verlag Berlin Heidelberg, 1 edition.

[128] Williams, R. J. and Baird, L. C. I. (1993). Tight performance bounds on greedy policies based on imperfect value functions. Technical report, College of Computer Science, Northeastern University.

[129] Xu, X., Zuo, L., and Huang, Z. (2014). Reinforcement learning algorithms with function approximation: Recent advances and applications. *Information Sciences*, 261:1–31.

[130] Yin, P., Lou, Y., He, Q., and Xin, J. (2015). Minimization of ℓ_{1-2} for compressed sensing. *SIAM Journal on Scientific Computing*, 37(1).

[131] Yuille, A. L. and Rangarajan, A. (2003). The concave-convex procedure (CCCP). *Neural Comput*, 15:915–936.

[132] Zhang, L., Yang, T., Jin, R., and Zhou, Z.-H. (2015). Online bandit learning for a special class of non-convex losses. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, AAAI'15, pages 3158–3164. AAAI Press.

[133] Zhao, P., Hoi, S. C. H., and Zhuang, J. (2013). Active learning with expert advice. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*.

[134] Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. In Fawcett, T. and Mishra, N., editors, *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 928–936.