DC programming and DCA for some classes of problems in machine learning and data mining

Manh Cuong Nguyen

THÈSE

présentée par

NGUYEN Manh Cuong

en vue de l'obtention du grade de

DOCTEUR DE L'UNIVERSITÉ DE LORRAINE

(arrêté ministériel du 7 Août 2006)

Spécialité: **INFORMATIQUE**

LA PROGRAMMATION DC ET DCA POUR CERTAINES CLASSES DE PROBLÈMES EN APPRENTISSAGE ET FOUILLE DE DONÉES.

Soutenue le 19 mai 2014 devant le jury composé de

Rapporteur	**BENNANI Younès**	*Professeur, Université Paris 13*
Rapporteur	**RAKOTOMAMONJY Alain**	*Professeur, Université de Rouen*
Examinateur	**GUERMEUR Yann**	*Directeur de recherche, LORIA-Nancy*
Examinateur	**HEIN Matthias**	*Professeur, Université Saarland*
Examinateur	**PHAM DINH Tao**	*Professeur émérite, INSA-Rouen*
Directrice de thèse	**LE THI Hoai An**	*Professeur, Université de Lorraine*
Co-encadrant	**CONAN-GUEZ Brieuc**	*MCF, Université de Lorraine*

Thèse préparée à l'Université de Lorraine, Metz, France
au sein de laboratoire LITA

Remerciements

Cette thèse a été préparée au sein du Laboratoire d'Informatique Théorique et Appliquée (LITA) de l'Université de Lorraine - France, sous la co-direction du Madame le Professeur LE THI Hoai An, Directrice du laboratoire Laboratoire d'Informatique Théorique et Appliquée (LITA), Université de Lorraine et Maître de conférence CONAN-GUEZ Brieuc, Institut Universitaire de Technologie (IUT), Université de Lorraine, Metz.

J'adresse toute ma gratitude à Madame le Professeur LE THI Hoai An et Maître de conférence CONAN-GUEZ Brieuc qui ont accepté d'être mes directeurs de thèse, pour leur aide, leur patience, leur soutien permanent, leurs conseils précieux, leurs encouragements et leur disponibilité tout au long de ce travail.

Je tiens à remercier plus particulièrement Monsieur le Professeur Pham Dinh Tao, Professeur à L'INSA de Rouen pour ses conseils, son suivi de mes recherches. Je voudrais lui exprimer toutes mes reconnaissances pour sa sympathie et les discussions très intéressantes qu'il a menées pour me suggérer les voies de recherche.

Je souhaite exprimer mes vifs remerciements à Monsieur BENNANI Younès, Professeur à l'Université Paris 13 et à Monsieur RAKOTOMAMONJY Alain, Professeur à l'Université de Rouen, de m'avoir fait l'honneur d'accepter la charge de rapporteur de ma thèse, ainsi que d'avoir participé à juger mon travail.

Je tiens à remercier Monsieur GUERMEUR Yann, Directeur de Recherche, LORIA Nancy et Monsieur HEIN Matthias, Professeur à l'Université Saarland d'avoir participé à juger mon travail.

Je remercie particulièrement le Docteur LE Hoai Minh pour l'aide inestimable, sa collaboration et pour les discussions très intéressantes que nous avons pu avoir.

Je remercie tous mes collègues Français et Vietnamiens rencontrés à Metz ainsi qu'à Rouen pour les moments agréables lors de mon séjour en France. Je remercie particulièrement tous mes collègues à LITA, l'Université de Lorraine, pour les partages dans le travail et dans la vie.

Mes remerciements s'adressent également au Gouvernement Vietnamien qui a financé mes études pendant la thèse. Je ne devrais pas oublier remercier tout l'équipe du personnel de l'Université de l'Industrie de Hanoi pour son soutien.

Je devrais témoigner mon affectation, mes reconnaissances à ma famille pour les sacrifices qu'elle a faites pour me soutenir lors des moments difficiles.

Enfin à tous ceux qui m'ont soutenu de près ou de loin et à tous ceux qui m'ont incité même involontairement à faire mieux, veuillez trouver ici le témoignage de ma profonde gratitude.

Résumé

La classification (supervisée, non supervisée et semi-supervisée) est une thématique importante de la fouille de données. Dans cette thèse, nous nous concentrons sur le développement d'approches d'optimisation pour résoudre certains types des problèmes issus de la classification de données. Premièrement, nous avons examiné et développé des algorithmes pour résoudre deux problèmes classiques en apprentissage non supervisée : la maximisation du critère de modularité pour la détection de communautés dans des réseaux complexes et les cartes auto-organisatrices. Deuxièmement, pour l'apprentissage semi-supervisée, nous proposons des algorithmes efficaces pour le problème de sélection de variables en semi-supervisée Machines à vecteurs de support. Finalement, dans la dernière partie de la thèse, nous considérons le problème de sélection de variables en Machines à vecteurs de support multi-classes. Tous ces problèmes d'optimisation sont non convexe de très grande dimension en pratique.

Les méthodes que nous proposons sont basées sur les programmations DC (Difference of Convex functions) et DCA (DC Algorithms) étant reconnues comme des outils puissants d'optimisation. Les problèmes évoqués ont été reformulés comme des problèmes DC, afin de les résoudre par DCA. En outre, compte tenu de la structure des problèmes considérés, nous proposons différentes décompositions DC ainsi que différentes stratégies d'initialisation pour résoudre un même problème. Tous les algorithmes proposés ont été testés sur des jeux de données réelles en biologie, réseaux sociaux et sécurité informatique.

Abstract

Classification (supervised, unsupervised and semi-supervised) is one of important research topics of data mining which has many applications in various fields. In this thesis, we focus on developing optimization approaches for solving some classes of optimization problems in data classification. Firstly, for unsupervised learning, we considered and developed the algorithms for two well-known problems: the modularity maximization for community detection in complex networks and the data visualization problem with Self-Organizing Maps. Secondly, for semi-supervised learning, we investigated the effective algorithms to solve the feature selection problem in semi-supervised Support Vector Machine. Finally, for supervised learning, we are interested in the feature selection problem in multi-class Support Vector Machine. All of these problems are large-scale non-convex optimization problems.

Our methods are based on DC Programming and DCA which are well-known as powerful tools in optimization. The considered problems were reformulated as the DC programs and then the DCA was used to obtain the solution. Also, taking into account the structure of considered problems, we can provide appropriate DC decompositions and the relevant choice strategy of initial points for DCA in order to improve its efficiency. All these proposed algorithms have been tested on the real-world datasets including biology, social networks and computer security.

Curriculum Vitae

PERSONAL INFORMATIONS

First name	NGUYEN
Last name	Manh Cuong
Date and place of birth	18/3/1978 at Thai Binh, Vietnam
Gender	Male
Nationality	Vietnamese
Personal address	Résidence Universitaire du Saulcy, 57010 Metz, France
Professional address	LITA, University of Lorraine, Ile du Saulcy, 57045 Metz, France.
Phone number	06 73 71 59 89
Email	Cuongnm78@yahoo.com

EDUCATION

University of Lorraine

PhD. Student **2010-2014**

Subject of the thesis: La programmation DC et DCA pour certaines classes de problèmes en Apprentissage et fouille de données.

Hanoi University of Technology

MSc. Student **2001-2003**

Subject of the thesis: "Mixed Similarity Measure and Its Applications"
Grade: Good

Hanoi University of Science

BSc. Student **1996-2000**

Subject of the thesis: "The Graph Algorithms for the Problems of Traffic in the City"
Grade: Good

TEACHING EXPERIENCE

Hanoi University of Industry, Vietnam

- Lecturer **2003-2010**

LANGUAGES

- French
- English
- Vietnamese

Publications

Refereed international journal papers

[1]. Le Thi, H.A., Nguyen, M.C.: *Efficient Algorithms for Feature Selection in Multi-class Support Vector Machine.* Submitted.

[2]. Le Hoai, M., Nguyen, M.C., Le Thi, H.A.: *DCA Based algorithm for feature selection in Semi-Supervised Support Vector Machines.* Submitted.

[3]. Le Thi, H.A., Nguyen, M.C., Pham Dinh, T.: *A DC programming approach for finding Communities in networks.* Submitted version to the Neural Computation journal (NECO).

[4]. Le Thi, H.A., Nguyen, M.C.: *Self-Organizing Maps by Difference of Convex functions optimization.* Revised version to the Data Mining and Knowledge Discovery journal (DAMI).

[5]. Conan-Guez, B., Nguyen, M.C.: *Mod-Müllner: An Efficient Algorithm for Hierarchical Community Analysis in Large Networks.* Submitted version to the International Journal of Social Network Mining (IJSNM).

Refereed papers in books / Refereed international conference papers

[1]. Phan, D.N., Nguyen, M.C., Le Thi, H.A.: A DC programming approach for sparse linear discriminant analysis. In Advances in Intelligent Systems and Computing, Volume 282, pp. 65-74, Springer (2014).

[2]. Le Thi, H.A., Nguyen, M.C.: *Efficient Algorithms for Feature Selection in Multi-class Support Vector Machine.* In Advanced Computational Methods for Knowledge Engineering, Studies in Computational Intelligence, Volume 479, pp. 41-52, Springer (2013).

[3]. Le Hoai, M., Le Thi, H.A., Nguyen, M.C.: *DCA Based algorithm for feature selection in Semi-Supervised Support Vector Machines.* In Machine Learning and Data Mining in Pattern Recognition, Lecture Notes in Computer Science, Volume 7988, pp. 528-542, Springer (2013).

[4]. Le, A.V., Le Thi, H.A., Nguyen, M.C.,Zidna, A.: *Network Intrusion Detection based on Multi-Class Support Vector Machine.* In Computational Collective Intelligence. Technologies and Applications, Lecture Notes in Computer Science Volume 7653, pp. 536-543, Springer (2012).

[5]. Le Thi, H.A., Nguyen, M.C., Conan-Guez, B., Pham Dinh, T.: *A new method for Batch Learning Self-Organizing Maps based on DC Programming and DCA.* Proceedings, 2nd Stochastic Modeling Techniques and Data Analysis International Conference SMTDA, pp. 473–556, Greece (2012).

[6]. Conan-Guez, B., Le Thi, H.A., Nguyen, M.C., Pham Dinh, T.: *Détection de communautés: une approche par programmation DC.* Proceeding of SFC conference, pp. 13–16, Orléans, France (2011).

Communications in national/ international conferences

[1]. Conan-Guez, B., Nguyen, M.C.: *Une adaptation de l'algorithme de Müllner pour la détection de communautés dans des réseaux complexes.* Journées Big Data & Visualization, Fouille et Visualisation de Données Massives, Paris, France (2013).

[2]. Conan-Guez, B., Le Thi, H.A., Nguyen, M.C., Pham Dinh, T.: *Finding Communities in networks: a DC programming approach.* The "Optimization" conference, Lisbon-Portugal (2011).

Contents

Introduction générale

Cadre général et nos motivations.

L'un des défis pour les scientifiques à l'heure actuelle consiste en l'exploitation optimale des informations stockées/évoluées dans les très nombreuses ressources. Les méthodes de Machine Learning et Data Mining (MLDM) qui offrent aujourd'hui une technologie mature pour traiter les problèmes "classiques" doivent faire face à l'explosion des nouveaux besoins liée au développement du web, aux masses de données, aux nouveaux types et formats de données. Un premier enjeu consiste à dépasser le cadre clasique de MLDM pour s'attaquer à cette nouvelle gamme de problèmes et pour répondre à ces nouveaux besoins. Les nouveaux modèles et méthodes d'optimisation s'avèrent cruciaux pour cela. L'interaction entre l'optimisation et MLDM est l'un des plus importants développements dans la science informatique moderne.

Dans ce contexte nous avons choisi volontairement le développement d'une approche innovante de l'optimisation déterministe, à savoir la programmation DC (Difference of Convex functions) et DCA (DC Algorithms), pour la résolution de certaines classes de problèmes étant des challenges en MLDM.

On peut distinguer deux branches de l'optimisation déterministe: la programmation convexe et la programmation non convexe. Un programme convexe ou un problème d'optimisation convexe est celui de la minimisation d'une fonction (objectif) convexe sous des contraintes convexes. Lorsque la double convexité chez l'objectif et les contraintes n'est pas vérifiée, on est en face un problème d'optimisation non convexe. La double convexité d'un programme convexe permet d'établir des caractérisations (sous forme de conditions nécessaires et suffisantes) de solutions optimales et ainsi de construire des méthodes itératives convergeant vers des solutions optimales. Théoriquement on peut résoudre tout programme convexe, mais encore faut-il bien étudier la structure du programme convexe en question pour proposer des variantes performantes peu coûteuses et donc capables d'atteindre des dimensions réelles très importantes. L'absence de cette double convexité rend la résolution d'un programme non convexe difficile voire impossible à l'état actuel des choses. Contrairement à la programmation convexe, les solutions optimales locales et globales sont à distinguer dans un programme non convexe. D'autre part si l'on dispose des caractérisations d'optimalité locale utilisables, au moins pour la classe des programmes non convexes assez réguliers, qui permettent la construction des méthodes convergeant vers des solutions locales (algorithmes locaux) il n'y a par contre pas de caractérisations d'optimalité globale sur lesquelles sont basées les méthodes itératives convergeant vers des solutions globales (algorithmes globaux). L'analyse et l'optimisation convexes modernes se voient ainsi contrainte à une extension logique et naturelle à la non convexité et la non différentiabilité. Les méthodes numériques conventionnelles de l'optimisation convexe ne fournissent que des minima locaux bien souvent éloignés de l'optimum global.

L'optimisation non convexe connaît une explosion spectaculaire depuis des années 90 car dans les milieux industriels, on a commencé à remplacer les modèles convexes par des modèles non convexes plus complexes mais plus fiables qui présentent mieux la nature des problèmes étudiés. Durant ces dernières années, la recherche en optimisation non convexe a largement bénéficié des efforts des chercheurs et s'est enrichie de nouvelles approches. Il existe deux approches différentes mais complémentaires en programmation non convexe:

i) Approches globales combinatoires: qui sont basées sur les techniques combinatoires de la Recherche Opérationnelle. Elles consistent à localiser les solutions optimales à l'aide des méthodes d'approximation, des techniques de coupe, des méthodes de décomposition, de séparation et évaluation. Elles ont connu de très nombreux développements importants au cours de ces dernières années à travers les travaux de H. Tuy (reconnu comme le pionnier), R. Horst, P. Pardalos et N. V. Thoai, ([7],[8]) ... L'inconvénient majeur des méthodes globales est leur lourdeur (encombrement en places-mémoires) et leur coût trop important. Elles ne sont pas applicables aux problèmes d'optimisation non convexes réels qui sont souvent de très grande dimension.

ii) Approches locales et globales d'analyse convexe qui sont basées sur l'analyse et l'optimisation convexe. Ici la programmation DC et DCA jouent le rôle central car la plupart des problèmes d'optimisation non convexe sont formulés/reformulés sous la forme DC. Sur le plan algorithmique, l'essentiel repose sur les algorithmes de l'optimisation DC (DCA) introduits par Pham Dinh Tao en 1985 à l'état préliminaire et développés intensivement à travers de nombreux travaux communs de Le Thi Hoai An et Pham Dinh Tao depuis 1993 pour devenir maintenant classiques et de plus en plus utilisés par des chercheurs et praticiens de par le monde, dans différents domaines des sciences appliquées ([3]-[5],[12]).

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, semi-continues inférieurement et propres sur $\mathbb{R}^n$. 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. Or chaque fonction DC admet un nombre infini 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.

Les travaux dans cette thèse sont basés sur la programmation DC et DCA. Cette démarche est justifiée par multiple arguments ([17, 18]):

- La programmation DC et DCA fournissent un cadre très riche pour MLDM: MLDM 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 [11] témoigne de la vitalité, la puissance et la percée de cette approche dans la communauté de MLDM.

- 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 MLDM (voir par exemple la liste des références dans [11]).

Il est intéressant de noter qu'on retrouve une version de DCA dans toutes les méthodes standard largement utilisées par la communauté de MLDM dont méthodes EM (Expectation-Maximisation) [4],[2], SLA (Succesive Linear Approximation) [2] et CCCP (ConCave-Convex Procedure) [26].

Nos contributions.

Nous nous sommes intéressés aux quatre classes des problèmes de MLDM qui n'ont pas été traitées par la programmation DC et DCA dans la litérature. Elles se situent sur tous les trois branches d'apprentissage: non supervisée, supervisée et semi-supervisée. Nous décrivons brièvement ci-après les grandes lignes de nos contributions. Les détails et la position de nos travaux par rapport à l'état de l'art de chaque classes de problèmes seront présentés dans le chapitre concerné.

La première, en apprentissage non supervisée (qui n'utilise que des données non étiquettées), est *le clustering des sommets d'un graphe par maximisation de la modularité*. Ce travail est motivé par une application intéressante et importante: détection des communautés dans les réseaux complexes. Au cours des dernières années, l'étude des réseaux complexes a suscité des grands intérêts dans de nombreuses disciplines, notamment la physique, la biologie et les sciences sociales. Des exemples de ces réseaux incluent des graphiques Web, les réseaux sociaux, les réseaux biochimiques. Malgré que ces réseaux appartiennent à des domaines très différents, ils partagent certaines caractéristiques structurelles communes, parmi lesquelles la structure communautaire est très importante. Un réseau est en fait composé de sous-réseaux de connexions denses, avec les connexions moins denses entre eux. Ces sous- réseaux sont appelés des communautés ou des modules. La détection des communautés est une pratique importante car elle permet d'analyser les réseaux à l'échelle mésoscopique: l'identification des pages Web connexes dans WWW, la détection des communautés dans les réseaux sociaux, la décomposition des réseaux métaboliques en modules fonctionnels. Le critère le plus utilisé pour caractériser une structure des communautés dans un réseau est la modularité, une mesure quantitative proposé par Newman et Girvan [21, 22] en 2004. La détection des communautés est ainsi formulée comme le problème de maximisation de la modularité qui consiste à trouver une partition de des noeuds d'un réseau ayant la modularité maximale. Depuis son introduction, cette mesure est devenue un outil essentiel pour l'analyse de réseau, et de nombreux travaux ont été développés. Il a été prouvé que le problème d'optimisation en question est NP-difficile ([3]). Sa résolution est complexe à cause de la triple difficulté: les variables sont discrètes (elles représentent l'affectation des noeuds aux clusters), dans la plupart des applications le nombre de sous-réseaux (le nombre de clusters) n'est pas connu a priori (donc les algorithmes de type K-means n'y conviennent pas), et, en pratique, la taille du problème est extrêmement large. C'est pourquoi la plupart des approches classiques dans la litérature sont de nature heuristique et souvent basées sur les techniques dans les graphes. Très récemment des grands efforts ont consacrés aux approches exactes de programmation mathématique via la reformulation du problème sous la forme d'une pro-

grammation linéaire ou quadratique en variables mixtes 0-1 etc. Toutefois, ces approches sont coûteuses donc ne peuvent pas traiter des problèmes de grande taille.

Nous proposons dans ce travail une approche originale basée sur la programmation DC et DCA. Nous formulons, dans le premier temps, ce problème dans l'espace des matrices sous la forme de maximisation d'une fonction quadratique sur un produit des simplexes et en variables binaires. Ce dernier problème est ensuite reformulé, de manière élégante grâce aux techniques sophistiquées, comme la minimisation d'une forme quadratique concave sur ce même ensemble mais en variables continues qui est en fait une programmation DC. En choisissant une décomposition DC appropriée nous développons un schéma DCA dont tous les calculs sont explicites. Les avantages de cette approche sont multiples: DCA est très simple, non coûteux en temps de calcul et jouit des propriétés de convergence intéressantes - il converge vers un minimum local après un nombre fini d'itérations. Par ailleurs, bien que le nombre de cluster est un paramèttre entré de DCA, il peut être modifié par DCA lui même au cours d'algorithme pour trouver un clustering optimal. Partant d'une estimation grossière du nombre de clusters, DCA peut détecter des clusters vides, il réduit ainsi le nombre de clusters et enfin fournit un regroupement optimal. En d'autres termes, le nombre de clusters est automatiquement détecté lors des itérations de DCA. Cette originalité de DCA constitue un grand intérêt de notre approche: trouver le nombre de clusters est une question difficile et toujours d'actualité pour des chercheurs en analyse de clustering depuis plusieurs années. Les expériences numériques sont effectués sur des données réelles ayant jusqu'à 4.194.304 noeuds et $30,359,198$ arrêtes. Les résultats comparatifs avec six algorithmes de référence montrent que DCA surpasse les algorithmes de référence non seulement sur la qualité et la rapidité mais aussi sur la scalabilité. En outre, il réalise un très bon compromis entre la qualité des solutions et le temps d'exécution.

En plus de DCA, nous proposons une adaptation d'une méthode existante [19] de type clustering hierarchique dans le but de réduction du temps de calcul dans les grands graphes. En exploitant la structure de graphe nous améliorons l'algorithme glouton de Mod-Mülner et le temps de calcul de cette version est considérablement raccourci.

La deuxième classe, également en apprentissage non supervisée, est *la carte auto-organisatrice* (Selft Organizing Map en anglais, et SOM en abréviation). La carte auto-organisatrice (nous appelons dans toute la suite "SOM"), introduite par Kohonen en 1982 ([9]), et ses variantes sont parmi les plus populaires approches du réseau neuronal artificiel en apprentissage non supervisée. Le principal objectif d'une SOM est de transformer un modèle de données entrées de dimension arbitraire en une carte discrète de dimension plus faible (en général une ou deux dimensions). Cette transformation doit assurer la préservation de relations topologiques entre les données entrées, ainsi que les similitudes entre les neurones voisins sur la carte (un réseau ordonné). Grâce à ces propriétés, SOM est une manière élégante d'interprétation des données complexes et un excellent outil pour la visualisation de donnéss et l'analyse de cluster. Depuis sa naissance ([9]), SOM est utilisée comme un outil d'analyse standard dans de nombreux domaines des sciences : la statistique, le traitement du signal, la théorie de contrôle, l'analyse financière, la physique expérimentale, la chimie, la médecine , et plus récemment en collections de documents, les réseaux sociaux, le fonctionnement du cerveau biologique. Le premier algorithme de SOM a été proposé par Kohonen comme une simple procédure itérative d'apprentissage basée sur une représentation de réseau neuronal ([9, 10]). Depuis sa naissance, de nombreux algorithmes de SOM ont été développés. Une des approches principales est basée sur des techniques d'optimisation. L'idée est de choisir une fonction d'énergie dont le minimum correspond à la carte topographique désirée puis développer un algorithme d'optimisation pour trouver ce minimum. Alors que nombreuses méthodes heuristiques ont été proposées,

il existe très peu d'algorithmes déterministes dans la litérature. Nous considérons le critère de Herkes [6] qui implique un problème d'optimisation non convexe non différentiable, et développons des algorithmes basés sur DCA pour sa résolution. Il est à noter que nous sommes les premiers qui proposons une méthode déterministe pour la minimisation du critère de Herkes sous la forme d'un problème d'optimisation à deux niveaux. En utilisant une représentation matricielle nous formulons le problème sous la forme DC avec une bonne décomposition DC qui donne la naissance d'un schéma DCA simple et explicite: il nécessite seulement le produit matrice-vecteur à chaque itération. Par la suite, comme tout algorithme de SOM, une méthode d'apprentissage (learning SOM) est proposée: on itérère la minimisation du critère de Herkes par DCA, à chaque étape t, pour une suite décroissante de temps t. Grâce à l'efficacité de DCA, notre algorithme d'apprentissage nécessite un nombre très faible d'étapes (3 seulement dans notre expérimentation), alors que dans les méthodes classiques ce nombre doit être très grande (5000 au moins) pour obtenir une bonne solution.

Notre travail en programmation DC et DCA pour la modélisation, la conception et la réalisation des DCA bien adaptés aux structures spécifiques de ces deux classes de problèmes en apprentissage non supervisée est ainsi composé de :

1. Etude approfondie des modèles d'optimisation non convexe et la modélisation DC de ces problèmes. Formulations et reformulations des programmes DC équivalents : choix des décompositions DC les mieux adaptées.
2. Mise en place des DCA résultants.
3. Etude des techniques de recherche d'un bon poin initial pour DCA.
4. Implémentations et simulations numériques pour leurs validations.
5. Mise en valeur l'algorithme de maximisation de modularité à travers ses applications dans les réseaux sociaux.

De manière analogue, nous étudions la troisième classe des problèmes qui concerne la sélection des variables en S3VM (Semi Supervised Support Vector Machine en anglais). L'apprentissage semi-supervisée se situe entre l'apprentissage supervisée et l'apprentissage non-supervisée - elle utilise un ensemble de données étiquettées et non étiquettées. Il a été démontré que l'utilisation de données non étiquettées, en combinaison avec des données étiquettées, permet d'améliorer significativement la qualité de l'apprentissage. Ceci étant, au point de vue algorithmique, la sélection des variables en S3VM est plus difficile que la sélection des variables en SVM. Il y a la double difficulté venant d'une part de la norme zéro et d'autre part de la non convexité et non différentiabilité de la fonction de perte. Il existe très peu d'algorithmes pour la sélection des variables en S3VM. La résolution des problèmes d'optimisation résultants devrait (naturellement) recourir à la programmation DC et DCA. Nous considérons la régularisation l_0 avec cinq approximations de la norme zéro ainsi que la technique de pénalité exacte, les formulons sous la forme DC puis développons les schémas DCA pour six programmes DC résultants. Avec une décomposition DC appropriée, tous nos schémas DCA consistent en résolution d'une programmation linéaire à chaque itération. Ainsi, finalement, nous constatons avec surprise que DCA procède la sélection des variables en SVM et S3VM de la même "manière", et donc la même complexité ! Les expérimentations numériques comparatives avec le modèle de régularisation l_1 (auquel nous appliquons également DCA) ont prouvé la performante et la superiorité de nos méthodes, aussi bien en classification qu'en sélection de variables.

La quatrième classe, en apprentissage supervisée (qui n'utilise que des données étiquettées), concerne la sélection des variables en classification par les machines à vecteurs de support, cas multi-classe (Multi-class Support Vector Machine en anglais, MSVM en abréviation). La sélection des variables en MSVM, ou classification supervisée par un MSVM parcimonieux, est une problématique se situant dans un domaine plus large appelé *l'apprentissage avec la parcimonie.* Au cours de cette dernière décenie ce domaine de recherche a attiré une attention particulière du monde des chercheurs en MLDM de par ses nombreuses applications importantes. De manière générale, un modèle d'apprentissage avec la parcimonie implique la minimisation d'une fonction qui est composée de deux partie: la première, la fonction de perte, représente "l'erreur" d'apprentissage, et la deuxième, appelée "la régularisation", contient la norme zéro du vecteur (définie comme le nombre de ses composantes non nulles) correspondant aux (ou à une partie des) variables du problème d'optimisation. La minimisation de la norme zéro (ou, plus généralement, d'une fonction contenant la norme zéro) est un problème NP-difficile [1] qui reste toujours un défi pour les chercheurs en optimisation et MLDM. La difficulté vient de la discontinuté de cette norme. Pour surmonter cet obstacle, trois approches principales ont été développées: la relaxation convexe, l'approximation non convexe continue, et la pénalité exacte. La méthode la plus populaire appartenant au groupe "relaxation convexe" (connue sous le nom LASSO [23]) consiste à remplacer la norme zéro par la norme l_1. La méthode la plus récente et efficace proposée dans [18, 15, 16] calcule la meilleure borne inférieure convexe de la régularisation $l_2 - l_0$ via sa bi-conjugé. Durant la même période, les approches d'approximation non convexes continues ont été largement développées dans différents contextes de MLDM, en particulier pour la sélection des variables dans SVM et en regression (voir par exemple [2, 5, 25, 20, 12, 13, 24]). De nombreuses approximations non convexes ont été proposées qui sont toutes des fonctions DC et la plupart des algorithmes sont basés sur DCA. La troisième approche introduite dans [12, 14, 18] est plus original au sens où on résout un problème DC équivalent (et non "approximation" comme dans les deux premières approches) grâce à la reformulation par la technique de pénalité exacte en programmation DC. Une étude complète de la programmation DC et DCA pour la minimisation de la norme zéro via les approximations non convexes et la pénalité exacte est donnée dans [14].

Dans ce travail nous adoptons les méthodes développées dans [14] au contexte de sélection de variables en MSVM qui consiste à déterminer un MSVM le plus parcimonieux possible pour classifier un ensemble de données étiquettées. En considérant le modèle MSVM avec les deux régularisations l_0 et $l_2 - l_0$ et utilisant cinq approximations non convexes continues de la norme zéro, nous obtenons 10 programmes DC différents et mettons en place des DCA correspondants. Tous les schémas DCA pour le modèle de régularisation l_0 (respectivement $l_2 - l_0$) consistent à la résolution d'une programmation linéaire (respectivement quadratique convexe) à chaque itération. D'autre part, nous proposons deux schémas DCA pour la résolution de deux problèmes DC reformulés via la pénalité exacte. Des expérimentations numériques sur nombreux jeux de données sont réalisés. Elles ont pour but d'identifier le(s) meilleur(s) schémas DCA, c.à..d celui (ceux) qui réalise(nt) le meilleur compromise entre l'erreur de classification et le nombre de variables sélectionnées ainsi que celle entre "la qualité" et "la rapidité".

Organisation du manuscrit

La thèse est composée de cinq chapitres. Dans le premier chapitre nous décrivons de manière succincte la programmation DC et DCA, des outils théoriques et algorithmiques servant des références aux autres. Chacun de quatre chapitres suivants est consacré à la présentation d'une de classes de problèmes abordées ci-dessus. Les chapitres 2 et 3 concernent, respectivement, le clustering des sommets d'un graphe par maximisation de la

modularité et la carte auto-organisatrice. Le chapitre 4 est destiné à la sélection des variables en S3VM et le chapitre 5 - la sélection des variables en MSVM. Enfin, un paragraphe sur des perspectives de nos travaux clôture le manuscrit.

Tous ces travaux, à l'étape finale suite aux nombreuses améliorations successives, font l'objet des articles soumis dans des journaux. Par suite, les quatre derniers chapitres seront présentés sous forme d'articles.

References

[1] Amaldi, E., Kann V.: On the approximability of minimizing non zero variables or unsatisfied relations in linear systems. Theoretical Computer Science 209, pp. 237–260, 1998.

[2] Bradley, B.S., Mangasarian, O.L.: Feature selection via concave minimization and support vector machines. Machine Learning Proceedings of the Fifteenth International Conferences (ICML'98), pp. 82–90, J. Shavlik editor, MorganKaufmann, San Francisco, California 1998.

[3] Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE Transactions on Knowledge and Data Engineering 20, pp. 172–188, 2008.

[4] Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm, J. Roy. Stat. Soc. B., 1977.

[5] Fan, J., Li, R.: Variable selection via nonconcave penalized likelihood and its Oracle Properties. Journal of the American Statistical Association 96, pp. 1348–1360, 2001.

[6] Heskes, T.: Energy functions for self organizing maps. Kohonen Maps, pp. 303–315, 1999.

[7] Reiner Horst , Hoang Tuy: Global Optimization: Deterministic Approaches. Springer, 1996.

[8] Reiner Horst, Panos M. Pardalos, Nguyen Van Thoai: Introduction to Global Optimization. Kluwer Academic Publisher, 1995.

[9] Kohonen, T.: Analysis of a simple self-organizing process. Biol. Cybern. 44, pp. 135–140, 1982.

[10] Kohonen, T.: Self-Organization Maps. Springer Heidelberg, 1997.

[11] Le Thi, H.A.: DC programming and DCA. `http://lita.sciences.univ-metz.fr/~lethi/DCA.html`.

[12] Le Thi, H.A., Le Hoai, M., Nguyen, V.V., Pham Dinh, T.: A DC Programming approach for Feature Selection in Support Vector Machines learning. Journal of Advances in Data Analysis and Classification 2(3), pp. 259–278, 2008.

[13] Le Thi, H.A., Nguyen, V.V., Ouchani: Gene selection for cancer classification using DCA. Journal of Fonctiers of Computer Science and Technology 3(6), 2009.

[14] Le Thi, H.A., Pham Dinh, T.: DC Programming and DCA for solving nonconvex programs involving ℓ_0-norm. National Institute for Applied Sciences, Rouen, forthcoming, 2011.

[15] Le Thi, H.A., Pham Dinh, T., Thiao M.: Large scale sparse ridge regression: a new and efficient convex regularization approach. Submitted.

[16] Le Thi, H.A., Pham Dinh, T., Thiao M.: Learning with sparsity by a new and efficient convex approach for ℓ_2-ℓ_0 regularization. Submitted.

[17] Le Thi, H.A.: DC programming and DCA in Machine Learning. Technique report, University of Lorraine, submitted, 2012.

[18] Pham Dinh, T., Le Thi, H.A.: Recent advances on DC programming and DCA. To appear in Transactions on Computational Collective Intelligence, Springer, 2013.

[19] Müllner, Daniel: Modern hierarchical, agglomerative clustering algorithms. Lecture Notes in Computer Science, volume 3918(1973), Berlin: Springer-Verlag, 2011, url=http://arxiv.org/abs/1109.2378.

[20] Neumann, J., Schnorr, C., Steidl, G.: Combined SVM-based feature selection and classification. Machine Learning 61(1–3), pp. 129–150, 2005.

[21] Newman, M.E.J.: Networks: An Introduction. Oxford University Press, 2010.

[22] Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E 69(2), 2004.

[23] Tibshirani, R.: Regression shrinkage and selection via the lasso. J. Roy. Stat. Soc. 46, pp. 431–439, 1996.

[24] Wei Luo, Lipo Wang, Jingjing Sun: Feature Selection for Cancer Classification Based on Support Vector Machine. Intelligent Systems, 2009. GCIS '09. WRI Global Congress 4, pp. 422–426, 2009.

[25] Weston, J., Elisseeff, A., Scholkopf, B.: Use of Zero-Norm with Linear Models and Kernel Methods. Journal of Machine Learning Research 3, pp. 1439–1461, 2003.

[26] Yuille, A.L., Rangarajan, A.: The Convex Concave Procedure. Neural Computation 15(4), pp. 915–936, 2003.

Chapter 1

DC programming and DCA

1.1 Introduction

In this chapter, we report a brief presentation about DC programming and DCA that we will be most useful in the sequel. These results are extracted from those presented in H.A. Le Thi 1994 ([3]), H.A. Le Thi 1997 ([4]), H.A. Le Thi and T. Pham Dinh 1997 ([5]), H.A. Le Thi and T. Pham Dinh 2005 ([7]), T. Pham Dinh and H.A. Le Thi 1998 ([13]). For a detailed discussion we refer to these references (see also [6], [8]-[12] and [14]).

1.2 DC programming and DCA

In recent years there has been a very active research in nonconvex programming. There are two different but complementary approaches, we can say two schools, in DC programming:

1. Combinatorial approach to global continuous optimization: this terminology is due to the fact that new introduced tools were inspired by the concepts of combinatorial optimization in global continuous optimization.
2. Convex analysis approach to nonconvex programming.

The first approach is older and inspired by tools and methods developed in the combinatorial optimization, but with the difference that one works in the continuous frame work. In this approach, optimal solutions are located by using the approximation methods, for instance, cutting techniques, decomposition methods, branch and bound, etc. The pioneer of this approach is H. Tuy whose first work was introduced in 1964. His work is abundant, included books by Horst-Tuy ([15, 17]) which present the theory, algorithms and applications of global optimization. Among the most important contributions to this approach, it is worth citing the ones by Hoang Tuy, R. Horst, H. Benson, H. Konno, P. Pardalos, Le Dung Muu, Le Thi Hoai An, Nguyen Van Thoai, Phan Thien Thach and Pham Dinh Tao. However, most robust and efficient global algorithms for solving D.C. programs actually do not meet the expected desire: solving real life d.c. programs in their true dimension.

The second approach relies on the powerful arsenal of analysis and convex optimization. The first work, due to Pham Dinh Tao (1975), concerning the calculation of matrix norms (fundamental problem in numerical analysis) that is a problem of maximizing a convex function over a convex set. The work of Toland (1978) [18] on the duality and optimality local DC optimization generalizes elegantly the results established by Pham Dinh Tao in convex maximization. The DC optimization theory is developed by Pham Dinh Tao, J. B. Hiriart Urruty, Jean - Paul Penot, Phan Thien Thach, Le Thi Hoai An. On the algorithmic part of the second approach, currently available as DCA (DC Algorithms) introduced by Pham Dinh Tao (1986), which are based on optimality conditions and duality in DC optimization. But it took until the joint work of Le Thi Hoai An and Pham Dinh Tao (see [3]-[6] and [11]-[13]) to show that it definitely needed in nonconvex optimization as one of the simplest and most performance algorithms, capable of handling large problems.

1.2.1 Notations and properties

This paragraph is devoted to a brief recall of convex analysis for facilitating the reading of certain passages. For more details, we refer to the work of P.J Laurent [2], of R.T

Rockafellar [16] and of A. Auslender [1]. Let X be the Euclidean space $\mathbb{R}^n$, $\langle .,. \rangle$ be the scalar product, $\|x\| = \langle x, x \rangle^{\frac{1}{2}}$ be the Euclidean norm, and the dual vector space of X is denoted by Y, which can be identified with X itself. We use an usual tool of convex analysis where a function can take the infinite value $\pm\infty$ [16]. We note $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$. A function $f : S \longrightarrow \overline{\mathbb{R}}$ is defined on a convex set S in X, the effective domain of f, denoted by $\mathrm{dom}(f)$, is

$$\mathrm{dom}(f) = \{x \in S : f(x) < +\infty\} \tag{1.1}$$

and the epigraph of f, denoted by $\mathrm{epi}(f)$, is

$$\mathrm{epi}(f) = \{(x, \alpha) \in S \times \mathbb{R} : f(x) < \alpha\}.$$

If $\mathrm{dom}(f) \neq \emptyset$ and $f(x) > -\infty$ for all $x \in S$ then we say that the function $f(x)$ is *proper*.

A function $f : S \longrightarrow \overline{\mathbb{R}}$ is called *convex* if its epigraph is a convex set in $X \times \overline{\mathbb{R}}$. This is equivalent to saying that S is a convex set in X and for all $\lambda \in [0, 1]$ we have

$$f((1-\lambda)x^1 + \lambda x^2) \leq (1-\lambda)f(x^1) + \lambda f(x^2) : \forall x^1, x^2 \in S. \tag{1.2}$$

Let $\mathrm{Co}(X)$ be the set of convex functions on X.

In (1.2), if the strict inequality holds for all $\lambda \in]0, 1[$ and for all $x^1, x^2 \in S$ with $x^1 \neq x^2$ then f is called *strictly convex* function.

A function f is called *strongly convex* on a convex set C if there exists a number $\rho > 0$ such that

$$f((1-\lambda)x^1 + \lambda x^2) \leq (1-\lambda)f(x^1) + \lambda f(x^2) - (1-\lambda)\lambda\frac{\rho}{2}\|x^1 - x^2\|^2, \tag{1.3}$$

for all $x^1, x^2 \in C$, and for all $\lambda \in [0, 1]$. It is amount to saying that $f - \frac{\rho}{2}\|.\|^2$ is convex on C. The *modulus of strong convexity* of f on C, denoted by $\rho(f, C)$ or $\rho(f)$ if $C = X$, is given by

$$\rho(f, C) = Sup\{\rho \geq 0 : f - \frac{\rho}{2}\|.\|^2 \text{ is convex on } C\} > 0. \tag{1.4}$$

Clearly, f is convex on C if and only if $\rho(f, C) \geq 0$. One says that f is *strongly convex* on C if $\rho(f, C) > 0$.

Theorem 1.1 *f strongly convex $\Longrightarrow$ f strictly convex $\Longrightarrow$ f convex.*

Let f be a proper convex function on X, a vector $y^0 \in Y$ is called a *subgradient* of f at a point $x^0 \in \mathrm{dom}(f)$ if

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

The set of all subgradients of f at x^0 is called the *subdifferential* of f at x^0 and is denoted by $\partial f(x^0)$.
Let $\epsilon > 0$, a vector y^0 is called ϵ-subgradient of f at point x^0 if

$$\langle y^0, x - x^0 \rangle + f(x^0) \leq f(x) + \epsilon \quad \forall x \in X.$$

Then the set of all $\epsilon - subgradients$ of f at point x^0 is called the the ϵ-subdifferential of f at x^0 and is denoted by $\partial_\epsilon f(x^0)$.

A function $f : S \longrightarrow \mathbb{R}$ is called *lower semi-continuous* (l.s.c) at a point $x \in S$ if

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

Let $\Gamma_0(X)$ be the set of all l.s.c proper convex functions on X.

Definition 1.1 *Let a function $f : X \longrightarrow \mathbb{R}$, the* conjugate function *$f^\star$ of f, is a function belonging to $\Gamma(Y)$ and defined by*

$$f^\star(y) = \sup\{\langle x, y\rangle - f(x) : x \in X\}. \tag{1.5}$$

$f^\star$ is an upper envelope of continuous affine functions $y \mapsto \langle x, y\rangle - f(x)$ on Y.

The main properties are summarized in the following proposition that will be needed for further:

Proposition 1.1 *If $f \in \Gamma_0(X)$ then:*

- *$f \in \Gamma_0(X) \Longleftrightarrow f^\star \in \Gamma_0(Y)$. In this case, we have $f = f^{\star\star}$,*
- *$y \in \partial f(x) \Longleftrightarrow f(x) + f^\star(y) = \langle x, y\rangle$ and $y \in \partial f(x) \Longleftrightarrow x \in \partial f^\star(y)$,*
- *$\partial f(x)$ is a closed convex set,*
- *If $\partial f(x) = \{y\}$ then f is differentiable at x and $\nabla f(x) = y$,*
- *$f(x^0) = min\{f(x), x \in X\} \Longleftrightarrow 0 \in \partial f(x^0)$.*

1.2.2 Fundamentals of DC analysis

Polyhedral convex functions

A convex set C is called a *polyhedral convex* if

$$C = \bigcap_{i=1}^{m} \{x : \langle a_i, x\rangle - \alpha_i \leq 0\} \text{ where } a_i \in Y, \alpha_i \in \mathbb{R}, \quad \forall i = 1, ..., m.$$

A function is called a polyhedral convex if

$$f(x) = \sup\{\langle a_i, x\rangle - \alpha_i : i = 1, ..., k\} + \chi_C(x).$$

where C is a polyhedral convex set and $\chi_C(.)$ stands for the indicator function of C, i.e. $\chi_C(x) = 0$ if $x \in C$ and $+\infty$ otherwise.

Proposition 1.2 *([16])*

- *Let f be a polyhedral convex function. f is everywhere finite if and only if $C = X$,*
- *f is polyhedral convex then $f^\star$ is also polyhedral. Moreover, if f is everywhere finite then*

$$f(x) = \sup\{\langle a_i, x\rangle - a_i : i = 1, ..., k\},$$
$$dom(f^\star) = co\{a_i : i = 1, ..., k\},$$
$$f^\star(y) = \min\{\Sigma_{i=1}^k \lambda_i \alpha_i : y = \Sigma_{i=1}^k \lambda_i a_i, \lambda_i \geq 0, \Sigma_{i=1}^k \lambda_i = 1\},$$

- *If f is polyhedral then $\partial f(x)$ is a nonempty polyhedral convex set at every point $x \in dom(f)$.*

DC functions

A function $f : \Omega \mapsto \overline{\mathbb{R}}$ defined on a convex set convex $\Omega \subset \mathbb{R}^n$ is called DC on Ω if it can be presented in the form of difference of two convex functions on Ω, i.e.

$$f(x) = g(x) - h(x),$$

where g and h are convex functions on Ω, $g - h$ is called a DC decomposition of f. Let $DC(\Omega)$ be the set of all DC functions on Ω, and $DC_f(\Omega)$ in case of g and h are finite convex on Ω.

DC functions have many important properties that were derived from 1950s by Alexandroff (1949), Landis (1951) and Hartman (1959); one of the main properties is their stability with respect to frequently used operations in optimization. Specifically:

Proposition 1.3 *(i) A linear combination of DC functions on Ω is DC on Ω,*

(ii) The upper envelope of a finite set of finite DC functions on Ω is DC on Ω, The lower envelope of a finite set of finite DC functions on Ω is DC on Ω,

(iii) Let $f \in DC_f(\Omega)$, then $|f(x)|, f^+(x) = max\{0, f(x)\}$ and $f^-(x) = min\{0, f(x)\}$ are DC on Ω.

These results generalize to the case of value in $\mathbb{R} \cup \{+\infty\}$ ([4]). It follows that the set of DC functions on Ω is a vector space ($DC(\Omega)$): it is the smallest vector space containing all convex functions on $\Omega(Co(\Omega))$.

Theorem 1.2 *Given a DC function f and a DC decomposition $f = g - h$, then for any finite convex function φ, $f = (g+\varphi)-(h+\varphi)$ gives another DC decomposition of f. Thus, a DC function has an infinite number of DC decompositions.*

Denoted by $C^2(\mathbb{R}^n)$, the class of twice continuously differentiable functions on $\mathbb{R}^n$.

Proposition 1.4 *Any function $f \in C^2(\mathbb{R}^n)$ is DC on an arbitrary compact convex set $\Omega \subset \mathbb{R}^n$.*

Since the subspace of polynomials on Ω is dense in the space $C(\Omega)$ of continuous functions on Ω, we have:

Corollaire 1.1 *The space of DC functions on a compact convex set $\Omega \subset \mathbb{R}^n$ is dense in $C(\Omega)$, i.e.*

$$\forall e > 0, \exists F \in C(\Omega) : |f(x) - F(x)| \leq \epsilon \quad \forall x \in \Omega.$$

Note that DC functions occur very frequently in practice, both differentiable and non-differentiable optimization. An important result established by Hartman (1959) permits identified DC functions in many situations, simply by using a local analysis of the convexity (local convex, local concave and local DC).

A function $f : D \mapsto \mathbb{R}$ defined on an open convex set $D \in \mathbb{R}^n$ is called local DC if for all $x \in D$ there is an open convex neighborhood U of x and a pair of convex functions g, h on U such that $f|_U = g|_U - h|_U$.

Proposition 1.5 *A local DC function on a convex set D is DC on D.*

DC optimization problem

Due to the preponderance and wealthy properties of DC functions, the transition of the subspace $Co(\Omega)$ to the vector space $DC(\Omega)$ permits to expand significantly convex optimization problems in the non-convexity. The field of optimization problems involving DC functions is relative large and open, covering most of the problems encountered in applications.

However, we can not immediately deal with any non-convex and non-differentiable optimization problem. The following classification has now become classic:

$$(1) \quad \sup\{f(x) : x \in C\}, \text{where f and C are convex}$$

$$(2) \quad \inf\{g(x) - h(x) : x \in X\}, \text{ where g and h are convex}$$

$$(3) \quad \inf\{g(x) - h(x) : x \in C, f_1(x) - f_2(x) \leq 0\},$$

where g, h, f_1, f_2 and C are convex, these seem to be large enough to contain substantially all nonconvex problems encountered in everyday life. Problem (1) is a special case of Problem (2) with $g = \chi_C$, the indicator function of C and $h = -f$. Problem (2) can be modified in the form equivalent to (1)

$$\inf\{t - h(x) : g(x) - t \leq 0\}.$$

While Problem (3) can be transformed to the form (2) by using exact penalty related to the DC constraints $f_1(x) - f_2(x) \leq 0$. Its resolution can also be reduced under certain technical conditions, that is a series of Problems (1). Problem (2) is called *a DC program*. It is a major interest both from practical and theoretical point of view. From the theoretical point of view, we can note that, as we noted above, the class of DC functions is remarkable stable with the operations frequently used in optimization. Moreover, there is an elegant duality theory ([8, 9, 18, 19, 3, 4, 5]) which, as convex optimization, has profound practical implications for numerical methods.

DC algorithms (DCA) is introduced by Pham Dinh Tao ([10, 11]) who presented a new approach based on the DC theory. In fact, these algorithms are a generalization of subgradients algorithms which were studied by the same author on the convex maximization ([8, 10]). However, it was until the joint work of Le Thi et Pham Dinh during the past decades (see [3]-[6] and [11]-[13])) that DCA has now become classical and popular.

Duality in DC programming

In convex analysis, the concept of duality (conjugate function, dual problem, etc.) is a very powerful fundamental concept. For convex problems and in particular linear, a duality theory has been developed over several decades [16]. More recently, an important

concept of duality in nonconvex analysis has been proposed and developed, first for convex maximization problems, before reaching the DC problems. DC duality introduced by Toland (1978) can be regarded as a generalization of logic work of Pham Dinh Tao (1975) on convex maximization. We will present below the main results (in DC optimization) on optimal conditions (local and global) and the DC duality. For more details, the reader is referred to the document of Le Thi (1997) (see [5]).

Let space $X = \mathbb{R}^n$, usual inner product $\langle ., . \rangle$ and the Euclidean norm $\|.\|$. Let Y be the dual space of X which can be identified with X itself and $\Gamma_0(X)$ be the set of all proper l.s.c convex functions on X.

Given $g(x)$ and $h(x)$ are two proper convex functions on X $(g, h \in \Gamma_0(X))$, considering the DC problem

$$\inf\{g(x) - h(x) : x \in X\} \quad (P)$$

and the dual problem

$$\inf\{h^\star(y) - g^\star(y) : y \in Y\} \quad (D)$$

where $g^\star(y)$ (resp. $h^\star(y)$) denotes the conjugate function of g (resp. h).

The results of DC duality defined by using the conjugate functions give an important relationship in DC optimization [18].

Theorem 1.1 *Let g and $h \in \Gamma_0(X)$, then*

(i)

$$\inf_{x \in dom(g)} \{g(x) - h(x)\} = \inf_{y \in dom(h^\star)} \{h^\star(y) - g^\star(y)\} \tag{1.6}$$

(ii) If y^0 is a minimizer of $h^\star - g^\star$ on Y then $x^0 \in \partial g^\star(y^0)$ is a minimizer of $g - h$ on X.

Proof 1.1 *(i)*

$$\begin{aligned} \alpha &= \inf\{g(x) - h(x) : x \in X\} \\ &= \inf\{g(x) - \sup\{\langle x, y\rangle - h^\star(y) : y \in Y\} : x \in X\} \\ &= \inf\{g(x) + \inf\{h^\star(y) - \langle x, y\rangle : y \in Y\} : x \in X\} \\ &= \inf_x \inf_y \{h^\star(y) - \langle x, y\rangle - g(x)\} \\ &= \inf\{h^\star(y) - g^\star(y) : y \in Y\}. \end{aligned}$$

(ii) cf. Toland ([18]).

The theorem 1.1 shows that solving the primal problem (P) implies resolution of the dual problem D and vice versa.

1.2.3 DC optimization

Global optimality in DC optimization

In convex optimization, x^0 minimizes a function $f \in \Gamma_0(X)$ if and only if $0 \in \partial f(x^0)$. In DC optimization, the following global optimality conditions [20] are formulated by using

ϵ-subdifferential of g and h. His demonstration is based on studying the behavior of ϵ-subdifferential of a convex function depending on the parameter ϵ. The demonstration in [4] is more simple and suitable in case of DC optimization: it simply expresses that global optimality condition is a geometry translation and optimal values of primal and dual DC programs are equal.

Theorem 1.2 ***(Global DC optimization)*** *Let $f = g - h$ where $g, h \in \Gamma_0(X)$ then x^0 is a global minimizer of $g(x) - h(x)$ on X if and only if,*

$$\partial_\epsilon h(x^0) \subset \partial_\epsilon g(x^0) \quad \forall \epsilon > 0. \tag{1.7}$$

Theorem 1.3

(i) If $f \in \Gamma_0(X)$, we can write $f = g - h$ with $f = g$ and $h = 0$. In this case, the global optimal in (P) - which is the same as the local optimal in (P) (because (P) is a convex problem) - is characterized by,

$$0 \in \partial f(x^0). \tag{1.8}$$

The fact that $\partial_\epsilon h(x^0) = \partial h(x^0) = \{0\}, \quad \forall \epsilon > 0, \forall x \in X$, the relationship (1.8) is equivalent to (1.7).

(ii) More generally, considering DC decompositions of $f \in \Gamma_0(X)$ in the form $f = g - h$ with $g = f + h$ and $h \in \Gamma_0(X)$ finite everywhere on X. The corresponding DC problem is a "false" DC problem because it is a convex optimization problem. In this case, the relationship (1.8) is equivalent to

$$\partial h(x^0) \subset \partial g(x^0).$$

(iii) We can therefore say that (1.7) clearly marks the transition from convex optimization to nonconvex optimization. This feature of the global optimality of (P) indicates the complexity of its practical use because it appeals to all ϵ-subdifferential at x^0.

Local optimality in DC optimization

We have seen that the relationship $\partial h(x^0) \subset \partial g(x^0)$ (using the subdifferential "exact") is necessary and sufficient condition of global optimization for a "false" DC problem (a convex optimization problem). In a global optimization problem, the minimization function is local convex "around" a local minimum, then it is clear that the relation of subdifferential inclusion will characterize a local minimum of a DC problem.

Definition 1.2 *Let g and $h \in \Gamma_0(X)$. A point $x^\bullet \in dom(g) \cap dom(h)$ is a local minimizer of $g(x) - h(x)$ on X if and only if*

$$g(x) - h(x) \geq g(x^\bullet) - h(x^\bullet), \quad \forall x \in V_{x^\bullet}, \tag{1.9}$$

where V_x denotes a neighborhood of x.

Proposition 1.6 ***(Necessary condition of local optimality)*** *If $x^\bullet$ is a local minimizer of $g - h$ then*

$$\partial h(x^\bullet) \subset \partial g(x^\bullet). \tag{1.10}$$

Proof 1.2 *If $x^\bullet$ is a local minimizer of $g - h$, then there exists a neighborhood V_x of x such that*

$$g(x) - g(x^\bullet) \geq h(x) - h(x^\bullet), \quad \forall x \in V_{x^\bullet}. \tag{1.11}$$

Therefore if $y^\bullet \in \partial h(x^\bullet)$ then

$$g(x) - g(x^\bullet) \geq \langle x - x^\bullet, y^\bullet \rangle, \quad \forall x \in V_{x^\bullet} \tag{1.12}$$

which is equivalent, under the convexity of g, at $y^\bullet \in \partial g(x^\bullet)$.

Note that for a number of DC problems and in particular for h polyhedral ones, the necessary condition (1.10) is also sufficient, as we will see a little further. We say that $x^\bullet$ is a critical point of $g - h$ if $\partial h(x^\bullet) \cap \partial g(x^\bullet)$ is non empty [18]. It is a weakened form of subdifferential inclusion. The search for such a critical point is at the DCA (simple form) which will be studied in the next section. In general, DCA converges to a local solution of a DC optimization problem. However, in theory, it is important to formulate sufficient conditions for local optimality.

Theorem 1.3 ***(Sufficient condition of local optimality ([4, 5]))*** *If $x^\star$ admits a neighborhood V such that*

$$\partial h(x) \cap \partial g(x^\star) \neq \emptyset, \quad \forall x \in V \cap dom(g), \tag{1.13}$$

then $x^\star$ is a local minimizer of $g - h$.

Corollaire 1.2 *If $x \in int(dom(h))$ verifies*

$$\partial h(x) \in int(\partial g(x)),$$

then x is a local minimizer of $g - h$.

Corollaire 1.3 *If $h \in \Gamma_0(X)$ is polyhedral convex then $\partial h(x) \subset \partial g(x)$ is a necessary and sufficient condition for x is a local minimizer of $g - h$.*

Proof 1.3 *This result generalizes the first obtained by C. Michelot in this case where $g, h \in \Gamma_0(X)$ are finite everywhere and h polyhedral convex [4, 5].*

For solving a DC optimization problem, it is sometimes easier to solve the dual problem (D) than the primal problem (P). Theorem 1.1 provides transportation by duality of global minimizers. We establish the same duality transportation of local minimizers.

Corollaire 1.4 ***(DC duality transportation of local minimizers [4, 5])*** *Supposed that $x^\bullet \in dom(\partial h)$ is a local minimizer of $g-h$, let $y^\bullet \in \partial h(x^\bullet)$ and $V_{x^\bullet}$ a neighborhood of $x^\bullet$ such that $g(x)-h(x) \geq g(x^\bullet)-h(x^\bullet), \quad \forall x \in V_{x^\bullet} \cap dom(g)$. If*

$$x^\bullet \in int(dom(g^\star)) \quad and \quad \partial g^\star(y^\bullet) \subset V_{x^\bullet}, \tag{1.14}$$

then $y^\bullet$ is a local minimizer of $h^\star - g^\star$.

Proof 1.4 *It is implied immediately from the Proposition 1.1 by restricting f in the interval $V_{x^\bullet} \cap dom(g)$.*

Theorem 1.4 *Obviously, by duality, all the results in this section are transposed to the dual problem D. For example: if y is a local minimizer of $h^\star - g^\star$, then $\partial g^\star(y) \subset \partial h^\star(y)$.*

1.2.4 DCA

This is a new method based on subgradient optimality and duality in DC optimization (non differential). This approach is completely different from classical subgradient methods in convex optimization. In DCA, the construction algorithm seeks to exploit the structure of the DC problem. It requires, first, to have a DC representation of the function to minimize, i.e. $f = g - h$ (g, h convex), because all transactions work only with the convex components. The sequence of descent directions is obtained by computing a sequence of subgradients not directly from the function f, but from convex components of primal and dual problems.

Principle of DCA

The construction of DCA, discovered by Pham Dinh Tao (1986), is based on characterization of local solutions in DC optimization of primal (P) and dual (D) problems.

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

$$\alpha = \inf\{h^\star(y) - g^\star(y) : y \in Y\} \quad (D).$$

The DCA consists of constructing two sequences $\{x^k\}$ and $\{y^k\}$. The first sequence is a candidate to be a solution of the primal problem and the second of the dual problem. These two sequences are related by duality and verify the following properties:

- the sequences $\{g(x^k) - h(x^k)\}$ and $\{h^\star(y^k) - g^\star(y^k)\}$ are decreasing,
- and if $(g-h)(x^{k+1}) = (g-h)(x^k)$ then the algorithm stops at $(k+1)^{th}$ iteration and the point x^k (resp. y^k) is a critical point of $g-h$ (resp. $h^\star - g^\star$),
- otherwise every limit point $x^\bullet$ of $\{x^k\}$ (resp. $y^\bullet$ of $\{y^k\}$) is a critical point of $g-h$ (resp. $h^\star - g^\star$).

The algorithm ultimately seeks a couple $(x^\bullet, y^\bullet) \in X \times Y$ such that $x^\bullet \in \partial g^\star(y^\bullet)$ and $y^\bullet \in \partial h(x^\bullet)$.

Schema of simplified DCA
The main idea of the implementation of the algorithm (simple form) is to construct a sequence $\{x^k\}$, verify at each iteration $\partial g(x^k) \cap \partial h(x^{k-1}) \neq \emptyset$, convergence to a critical point $x^\bullet(\partial h(x^\bullet) \cap \partial g(x^\bullet) \neq \emptyset)$ and symmetrically, similar way by duality, a sequence $\{y^k\}$ such that $\partial g^\star(y^{k-1}) \cap \partial h^\star(y^k) \neq \emptyset$ convergence to a critical point.
They are constructed as follows:

Algorithm: DCA generic scheme
Step 0. Choose an initial point x^0.
Step 1. For each k, x^k is known, computing $y^k \in \partial h(x^k)$.
Step 2. Finding $x^{k+1} \in \partial g^\star(y^k)$.
Step 3. If stopping test is verified **STOP**; otherwise $k \leftarrow k+1$, goto Step 1.

This description, with the help of iteration diagrams of fixed points of multi-applications ∂h and $\partial g^\star$, thus appears to be very simple.

Existence of generated sequences

The DCA algorithm is well defined if we can actually build the two sequences $\{x^k\}$ and $\{y^k\}$ as above from an arbitrary initial point x^0.

- By construction, if $x^0 \in \text{dom}(\partial h)$, then $y^0 \in \partial h(x^0)$ is well defined.
- For $k \geq 1$, y^k is well defined if and only if x^k is defined and contained in $\text{dom}(\partial h)$; consequently, x^k and y^k are well defined if and only if $\partial g^\star(y^{k+1}) \cap \text{dom}(\partial h)$ is non empty, which implies that $y^{k+1} \in \text{dom}(\partial g^\star)$.

Lemme 1.1 *[5] The sequences $\{x^k\}$, $\{y^k\}$ in DCA are well defined if and only if*

$$dom(\partial g) \subset dom(\partial h), \quad and \quad dom(\partial h^\star) \subset dom(\partial g^\star).$$

The convergence of the algorithm is ensured by the following results [5]:

Let ρ_i and $\rho_i^\star, (i = 1, 2)$ be positive real numbers such that $0 \leq \rho_i < \rho(f_i)$ (resp. $0 \leq \rho_i^\star < \rho_i^\star(f_i^\star)$) where $\rho_i = 0$ (resp. $\rho_i^\star = 0$) if $\rho(f_i) = 0$ (resp. $\rho(f_i^\star) = 0$) and ρ_i (resp. $\rho_i^\star$) can take the value $\rho(f_i)$ (resp. $\rho(f_i^\star)$) if this upper bound is reached. We consider $f_1 = g, f_2 = h$.

Theorem 1.4 *If the sequences $\{x^k\}$ and $\{y^k\}$ are well defined, then we have:*

(i)

$$(g-h)(x^{k+1}) \leq (h^\star - g^\star)(y^k) - \frac{\rho_h}{2}\|dx^k\|^2 \leq (g-h)(x^k) - \frac{\rho_1+\rho_2}{2}\|dx^k\|^2$$

(ii)

$$(h^\star - g^\star)(y^{k+1}) \leq (g-h)(x^{k+1}) - \frac{\rho_1^\star}{2}\|dy^k\|^2 \leq (h^\star - g^\star)(y^k) - \frac{\rho_1^\star+\rho_2^\star}{2}\|dy^k\|^2$$

where $dx^k = x^{k+1} - x^k$

Corollaire 1.5 *([5])**(Convergence)***

1.
$$\begin{aligned}(g-h)(x^{k+1}) &\leq (h^\star - g^\star)(y^k) - \tfrac{\rho_2}{2}\|dx^k\|^2 \\ &\leq (g-h)(x^k) - [\tfrac{\rho_2}{2}\|dx^{k-1}\|^2 + \tfrac{\rho_1^\star}{2}\|dy^k\|^2]\end{aligned}$$

2.
$$\begin{aligned}(g-h)(x^{k+1}) &\leq (h^\star - g^\star)(y^k) - \tfrac{\rho_2^\star}{2}\|dx^k\|^2 \\ &\leq (g-h)(x^k) - [\tfrac{\rho_2^\star}{2}\|dx^{k-1}\|^2 + \tfrac{\rho_1^\star}{2}\|dy^k\|^2]\end{aligned}$$

3.
$$\begin{aligned}(h^\star - g^\star)(y^{k+1}) &\leq (g-h)(x^{k+1}) - \tfrac{\rho_1^\star}{2}\|dy^k\|^2 \\ &\leq (h^\star - g^\star)(y^k) - [\tfrac{\rho_1^\star}{2}\|dy^k\|^2 + \tfrac{\rho_2^\star}{2}\|dx^k\|^2]\end{aligned}$$

4.
$$\begin{aligned}(h^\star - g^\star)(y^{k+1}) &\leq (g-h)(x^{k+1}) - \tfrac{\rho_1}{2}\|dy^{k+1}\|^2 \\ &\leq (h^\star - g^\star)(y^k) - [\tfrac{\rho_1}{2}\|dx^{k+1}\|^2 + \tfrac{\rho_2}{2}\|dx^k\|^2]\end{aligned}$$

Corollaire 1.6 *([5]) If the equalities take place, we have:*

1. $(g-h)(x^{k+1}) = (h^\star - g^\star)(y^k) \Longleftrightarrow y^k \in \partial h(x^{k+1})$
2. $(g-h)(x^{k+1}) = (g-h)(x^k) \Longleftrightarrow x^k \in \partial g^\star(y^k), \quad y^k \in \partial h(x^{k+1})$
3. $(h^\star - g^\star)(y^k) = (g-h)(x^k) \Longleftrightarrow x^k \in \partial g^\star(y^k)$
4. $(h^\star - g^\star)(y^{k+1}) = (h^\star - g^\star)(y^k) \Longleftrightarrow y^k \in \partial h(x^{k+1}), \quad x^{k+1} \in \partial g^\star(y^{k+1})$

In general, the qualities (robustness, stability, convergence rate, good local solutions) of DCA depend on DC decomposition of the objective function $f = g - h$. Theorem 1.4 shows that the strong convexity of convex components in primal and dual problems can affect DCA. To make convex components g and h strongly convex, we can usually apply the following operation

$$f = g - h = \left(g + \frac{\lambda}{2}\|.\|^2\right) - \left(h + \frac{\lambda}{2}\|.\|^2\right).$$

In this case, the convex components in the dual problem will be continuously differentiable.

Computation of subgradients

The description of DCA iteration schemes uses fixed points of multi-applications ∂h and $\partial g^\star$ (∂g and $\partial h^\star$), this can be expressed as follows:

$$\begin{array}{lcl} x^k & \leftarrow & y^k \in \partial h(x^k) \\ & \swarrow & \\ x^{k+1} \in \partial g^\star(y^k) & \leftarrow & y^{k+1} \in \partial h(x^{k+1}) \\ (y^k \in \partial g(x^{k+1})) & & (x^{k+1} \in \partial h^\star(y^{k+1})) \end{array} \tag{1.15}$$

We see a perfect symmetry of the action of two sequences $\{x^k\}$ and $\{y^k\}$ on the dual DC optimization.

The calculation of the subgradient of the function h at a point x^k is usually easy, in many practical problems we know the explicit expression of ∂h. In contrast, the calculation of a subgradient in the conjugate of the convex function g at point y^k usually requires solving the convex program,

$$\partial g^\star(y^k) = argmin\{g(x) - \langle y^k, x\rangle : x \in X\}. \tag{1.16}$$

Indeed, recall that the explicit expression of the conjugate of a given function, in practice, is not known. From (1.16), note that the computation of x^{k+1} is equivalent to minimizing a convex function derived from DC function $f = g - h$, by approximating the concave component $-h$ by its affine minorization at the point x^k, i.e.

$$x^{k+1} \in \partial g^\star(y^k): \quad x^{k+1} \in argmin\{g(x) - \Big[\langle y^k, x - x^k\rangle + h(x^k)\Big] : x \in X\}.$$

And similarly, by duality:

$$y^{k+1} \in \partial h(x^{k+1}): \quad y^{k+1} \in argmin\{h^\star(y) - \Big[\langle x^{k+1}, y - y^k\rangle + g^\star(y^k)\Big] : y \in Y\}.$$

Polyhedral DC optimization

Polyhedral DC optimization occurs when one convex component g or h is polyhedral convex. Like the polyhedral convex optimization problems, this class of problems of DC optimization is frequently encountered in practice and has interesting properties. We will see that the description of DCA is particularly simple ([3, 4, 5]).

Consider a DC program

$$\inf\{g(x) - h(x) : x \in X\} \quad (P).$$

When the convex component h is polyhedral, i.e.

$$h(x) = \max_{x \in X}\{\langle a^i, x\rangle - b^i : i = 1, ..., m\},$$

then the calculation of the subgradients $y^k = \partial h(x^k)$ is explicit. It is clear that limiting (naturally) the choice of subgradients or gradients of affine minorization function h, i.e. $\{y^k\} \in \{a^i : i = 1, ..., m\}$, which is a finite set, following the iteration $\{y^k\}$ is finite ($k \leq m$). Indeed, the sequence $\{(h^\star - g^\star)(y^k)\}$ is, by construction of DCA, decreasing and the choices of iterations y^k are finite. Similarly, by duality the sequence $\{x^k\}$ is finite and $\{(g - h)(x^k)\}$ is decreasing.

Theorem 1.5 ***(Finite convergence)***

- *the sequences* $\{g(x^k) - h(x^k)\}$ *and* $\{h^\star(y^k) - g^\star(y^k)\}$ *are decreasing,*
- *when* $(g - h)(x^{k+1}) = (g - h)(x^k)$ *then the algorithm stops at* $(k + 1)^{th}$ *iteration and a point* x^k *(resp.* y^k*) is a critical point of* $g - h$ *(resp.* $h^\star - g^\star$*).*

Note that if the convex component g is polyhedral, the conjugate function g^* is polyhedral too and writing the dual problem, we found the same results as above.

DCA interpretations

At each iteration of DCA, we replace the second component h in primal DC program by its affine minorization $h_k(x) = h(x^k) + \langle x - x^k, y^k \rangle$ in a neighborhood of x^k to obtain the convex program following

$$\inf\{\overline{f}_k = g(x) - h_k(x) : x \in \mathbb{R}^n\} \tag{1.17}$$

whose set of optimal solutions is $\partial g^\star(y^k)$.

Similarly, the second DC component $g^\star$ in dual DC program (1.6) is replaced by its affine minorization $(g^\star)_k(y) = g^\star(y^k) + \langle y - y^k, x^{k+1} \rangle$ in a neighborhood of y^k to give birth to the convex program

$$\inf\{h^\star(y) - (g^\star)_k(y) : y \in \mathbb{R}^n\} \tag{1.18}$$

which $\partial h(x^{k+1})$ is the set of optimal solutions. It should be noted that DCA works with DC components g and h and not with the function f itself. Each DC decomposition of f gives rise to a DCA.

Such as $\overline{f}_k$ is a convex function, the minimizer x^{k+1} is defined by $0 \in \partial \overline{f}_k(x^{k+1})$ and the minorization of f by $\overline{f}_k$ ensures the decreasing of the sequence $\{f(x^k)\}$. Indeed, as h_k is a affine minorization function of h at x_k, $\overline{f}_k$ is a minorization convex function of f,

$$f(x) \leq \overline{f}_k(x), \quad \forall x \in X,$$

which coincides at x^k with f,

$$f(x^k) = \overline{f}(x^k).$$

Therefore determining the iteration x^{k+1} as the minimum convex program (1.17), the decreasing of the sequence of iterations is endured,

$$f(x^{k+1}) \leq f(x^k).$$

If at the iteration $k+1$, $f(x^{k+1}) = f(x^k)$ then x^{k+1} is a critical point of f ($0 \in \partial \overline{f}_k(x^{k+1}) \Longrightarrow 0 \in \partial f(x^{k+1})$).

Theorem 1.5 *If $\overline{f}_k$ is strictly convex then there exists a unique minimizer x^{k+1}.*

Comment: It is important to note that the function f is replaced by function $\overline{f}_k$ in overall domain of f,

$$\overline{f}_k(x) = g(x) - (\langle y^k, x - x^k \rangle + h(x^k)) \quad \text{with} \quad y^k \in \partial h(x^k), \forall x \in X$$

which, considered locally in the neighborhood of x^k, is a first order approximation of f and globally on $\mathbb{R}^n$. It is noteworthy that $\overline{f}_k$ is not defined narrowly from local information of f at neighborhood of x^k (i.e. $f(x^k), \partial f(x^k)$,...) but incorporates all of the first convex components of f in its definition, i.e. $\overline{f}_k = g - h_k = f - (h + h_k)$. In other words, $\overline{f}_k$ is not simply a local approximation of f in a neighborhood of x^k, but should rather be described as "convexification majorant" of f globally related to DC function by the first convex component defined on $\mathbb{R}^n$. Therefore, no displacement of the x^k to x^{k+1} is determined from f globally defined for all $x \in \mathbb{R}^n$. DCA can not be simply regarded as a local approximation method or a local descent method in classic, the global characteristic is the

"convexification majorant". Thus, unlike conventional local approaches (deterministic or heuristic), DCA operates simultaneously local and global properties of the function to be minimized during the iterative process and in practice converges to a good solution local and sometimes global.

For a comprehensive study of DC programming and DCA, refer to [3]-[6] and [11]-[13] and the reference therein. The treatment of a nonconvex problem by DC approach and DCA should have two tasks: looking for an appropriate DC decomposition and looking for a good starting point.

For a DC program given, the issue of finding a good DC decomposition remains open, in practice, we look for a DC decomposition to adapt with the structure of the DC program for which the studied sequences $\{x^k\}$ and $\{y^k\}$ are easy to calculate. If the calculation is explicit then the corresponding DCA is less expensive time and it is able to support very large dimensions.

1.3 Conclusion

Section 1.2 introduces all the prerequisite notions and definitions concerning DC programming and DCA. The fundamentals of DC analysis, such as Polyhedral functions, DC functions, DC duality, global and local optimality in DC optimization are presented. After that, the presentation focuses on DCA is introduced, with the principle, the computation and the interpretations of DCA.

References

[1] Auslender, A.: Optimisation Méthodes Numériques. Paris: Masson, 1976.

[2] Laurent, P.J.: Approximation et optimisation. Paris: Hermann, 1972.

[3] Le Thi, H.A.: Analyse numérique des algorithmes de l'optimisation DC. Approches locale et globale. Codes et simulations numériques en grande dimension. Applications. Thèse de Doctorat, Université de Rouen, 1994.

[4] Le Thi, H.A.: Contribution à l'optimisation non convexe et l'optimisation globale: Théorie, Algorithmes et Applications. Habilitation à Diriger des Recherches, Université de Rouen, 1997.

[5] Le Thi, H.A., Pham Dinh, T.: Solving a class of linearly constrained indefinite quadratic problems by DC algorithms. Journal of Global Optimization 11, pp. 253–285, 1997.

[6] Le Thi, H.A., Pham Dinh, T., Nguyen Van, T.: Combination between Local and Global Methods for Solving an Optimization Problem over the Efficient Set. European Journal of Operational Research 142, pp. 257–270, 2002.

[7] Le Thi, H.A., Pham Dinh, T.: DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems. Ann. Oper. Res. Springer-Verlag 133, pp. 23–46, 2005.

[8] Pham Dinh, T.: Elements homoduaux relatifs à un couple de normes (φ, ψ). Applications au calcul de $S_{\varphi\psi}(A)$". Technical Report, Grenoble, 1975.

[9] Pham Dinh, T.: Calcul du maximum d'une forme quadratique définie positive sur la boule unité de la norme du max. Technical Report, Grenoble, 1976.

[10] Pham Dinh, T.: Algorithms for solving a class of non convex optimization problems. Methods of subgradients. Fermat days 85. Mathematics for Optimization, Elsevier Science Publishers B.V. North-Holland, 1986.

[11] Pham Dinh, T.: Duality in DC (difference of convex functions) optimization. Subgradient methods. Trends in Mathematical Optimization, International Series of Numer Math, volume 84, pp. 277–293, 1988.

[12] Pham, D.T., Le Thi, H.A.: Convex analysis approach to d.c. programming: Theory, Algorithm and Applications. Acta Mathematica Vietnamica 22, pp. 289–355, 1997.

[13] Pham Dinh, T., Le Thi, H.A.: DC optimization algorithms for solving the trust region sub-problem. SIAM J. Optim. 8, pp. 476–505, 1998.

[14] Pham Dinh, T., Le Thi, H.A.: Recent advances in DC programming and DCA. Transactions on Computational Collective Intelligence, Volume 8342, pp. 1–37, 2014.

[15] Reiner Horst , Hoang Tuy: Global Optimization: Deterministic Approaches. Springer, 1996.

[16] Rockafellar, R.T.: Convex Analysis. Princeton University Press, Princeton, 1970.

[17] Tuy, H.: DC Optimisation : Theory, Methods and Algorithms, Handbook of Global Optimisation. Horst and Pardalos eds, Klaver Academic Publishers, pp. 149–216, 1995.

[18] Toland, J.F.: Direct Calculation of the Information Matrix via the EM Algorithm. Journal of Mathematical Analysis and Applications 66, pp. 399–415, 1978.

[19] Urruty, J.B.H.: Generalized differentiability, duality and optimization for problem dealing with differences of convex functions. Lecture Notes in Economics and Mathematical Systems, volume 256, pp. 260–277, Springer Verlag, 1985.

[20] Urruty, J.B.H.: Conditions nécessaires et suffisantes d'optimalité globale en optimisation de différences de deux fonctions convexes, pp. 459–462. I. CRAS, 1989.

Chapter 2

Modularity maximization in network and application to community detection

2.1 DC programming approach

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

A DC programming approach for finding Communities in networks*

Hoai An LE THI · Manh Cuong NGUYEN · Tao PHAM DINH

Submitted version to the Neural Computation journal (NECO).
Received: date / Accepted: date

Abstract Automatic discovery of community structures in complex networks is a fundamental task in many disciplines, including physics, biology and social science. The most used criterion for characterizing the existence of a community structure in a network is modularity, a quantitative measure proposed by Newman and Girvan [Physical Review E 69, 026113 (2004)]. The discovery community can be formulated as the so called modularity maximization problem that consists of finding a partition of nodes of a network with the highest modularity. In this article, we propose a fast and scalable algorithm, called DCAM, based on DC (Difference of Convex function) programming and DCA (DC Algorithms), an innovative approach in nonconvex programming framework for solving the modularity maximization problem. The special structure of the considered problem has been well exploited to get an inexpensive DCA scheme that requires only matrix-vector product at each iteration. Starting with a very large number of communities, DCAM furnishes, as output results, an optimal partition together with the optimal number of communities c^*, i.e., the number of communities is discovered automatically during DCAM's iterations. Numerical experiments are performed on a variety of real-world network datasets with up to $4,194,304$ nodes and $30,359,198$ edges. The comparative results with six reference algorithms show that the proposed approach outperforms them not only on quality and rapidity but also on scalability. Moreover, it realizes a very good trade-off between the quality of solutions and the runtime.

Keywords Networks, community, modularity, DC programming, DCA

1 Introduction

In recent years, the study of complex networks has attracted a great deal of interest in many disciplines, including physics, biology and social science. Examples of such networks include Web graphs, social networks, citation networks, biochemical networks. Despite the fact that these networks belong to very distinct fields, they all surprisingly share some common structural features, such as the power law degree distributions, small-world average shortest paths, high clustering coefficients.

* This research has been suported by "Fonds Européens de Développement Régional" (FEDER) Lorraine via the project "**Inno**vations techniques d'optimisation pour le traitement **Ma**ssif de **D**onnées"(InnoMaD)

Hoai An LE THI · Manh Cuong NGUYEN
Laboratory of Theoretical and Applied Computer Science
University of Lorraine, Ile du Saulcy, 57045 Metz, France.
E-mail: hoai-an.le-thi@univ-lorraine.fr, manh-cuong.nguyen@univ-lorraine.fr

Tao PHAM DINH
Laboratory of Mathematics. National Institute for Applied Sciences - Rouen
76801 Saint-Etienne-du-Rouvray Cedex, France. E-mail: pham@insa-rouen.fr

Community structure is one of the important network features. The whole network is in fact composed of densely connected sub-networks, with only sparser connections between them. Such sub-networks are called communities or modules. Detection of communities is of significant practical importance as it allows to analyze networks at a megascopic scale: identification of related web pages in the WWW, uncovering of communities in social networks, decomposition of metabolic networks in functional modules.

If there exists community structure in a network, the intra-community edges should be significantly denser than the inter-community edges. Hence, detecting communities amounts to searching for the structure that maximizes the number of intra-community edges while minimizing the number of inter-community edges. In 2004, Girvan and Newman [46,47] has proposed a quantitative measure, called the modularity (Q measure), for characterizing the existence of community structure in a network. The modularity Q of a particular partition is defined as the number of edges inside clusters, minus the expected number of such edges if the graph were random conditioned on its degree distribution.

More formally, consider an undirected unweighted network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes ($\mathcal{V} = \{1, \ldots, n\}$), and m edges ($m = Card(\mathcal{E})$). Denote by A the adjacency matrix:

$$A_{ij} = \begin{cases} 1, \ if \ (i,j) \in \mathcal{E} \\ 0, \ \text{otherwise.} \end{cases}$$

The degree of node i is denoted ω_i ($\omega_i = \sum_{j=1}^{n} A_{ij}$), and ω stands for the vector whose components are ω_i.

Let $\mathcal{P}$ be a partition of $\mathcal{V}$ and let $\delta(i,j)$ be a function which takes value 1 if nodes i, j are in the same community, and 0 otherwise. The modularity measure is defined as follows:

$$Q(\mathcal{P}) = \frac{1}{2m} \sum_{i,j=1}^{n} \left(A_{ij} - \frac{\omega_i \omega_j}{2m} \right) \delta(i,j).$$

The fist part of $Q(\mathcal{P})$, say $\frac{1}{2m} \sum_{i,j=1}^{n} A_{ij} \delta(i,j)$, corresponds to the fraction of intra-community links, whereas the second part $\frac{1}{2m} \sum_{i,j=1}^{n} \frac{\omega_i \omega_j}{2m} \delta(i,j)$ corresponds to the same fraction in a random network. The modularity values range from -0.5 to 1 ([8]), with a higher positive value indicating a stronger support for community structure in the network. Hence, community discovery can be formulated as the so called modularity maximization problem that consists of finding a partition of nodes of a network with the highest modularity. Since its introduction, the modularity measure becomes a central tool for network analysis, many works have studied its properties (see e.g. [12,19,3,8]), and numerous algorithms have been developed to maximize modularity. It has been proved in [8] that the modularity maximization is a NP-hard problem. Hence exact methods are computationally intractable even for very small networks, and most existing methods are approximate or heuristic.

In terms of clustering, the approaches differ in whether or not a hierarchical partition (recursively subdividing communities into sub-communities) is sought, whether the number of communities is pre-specified by the user or decided by the algorithm, as well as other parameters. In this sense, three classes of algorithms can be distinguished: divisive hierarchical clustering (tackle multiple 2-partition problems repeatedly, [1,16,45]), agglomerative hierarchical clustering (merge clusters repeatedly, [5,10,43,52,63]) and k-way partitioning ([17,21,37,44]). On another hand, it has been shown in [64] that modularity-based clustering can be understood as a special instance of spectral clustering. Methods based on spectral clustering are developed in [44,64,54,59].

In terms of optimization, one distinguishes three mains approaches for modularity maximization: heuristics, meta heuristics, and mathematical programming approaches. Numerous heuristics are developed, among of others [5,10,16,17,43–45,52,9]. For meta heuristics, one can cite simulated annealing [40,21,41], genetic search [60], greedy algorithms ([63,55]). The mathematical programming approaches are more recent. In these approaches the modularity maximization is formulated as either an integer (or mixed integer) linear programming problem or an integer / mixed integer quadratic program. Due to the NP-hardness of these problems, most

of proposed methods are approximate. In [1] the authors consider an integer linear programming formulation and introduced two approximate algorithms, the first is linear programming followed by randomized rounding, and the second is based on a vector programming relaxation of a quadratic program which recursively splits one partition into two smaller partitions while a better modularity can be obtained. In [17], an integer quadratic formulation is considered and an iterative algorithm is proposed which solves a quadratic program at each iteration. There are few exact algorithms, the first is introduced in [65] and the most recent is based on column generation approaches for mixed integer linear / quadratic programming ([2]). The largest problem solved to date by exact methods has 512 entities ([2]).

Very recently, new modularity measures based on the modularity Q and the modularity-density D-measure [38], and then new formulations of modularity maximization via mixed integer nonlinear programming were investigated in [23,27]. These approaches aim to identify overlapping communities or small communities which can not be revealed by the original modularity maximization problem.

In this paper, we focus on maximizing the original modularity Q by a mathematical programming approach via an integer quadratic programming problem. Our method is based on DC (Difference of Convex functions) programming and DCA (DC Algorithms), an innovative continuous approach in nonconvex programming framework. DC programming and DCA were introduced by Pham Dinh Tao in their preliminary form in 1985 and have been extensively developed since 1994 by Le Thi Hoai An and Pham Dinh Tao and become now classic and increasingly popular (see e.g. [11,30–36,48–51,56,58] and the list of references in [28]). They address the DC programs of the form

$$\alpha = \inf\{f(x) := g(x) - h(x) : x \in \mathbb{R}^p\} \quad (P_{dc}) \tag{1}$$

where $g, h : \mathbb{R}^p \to \mathbb{R} \cup \{+\infty\}$ are lower semicontinuous proper convex functions on $\mathbb{R}^p$. Such a function f is called DC function, and $g - h$, DC decomposition of f while g and h are DC components of f. The construction of DCA involves DC components g and h but not the 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 important from algorithmic point of views. Moreover, despite its local character, DCA with a good initial point can converge to global solutions. Finding a "good" initial point is then also an important stage of DCA. How to develop an efficient algorithm based on the generic DCA scheme for a practical problem is thus a judicious question to be studied, and the answer depends on the specific structure of the problem being considered.

Our work is 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 ([11,30–36,48,49,56–58]) for which they provided quite often a global solution and proved to be more robust and efficient than standard methods. Working on the matrix space, we first formulate the modularity maximization problem as maximizing a quadratic function under the cartesian product of unit simplices with binary variables. This problem is then equivalently reformulated as maximizing a quadratic function under the same set but now with continuous variables on which DCA can be applied. We propose an appropriate DC decomposition that gives rise, after a suitable computational strategy, to a very simple DCA scheme (called DCAM) in which all computations are explicit. The advantages of our algorithm are multiple:

– Thanks to DC decomposition technique, the initial combinatorial optimization problem is transformed equivalently, in an elegant way, to a continuous problem. Surprisingly, due to this customized choice of DC decomposition, although our algorithm works on a continuous domain, it constructs a sequence in the discrete feasible set of the initial problem. Such an original property is important for large scale setting: in ultra large networks, if we stop the algorithm before its convergence, we get always an integer solution, i.e. the algorithm furnishes an "approximate" solution without rounding procedures.

– Again thanks to a good choice of DC decomposition, DCAM enjoys interesting convergence properties: it converges, after a finitely many iterations, to a local solution in almost cases. In particular, DCAM is very inexpensive in terms of CPU time since all computations are explicit. In fact, each iteration requires only one matrix-vector product and maximizes a linear function on a simplex whose solutions are explicitly computed (no solver is required).
– Although the number of cluster is an input parameter of the algorithm, it can be modified by DCAM itself during its iterations. Starting with a very large number of clusters (even with n, the number of nodes of the network), DCAM can detect empty clusters and provide an optimal partition together with the optimal number of communities. That nice feature constitutes a great advantage of DCAM: finding the number of clusters is a difficult and still relevant issue for researchers in clustering analysis.

These benefits are approved through our numerical results on read-world networks: DCA outperforms standard approaches on both solution quality and computation cost, and also on scalability.

The rest of the paper is organized as follows. In section 2, we present the integer quadratic formulation of the modularity maximization problem. Section 3 is devoted to DC programming and DCA for solving this quadratic program. First, we give a brief presentation of DC programming and DCA and then present reformulation techniques as well as a DC formulation of the considered problem. Afterwards we show how to determine the resulting DCA scheme and provide some discussions about the algorithm. Computational experiments are reported in Section 4 and finally Section 5 concludes the paper.

2 An integer quadratic formulation of the modularity maximization problem

Assume that each vertex (entity) belongs to exactly one community, i.e., the case of overlapping communities is not considered here.

Let $\mathcal{P}$ be a partition of $\mathcal{V}$ and let c be the number of communities in $\mathcal{P}$. Define the binary assignment matrix $U = (U_{ik})_{i=1,\dots,n}^{k=1,\dots,c}$ in $\mathcal{P}$, say $U_{ik} = 1$ if the vertex i belongs to the community k and 0 otherwise. Then the modularity can also be expressed according to U as follows:

$$Q(U) = \frac{1}{2m} \sum_{i,j=1}^{n} B_{ij} \sum_{k=1}^{c} U_{ik} U_{jk},$$

where $B := A - \frac{1}{2m}\, \omega\omega^T$ is a constant matrix, called the modularity matrix. It depends only on the network (independent on $\mathcal{P}$). Hence the modularity maximization problem can be written as

$$\begin{aligned} \max_{U} Q(U) := \quad & \tfrac{1}{2m} \textstyle\sum_{i,j=1}^{n} B_{ij} \sum_{k=1}^{c} U_{ik} U_{jk} && (2) \\ s.t \quad & \\ & \textstyle\sum_{k=1}^{c} U_{ik} = 1 \ \forall i = 1, ..., n && (3) \\ & U_{ik} \in \{0, 1\} \forall i = 1, ..., n; \forall k = 1, ..., c. && (4) \end{aligned}$$

The constraint (3) ensure that each entity belongs to exactly one community. Observe that maximizing modularity gives an optimal partition together with the optimal number of communities c.

For a matrix $U \in \mathbb{R}^{n.c}$, $U_{i.}$ and $U_{.j}$ denote the i^{th} row and the j^{th} column of U respectively. The transpose of U is denoted by U^T, and $(U_i)^T$ will be written as U_i^T for simplicity.

Let $\mathcal{M}_{n,c}(\mathbb{R})$ denote the space of real matrices of order $n \times c$. We can identify by rows (resp. columns) each matrix $U \in \mathcal{M}_{n,c}(\mathbb{R})$ with a row-vector (resp. column-vector) in $(\mathbb{R}^c)^n$ (resp. $(\mathbb{R}^n)^c$) by writing respectively

$$U \longleftrightarrow \mathfrak{U} = (U_{1.}, \dots, U_{n.}), U_{i.}^T \in \mathbb{R}^c,\ \mathfrak{U}^{\mathrm{T}} \in (\mathbb{R}^{\mathrm{c}})^{\mathrm{n}}, \quad (5)$$

and

$$U \longleftrightarrow \mathcal{U} = \begin{pmatrix} U_{.1} \\ \vdots \\ U_{.c} \end{pmatrix},\ U_{.j} \in \mathbb{R}^n, \mathcal{U} \in (\mathbb{R}^n)^c. \tag{6}$$

The inner product in $\mathcal{M}_{n,c}(\mathbb{R})$ is defined as the inner product in $(\mathbb{R}^c)^n$ or $(\mathbb{R}^n)^c$. That is

$$\langle X, Y\rangle_{\mathcal{M}_{n,c}(\mathbb{R})} = \langle \mathfrak{X}^T, \mathfrak{Y}^T\rangle_{(\mathbb{R}^c)^n} = \langle \mathcal{X}, \mathcal{Y}\rangle_{(\mathbb{R}^n)^c} = Tr(X^T Y),$$

where $Tr(X^T Y)$ denotes the trace of the matrix $X^T Y$. Hence the modularity measure can be computed as

$$Q(U) = \frac{1}{2m} Tr(U^T B U) = \frac{1}{2m} \sum_{j=1}^{c} (U_{.j})^T B U_{.j} = \frac{1}{2m} \sum_{i=1}^{n} U_{i.}([(BU)]_{i.})^T. \tag{7}$$

In the sequel, for simplifying computations, we shall choose either representation (matrix or vector) of U in a convenient way. For instance, with the column identification of U we can express the modularity measure via vector representation as

$$Q(\mathcal{U}) = \frac{1}{2m} \mathcal{U}^T B_b \, \mathcal{U},$$

where B_b is the diagonal block matrix of order $n.c \times n.c$ given by:

$$B_b := \begin{pmatrix} B & & & \\ & B & & \\ & & \ddots & \\ & & & B \end{pmatrix}.$$

Let Δ_c be the $(c-1)-$simplex defined as

$$\Delta_c = \{x \in [0,1]^c : \sum_{k=1}^{c} x_k = 1\}$$

and let $\Delta \subset [0,1]^{n*c}$ be the Cartesian product of n simplices Δ_c, say

$$\Delta := \Delta_c \times ... \times \Delta_c.$$

Denote by $V(\Delta_c)$ (resp. $V(\Delta)$) the vertex set of Δ_c (resp. Δ). It is easy to see that

$$V(\Delta) = V(\Delta_c) \times ... \times V(\Delta_c).$$

With the matrix representation, the problem (2) is written as:

$$\max \left\{ \frac{1}{2m} Tr(U^T B U) : U_{i.} \in V(\Delta_c), \forall i = 1, \ldots, n \right\},$$

and with the vector column representation it is expressed as:

$$\max \left\{ Q(\mathcal{U}) := \frac{1}{2m} \mathcal{U}^T B_b \mathcal{U} : \mathcal{U} \in V(\Delta) \right\}. \tag{8}$$

In the next section we will develop DC programming and DCA for solving the modularity maximization problem of the form (8).

3 Solving the modularity maximization problem by DC programming and DCA

3.1 A brief introduction of DC programming and DCA

As indicated in the introduction, DC programming and DCA address the problem of minimizing a DC function on the whole space $\mathbb{R}^p$ or on a closed convex set $C \in \mathbb{R}^p$. Generally speaking, a DC program takes the form:

$$\alpha = \inf\{f(x) := g(x) - h(x) : x \in \mathbb{R}^p\}, \quad (P_{dc}) \tag{9}$$

where $g, h : \mathbb{R}^p \to \mathbb{R} \cup \{+\infty\}$ are lower semi-continuous proper convex functions on the Euclidean space $\boldsymbol{X} := \mathbb{R}^p$. When either g or h is a polyhedral convex function (say, a pointwise supremum of a finite collection of affine functions) (P_{dc}) is called a polyhedral DC program.

A convex constraint $x \in C$ can be incorporated in the objective function f by using the indicator function on C, denoted χ_C, which is defined by $\chi_C(x) = 0$ if $x \in C$, $+\infty$ otherwise. By the way, any convex constrained DC program can be written in the standard form (P_{dc}) by adding the indicator function of the convex constraint set to the first DC component g.

Let $y \in \boldsymbol{Y}$, where $\boldsymbol{Y}$ is the dual space of $\boldsymbol{X} = \mathbb{R}^p$ that can be identified with $\mathbb{R}^p$ itself. Let $g^*(y)$ defined by:

$$g^*(y) = \sup\{\langle x, y\rangle - g(x) \ : \ x \in \mathbb{R}^p\}$$

be the conjugate function of g, then, the following program is called the dual program of (P_{dc}):

$$\alpha_D = \inf\{h^*(y) - g^*(y) : y \in \boldsymbol{Y}\}. \quad (D_{dc})$$

One can prove that $\alpha = \alpha_D$ (see e.g. [29,50]) and there is a perfect symmetry between primal and dual DC programs: the dual of (D_{dc}) is exactly (P_{dc}).

For a convex function h, the subdifferential of h at x_0, denoted by $\partial h(x_0)$, is defined by

$$\partial h(x_0) \equiv \{y \in \mathbb{R}^p : h(x) \geq h(x_0) + \ \langle x - x_0, y\rangle, \forall x \in \mathbb{R}^p\}.$$

The sub-differential $\partial h(x_0)$ is a closed convex set which generalizes the derivative in the sense that h is differentiable at x_0 if and only if $\partial h(x_0) \equiv \{\nabla_x h(x_0)\}$.

DCA is based on the local optimality conditions for (P_{dc}), namely

$$\partial h(x^*) \cap \partial g(x^*) \neq \emptyset \tag{10}$$

(such a point x^* is called *critical point* of $g - h$ or generalized KKT point for (P_{dc})), and

$$\emptyset \neq \partial h(x^*) \subset \partial g(x^*). \tag{11}$$

The condition (11) is necessary local optimality for (P_{dc}). It is also sufficient for many classes of DC programs quite often encountered in practice. In particular it is sufficient for polyhedral DC programs.

The transportation of global solutions between (P_{dc}) and (D_{dc}) is expressed by the following propositions:

Property 1

$$[\cup_{y^* \in \mathcal{D}} \partial g^*(y^*)] \subset \mathcal{P}, \ \ [\cup_{x^* \in \mathcal{P}} \partial h(x^*)] \subset \mathcal{D}, \tag{12}$$

where $\mathcal{P}$ and $\mathcal{D}$ denote the solution sets of (P_{dc}) and (D_{dc}) respectively.

Under technical conditions, this transportation holds also for local solutions of (P_{dc}) and (D_{dc}) (see [29,50] for more details).

Property 2 *Let x^* be a local solution to (P_{dc}) and let $y^* \in \partial h(x^*)$. If g^* is differentiable at y^* then y^* is a local solution to (D_{dc}). Similarly, let y^* be a local solution to (D_{dc}) and let $x^* \in \partial g^*(y^*)$. If h is differentiable at x^* then x^* is a local solution to (P_{dc}).*

The main idea of DCA is simple: each iteration of DCA approximates the convex function h by its affine minorant defined by $y^k \in \partial h(x^k)$, and solves the resulting convex program.

DCA - general scheme
initializations: let $x^0 \in \mathbb{R}^p$ be a guess, set $k := 0$.

repeat
1. calculate $y^k \in \partial h(x^k)$.
2. calculate $x^{k+1} \in \arg\min\{g(x) - \langle x, y^k\rangle : x \in \mathbb{R}^p\}$ (P_k).
3. $k = k + 1$.

until convergence of $\left\{x^k\right\}$.

Convergence properties of DCA and its theoretical basis can be found in [29, 50, 51]. For instance, it is important to mention that (for simplicity, we omit here the dual part of these properties)

- DCA is a descent method without linesearch: the sequence $\{g(x^k) - h(x^k)$ is decreasing. If $g(x^{k+1}) - h(x^{k+1}) = g(x^k) - h(x^k)$ then x^k is a critical point of $g - h$.
- If the optimal value α of problem (P_{dc}) is finite and the infinite sequence $\{x^k\}$ is bounded, then every limit point x^* of the sequence $\{x^k\}$ is a critical point of $g - h$.In such a case, DCA terminates at k-th iteration.
- DCA has a *linear convergence* for DC programs. Especially, for polyhedral DC programs the sequence $\{x^k\}$ contains finitely many elements and the algorithm converges after a finite number of iterations.

For a complete study of DC programming and DCA the reader is referred to [29, 50, 51] and the references therein. The solution of a nonconvex program (P_{dc}) by DCA must be composed of two stages: the search of an *appropriate* DC decomposition of f and that of a *good* initial point. In the last years research is very active on the use of DC programming and DCA for Machine Learning and Data Mining.

It should be noted that

i) the convex concave procedure (CCCP) for constructing discrete time dynamical systems mentioned in [57] is a special case of DCA applied to smooth optimization;

ii) the SLA (Successive Linear Approximation) algorithm developed in [7] is a version of DCA for concave minimization;

iii) the EM algorithm [13] applied to the log-linear model is a special case of DCA.

We show below how to use DCA for solving the equivalent modularity maximization problem (8).

3.2 DC programming and DCA for solving problem (8)

Both column and row identifications displayed in Section 2 will be exploited here in order to provide an explicit DCA which could handle large-size problems.

3.2.1 A continuous reformulation of problem (8)

Using a DC decomposition technique we first reformulate (8) as a continuous optimization problem.

For any real number μ, the modularity measure can be rewritten as:

$$Q(\mathcal{U}) = \frac{1}{2m}\mathcal{U}^T(B_b + \mu I_b)\mathcal{U} - \frac{1}{2m}\mu\mathcal{U}^T\mathcal{U},$$

where I_b is the identity matrix of order $n.c.$

Let h be the quadratic function defined by $h(\mathcal{U}) := \frac{1}{2}\mathcal{U}^T(B_b + \mu I_b)\mathcal{U}$. As $\mathcal{U} \in V(\Delta)$, we have

$$\mathcal{U}^T\mathcal{U} = Tr(U^TU) = \sum_{i=1}^{n} U_{i.}(U_{i.})^T = n.$$

Therefore the problem (8) is equivalent to

$$\max\left\{h(\mathcal{U}) := \frac{1}{2}\mathcal{U}^T(B_b + \mu I_b)\mathcal{U} : \mathcal{U} \in V(\Delta)\right\}. \tag{13}$$

Let μ be a scalar such that $\mu > -\lambda_1(B)$, where $\lambda_1(B)$ is the smallest eigenvalue of the modularity matrix B. Hence the quadratic function h is strongly convex and we call μ the regularization parameter. As h is strongly convex, (13) and the convex maximization problem

$$\max\{h(\mathcal{U}) : \mathcal{U} \in \Delta\} \tag{14}$$

are equivalent in the sense that they have the same optimal value and the same solution set.

3.2.2 A DC formulation of problem (14)

In the sequel we consider the problem (14) with $\mu > -\lambda_1(B)$. This convex maximization problem (14) can be written as

$$\min\{-h(\mathcal{U}) : \mathcal{U} \in \Delta\}$$

which is equivalent to

$$\min\left\{f(\mathcal{U}) := \chi_\Delta(\mathcal{U}) - h(\mathcal{U}) : \mathcal{U} \in \mathbb{R}^{n.c}\right\}, \tag{15}$$

where χ_Δ is the indicator function on Δ. Clearly, the function χ_Δ is convex (because that Δ is a convex set), and then (15) is a DC program with the following natural DC decomposition

$$f(\mathcal{U}) := g(\mathcal{U}) - h(\mathcal{U}),\ g(\mathcal{U}) = \chi_\Delta(\mathcal{U}).$$

Since χ_Δ is a polyhedral function, (15) is a polyhedral DC program. Moreover, h is differentiable and its gradient is given by $\nabla h(\mathcal{U}) = (B_b + \mu I_b)\mathcal{U}$.

3.2.3 The DCA scheme corresponding to (15)

According to the generic DCA scheme, applying DCA to the problem (15) consists of computing, at each iteration k, $\mathcal{Y}^k = \nabla h(\mathcal{U}^k) = (B_b + \mu I_b)\mathcal{U}^k$ and then solving the next problem to obtain $\mathcal{U}^{k+1}$:

$$\min\left\{\chi_\Delta(\mathcal{U}) - \langle \mathcal{U}, \mathcal{Y}^k\rangle : \mathcal{U} \in \mathbb{R}^{n.c}\right\}). \tag{16}$$

Denote by $Y^k = (B+\mu I)U^k$ (I denotes the identity matrix of order n). Now, taking into account the useful properties concerning matrix representation of $\mathcal{U}$ and $\mathcal{Y}^k$, we can compute explicitly an optimal solution of (16) in the following way:

$$\min\left\{\chi_\Delta(\mathcal{U}) - \langle \mathcal{U}, \mathcal{Y}^k\rangle : \mathcal{U} \in \mathbb{R}^{n.c}\right\} \Longleftrightarrow \max\left\{\langle \mathcal{U}, \mathcal{Y}^k\rangle : \mathcal{U} \in \Delta\right\} \tag{17}$$

$$\begin{aligned} &= \max\left\{\sum_{i=1}^{n}\langle U_{i.}, Y_{i.}^k\rangle : U_{i.} \in \Delta_c, \forall i = 1, \ldots n\right\} \\ &= \max\left\{\sum_{i=1}^{n}\langle U_{i.}, Y_{i.}^k\rangle : U_{i.} \in V(\Delta_c), \forall i = 1, \ldots n\right\}. \end{aligned}$$

The last problem is separable and solving it amounts to solving n problems of the form

$$\max\left\{\langle U_{i.}, Y_{i.}^k\rangle : U_{i.} \in V(\Delta_c)\right\}$$

whose an optimal solution is given by

$$U_{i.}^{k+1} = e_{\arg\max_{j=1\ldots c} Y_{ij}^k}, \tag{18}$$

where $\{e_j : j = 1, \ldots, c\}$ is the canonical basis of $\mathbb{R}^c$.

The sequence $\left\{\mathcal{U}^k\right\}$ computed by DCA has the following interesting property which shows the efficiency of DCA in the search of the *optimal* number of clusters.

Proposition 1 *If at the iteration k one has $U_{il}^k = 0\ \forall i = 1, ...n$, then $U_{il}^{k+1} = 0\ \forall i = 1, ...n$. Consequently, if a cluster is empty at an iteration k then it can be deleted definitely in DCA. By the way, the sequence $\{c^k\}$ (c^k stands for the number of nonempty clusters at the iteration k) is decreasing during DCA's iterations and reduced to c^* at an iteration k_*, say $c^0 \geq c^1 \geq \ldots \geq c^k \geq c^{k+1} \geq \ldots \geq c^{k_*} = c^*$ and $c^k = c^*$ for all $k > k_*$.*

Proof. If there exists $l \in \{1, ...c\}$ such that $U_{il}^k = 0\ \forall i = 1, ...n$, then $Y_{il}^k = (B + \mu I)_{i.} U_{.l}^k = 0$ $\forall i = 1, ...n$. Now let $j_* := \arg\max_j Y_{ij}^k$, we will prove that $j_* \neq l$. Indeed, we have (e is the vector of ones in $\mathbb{R}^n$)

$$\sum_{j=1}^{c} Y_{ij}^k = \sum_{j=1}^{c} (B + \mu I)_{i.} U_{.j}^k = (B + \mu I)_{i.} \sum_{j=1}^{c} U_{.j}^k = (B + \mu I)_{i.} e^T = \left(\sum_{t=1}^{n} B_{it} \right) + \mu = \mu.$$

Hence $Y_{i.}^k$ contains at least one positive element and therefore $Y_{ij_*}^k := \max_{j=1...c} Y_{ij}^k > 0$. That means $j_* \neq l$. From (18) it follows that $U_{il}^{k+1} = 0$.

The above proposition allows us to update the number of clusters during DCA's iterations. Starting with c^0, a very large number of clusters (the number of nodes in the network), DCA detects automatically empty clusters. Then by removing all the columns l such that $U_{.l}^k = 0$ we reduce considerably (during some first iterations) the dimension of matrices Y^k and U^k.

The proposed DCA including the update of clusters number can be described as follows:

DCAM
initialization: Let U^0 be a $n \times c$ matrix with binary values, let c^0 be an integer number sufficiently large.
Let ε and ϵ be small positive values. Calculate $\mu = -\lambda_{min}(B) + \varepsilon$.
$k \leftarrow 0$. Set $c^0 \leftarrow c$
repeat
1. Compute $Y^k = (B + \mu I)U^k$,
2. Set $U_{i.}^{k+1} = e_{\arg\max_{j=1,\ldots,c^k} Y_{ij}^k}, \forall i \in \{1, \ldots, n\}$,
3. For $l = 1, ...c$
   if $U_{.l}^{k+1} = 0$ then update $c - 1 \leftarrow c$, remove the column $U_{.l}^{k+1}$ from U^{k+1}
   end for

$k \leftarrow k + 1$,
until $\frac{\|U^{k+1} - U^k\|}{\|U^k\|} < \epsilon$.
.

Theorem 1 (Convergence properties of DCAM)
(i) DCAM generates the sequence $\left\{\mathcal{U}^k\right\}$ contained in $V(\Delta)$ such that the sequence $\left\{-Q(\mathcal{U}^k)\right\}$ is decreasing (or, equivalently, the sequence $\left\{Q(\mathcal{U}^k)\right\}$ is increasing)
(ii) The sequence $\left\{\mathcal{U}^k\right\}$ converges to $\mathcal{U}^* \in V(\Delta)$ after a finite number of iterations.
(iii) The point $\mathcal{U}^*$ is a KKT point of the problem (15). Moreover, if

$$\arg\max_j (B + \mu I)U_{ij}^* \quad \text{is a singleton } \forall i \in \{1, \ldots, n\}, \tag{19}$$

then $\mathcal{U}^*$ is a local solution to (15).

Proof. (i) and (ii) are direct consequences of the convergence properties of DCA for a polyhedral DC program. Moreover, since h is differentiable everywhere, the condition $\partial h(\mathcal{U}^*) \cap \partial g(\mathcal{U}^*) \neq \emptyset$ becomes $\partial h(\mathcal{U}^*) \subset \partial g(\mathcal{U}^*)$ which is the KKT condition for the problem (15). Hence the first part of (iii) is straightforward. Only the second part of (iii) needs a proof.

We first note that $g := \chi_\Delta$ is a polyhedral convex function, so is its conjugate $g^* := \chi_\Delta^*$. Hence the dual DC program of (15) is a polyhedral DC program and the relation

$$\emptyset \neq \partial g^*(\mathcal{Y}^*) \subset \partial h^*(\mathcal{Y}^*) \tag{20}$$

is a necessary and sufficient local optimality condition [29,50,51]. Clearly that, the condition (19) is verified if and only if the problem

$$\min \left\{ \chi_{\Delta}(\mathcal{U}) - \langle \mathcal{U}, \mathcal{Y}^* \rangle : \mathcal{U} \in \mathbb{R}^{n.c} \right\} \tag{21}$$

admits an unique optimal solution, i.e. g^* is differentiable at $\mathcal{Y}^* = (B_b + \mu I_b)\mathcal{U}^*$. In other words, by returning to matrix representation with $Y^* = (B + \mu I)U^*$, problem (21) can be expressed in the form (17) and uniqueness of solutions to (21) is holds iff, for $i = 1, ..., n$, the row $[(B+\mu I)U^*]_{i.}$ has a unique greatest entry. In this case the relation (20) holds, and therefore, $\mathcal{Y}^*$ is a local solution to the dual DC program of (15). Using the transportation of local minimizers (see the two properties mentioned in Section 3.1) between primal and dual DC programs, we conclude that $\mathcal{U}^*$ is a local solution to the problem (15).

3.3 Discussion

The main advantages of our algorithms have been mentioned in the introduction of the paper. Here we give a more detailed discussion about the complexity of the algorithm and the choice of c^0. DCAM algorithm is simple and easy to implement. One iteration of the algorithm only relies on very few basic operations, which leads to a very low computation cost. As explained in Section 3.1, DCA is a descent method without linesearch. Therefore, there is no need to evaluate the objective function numerous times as in standard gradient descent schemes, for instance.

An important point which contributes to the efficiency of the proposed algorithm comes from the fact that even if we reformulate the initial combinatorial problem as a continuous problem (equation 15), DCA works on a finite set which is the vertex set of a polytope. In fact, DCA consists of computing the gradient matrix Y^k and solving one linear program at each iteration. This linear program can be decomposed onto n linear programs on the $(c-1)-$simplex Δ_c whose optimal solutions are explicitly defined among its vertex set (one has to find a vertex l of Δ_c such that $Y^k_{il} = \max_{j=1...c} Y^k_{ij}$). Therefore, contrary to other relaxation approaches, in which cluster assignments are represented as a full matrix (see for instance the deterministic annealing method [37]), in the DCAM algorithm the affectation matrix is a binary matrix all along the iterations. Matrix operations can therefore be performed very efficiently. The total number of operations to solve this main linear program is nc.

We remark that the key operations in DCAM consist of computing the gradient matrix Y^k, which is the product of the full matrix B and the binary assignment matrix U. A naive implementation of this product costs $O(n^2)$ operations. However, as explained in section 2 (and in [45]), the modularity matrix is the sum of the sparse matrix A and the matrix $\omega\omega^T$ of rank 1. Therefore, the product can be performed as follows:

$$Y^k = (B + \mu I)U^k = (A + \mu I)U^k - \frac{1}{2m}\omega(\omega^T U^k).$$

The product $(A + \mu I)U^k$ requires $O(m)$ operations, where m denotes the number of edges (we don't take into account the μI term as it can be pre-calculated once and for all before iterations). The computation of $\frac{1}{2m}\omega(\omega^T U^k)$ requires $O(nc)$ operations. Therefore the total cost of computing Y^k of one iteration is $O(m + nc)$.

It should be noticed that Y^k can be computed incrementally. Indeed, denote by $\delta^{k+1} = U^{k+1} - U^k$, we have $Y^{k+1} = (B + \mu I)U^{k+1} = Y^k + (B + \mu I)\delta^{k+1}$. We observe that each row of δ^{k+1} which corresponds to a change in the assignment contains only two non null components: -1 for the previous community and $+1$ for the new community, and other rows are null. We denote q the number of changes in the assignments (number of non null rows in δ^{k+1}). Then, the incremental computation $Y^{k+1} = Y^k + (A + \mu I)\delta^{k+1} - \frac{1}{2m}\omega(\omega^T\delta^{k+1})$ requires roughly $2\min(m, nq)+2nc$ sums and products in the worst case. Therefore, thanks to simple calculations, if $q < \frac{m}{2n}$, the incremental approach is more efficient than the full calculation. If m takes a high value, the incremental approach will be used very often, and the running time will be shorter. In practice, the incremental approach can reduce running time of long experiment by 15%.

The computation of the smallest eigenvalue $\lambda_1(B)$ of the modularity matrix B can be done in $O((n+m)n)$ thanks to the shifted power method. We suppose here that the power method requires n iterations to converge (see [45]). It is worth noticing that DCA doesn't require an exact computation of $\lambda_1(B)$, we only need a tight lower bound on this eigenvalue. Therefore, in practical situations, the number of iterations is much smaller than n.

Finally, it is worth noting once again the great advantage of DCAM in finding automatically the optimal number of clusters: starting with a very large value c^0 (the number of communities), DCAM furnishes, as output results, an optimal partition together with the optimal number of communities c^*. As for the input value c^0, to avoid a "underestimation" of c^* we start with a very large value c^0, for example $c^0 = n$. Obviously, for large size networks, smaller c^0 is, shorter CPU time would be, and the choice of $c^0 = n$ in these networks is not reasonable. Hence a suitable value of c^0 should be chosen to avoid too many unnecessary calculations during the first iterations. It is well known in practice that the number of clusters of a network having n nodes is around $\sqrt{\frac{n}{2}}$. Hence we can take $c^0 \in [\sqrt{\frac{n}{2}}, n]$, for example $c^0 = n$ in small and medium size networks ($n \leq 500\ 000$) and larger n is, c^0 should be closer to $\sqrt{\frac{n}{2}}$.

In our experiments we observe that the sequence $\{c^k\}$ is decreasing (quickly during some first iterations of DCAM) and reduced to c^* at the half of DCAM scheme, and during the remaining iterations DCAM continues to optimize the modularity with this value c^*. So, by removing empty clusters and updating the value of c during DCAM's iterations we reduce considerably the complexity of DCAM.

4 Experimental results

We test DCAM on several real networks and compare it with six existing approaches. All the experiments have been conducted on a Intel(R) Core(TM) i7-2720QM CPU 2 × 2.20 Ghz with 4 Gb of memory.

4.1 Datasets and reference algorithms

To evaluate the performance of the algorithms, we conduct experiments on **15** well-known networks, which have been tested in several previous works (see [45] for instance).

The description in details of the datasets is presented in Table 1 (note that the considered optimization problem have nc variables and $nc + n$ constrains).We get these datasets from different sources:

- Zachary's Karate Club, Dolphin Social Network, Les Miserables, Books about US politics, American College football are obtained from the Mark Newman's network data repository at `http://www-personal.umich.edu/~mejn/netdata/`;
- Jazz musicians network, Email communication network, PF2177, GearBox Tmt/sym, Hook/1498, Af/shell10, Rgg_n_2_21_s0 and Rgg_n_2_22_s0 are downloaded from the University of Florida Sparse Matrix Collection at `http://www.cise.ufl.edu/research/sparse/matrices/`
- and Email Enron at `http://snap.stanford.edu/data/email-Enron.html`;

The DCAM is compared to six reference algorithms, the first four are heuristic: CNM - the fast modularity maximization algorithm developed by Clauset, Newman and Moore (CNM) [10]; FG - the improvement implementation version of CNM proposed by Wakita and Tsurumi in [63]; WT - the Walktrap algorithm based on a new distance between nodes using random walks [52]; SP - the Spectral Bisection algorithm proposed by Newman [45]. The last two algorithms proposed by Agarwal et al. [1] are based on mathematical programming: LP (linear programming followed by randomized rounding), and VP (a vector programming relaxation of a quadratic program which recursively splits one partition into two smaller partitions while a better modularity can be obtained). The software of these algorithms are obtained from

- `http://deim.urv.cat/~sgomez/radatools.php`) for the SP algorithm,

Name	Note	# vertices	# edges
Zachary's Karate Club	KAR	34	78
Dolphin Social Network	DOL	62	159
Les Miserables	MIS	77	254
Books about US politics	BOK	105	441
American College football	BAL	115	613
Jazz musicians network	JAZ	198	2,742
Email communication network	EMA	1,133	5,451
Pf2177	PF2	9,728	367,436
Email_Enron	EER	36,691	183,830
GearBox	GEB	153,746	4,617,075
Tmt/sym	TMT	726,713	2,903,837
Hook/1498	HOK	1,498,023	31,207,734
Af/shell10	AFS	1,508,065	27,090,195
Rgg_n_2_21_s0	RG1	2,097,152	14,487,995
Rgg_n_2_22_s0	RG2	4,194,304	30,359,198

Table 1 The description of datasets

- `http://cs.unm.edu/~aaron/research/fastmodularity.htm`) for CNM,
- `http://igraph.sourceforge.net/`) for FG and WT (the Igraph library on C^{++}),
- `http://www-scf.usc.edu/~gaurava/` for VP and LP (Agarwal's web page).

4.2 Experiment setting

We consider two versions of DCAM according to the choice of initial points. The first, called DCAM1, starts with a random initial point with values in $\{0, 1\}$. The second, DCAM2, starts with a particular point computed by the following two-steps procedure. At the first step, each node in the network is assigned a random numeric label which belongs to $\{1, 2, ..., c\}$. In the second step, the label of every node is updated by the new label corresponding to the most frequent in its neighborhood. This step is performed iteratively with a small number of iterations (2 iterations in our experiment). Note that the idea of step 2 is based on the Label Propagation Algorithm (LPA) proposed by Raghavan et al. [53].

Since the initial point is randomly computed (even if DCAM2 uses a particular procedure for initial points, in the first step the label of points are randomly chosen), we run each DCAM 5 times and take the best solution (i.e. the one corresponding to the largest modularity value). CPU time is computed as the total CPU of 5 runs, including the time for finding the initial point. As for the input value c^0, at the first run we take $c^0 = n$ except for TMT, HOK, AFS (resp. RG1, RG2) datasets we set $c^0 = 5.\sqrt{\frac{n}{2}}$ (resp. $c^0 = \sqrt{\frac{n}{2}}$). Let c^{*1} be the number of "optimal" clusters given by the first run of DCAM. Since the solution given by DCAM is sub-optimal, c^{*1} is close to the "real optimal" number of clusters. Hence in the remaining four runs we start DCAM with a larger value c^0, say $c^0 := 2c^{*1}$.

In the stop condition of DCAM $\varepsilon = 10^{-6}$ for the first ten datasets and $\varepsilon = 10^{-9}$ for the last five datasets.

In the second experiment we study the influence of the value c^0 on the performance of DCAM. We run DCAM2 5 times for each of three values of c^0 that are n, $\frac{n}{2}$ and $5.\sqrt{\frac{n}{2}}$.

4.3 Numerical results

The modularity values given by DCAM1, DCAM2 and the references algorithms as well as the corresponding number of communities are presented in Table 2. Here, c^* and Q denote, respectively, the number of communities and the modularity value furnished by the algorithms. More clearly, the Figure 1 presents the comparative modularity results of the algorithms.

The CPU time in seconds of all algorithms are presented in Table 3.

The results of DCAM when c^0 varies are reported in Table 4 where c^* and Q are the best value given by 5 runs and CPU time is the total CPU time of 5 runs.

Comments on numerical results.

Table 2 Comparative results (number of communities and modularity value) of 8 algorithms

Dataset	DCAM1		DCAM2		CNM		WT		SP		FG		VP		LP		
	c*	Q	c*	Q	c*	Q	c*	Q	c*	Q	c*	Q	c*	Q	c*	Q	UB
KAR	4	**0.420**	4	**0.420**	3	0.381	3	0.394	4	0.393	3	0.381	3	**0.420**	4	**0.420**	0.420
DOL	5	**0.529**	4	0.528	4	0.495	5	0.501	5	0.491	4	0.496	3	0.526	5	**0.529**	0.531
MIS	6	0.549	6	**0.560**	5	0.501	8	0.521	8	0.521	5	0.501	4	**0.560**	6	**0.560**	0.561
BOK	5	0.526	5	**0.527**	4	0.502	4	0.515	4	0.467	4	0.502	3	**0.527**	5	**0.527**	0.528
BAL	10	**0.605**	10	**0.605**	7	0.577	10	0.603	8	0.477	6	0.550	5	**0.605**	10	**0.605**	0.606
JAZ	4	**0.445**	4	**0.445**	4	0.439	10	0.433	3	0.394	4	0.439	3	**0.445**	8	**0.445**	0.446
EMA	39	0.535	12	0.551	13	0.513	44	0.525	22	0.493	17	0.501	4	**0.579**	5	0.526	-
PF2	19	0.546	10	**0.558**	10	0.545	20	0.454	17	0.431	10	0.545	-	-	-	-	-
EER	1282	0.521	1072	**0.579**	1637	0.510	-	-	2631	0.409	1605	0.517	-	-	-	-	-
GEB	431	0.735	142	**0.841**	17	0.831	-	-	-	-	19	0.831	-	-	-	-	-
TMT	3409	0.618	136	**0.829**	9	0.797	-	-	-	-	10	0.798	-	-	-	-	-
HOK	2433	0.594	16	**0.854**	-	-	-	-	-	-	-	-	-	-	-	-	-
AFS	1736	0.627	8	**0.738**	-	-	-	-	-	-	-	-	-	-	-	-	-
RG1	1954	0.758	277	**0.839**	-	-	-	-	-	-	-	-	-	-	-	-	-
RG2	1448	0.755	350	**0.827**	-	-	-	-	-	-	-	-	-	-	-	-	-

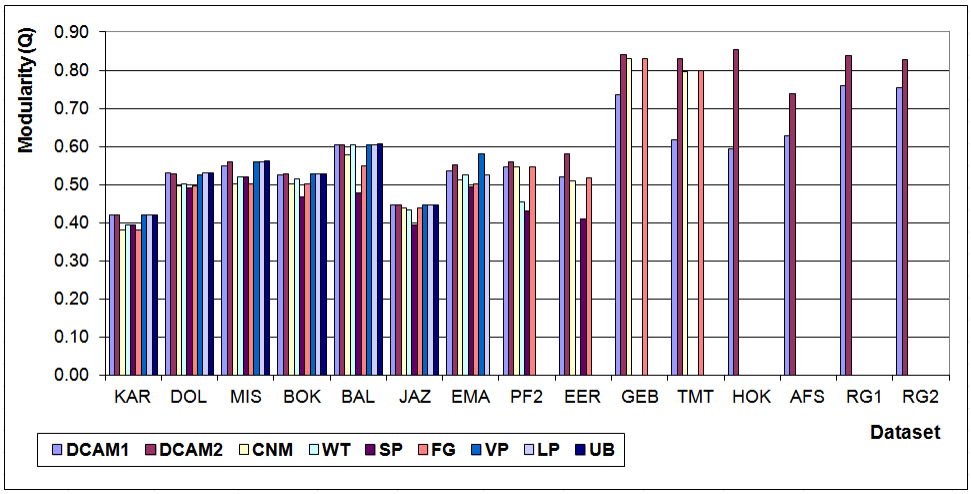


Fig. 1 Comparative modularity values of 8 algorithms

4.3.1 Comparison between DCAM and the four heuristic algorithms CNM, SP, WT, FG

DCAM2 always gives the largest modularity value while the running time is shortest. The total CPU time of DCAM2 is much smaller than those of these four reference algorithms. Considering the four medium size networks FP2, EER, GEB, TMT ($9000 < n < 1000000$): the gain ratio varies, respectively, from 27 to 108.5 and from 1.12 to 76 in comparing with CNM and FG (the fastest heuristic algorithm); WP (resp. SP) can solve only one (resp. two) problem(s) and the gain ratio is 53 (resp. 11.6 and 576) times. No heuristic algorithm can handle large scale problems (the networks with more than one million nodes) while DCAM2 is scalable: it works well on large size networks with upto $4,194,304$ nodes and $30,359,198$ edges.

The modularity values given by DCAM1 are quite close to (but slightly smaller than) those of DCAM2 and they are better than those computed by CNM, SP, WT (resp. FG) on all (resp. on 9/11) datasets. DCAM1 is faster than CNM, WT and SP on all datasets, the gain ratio is up to 38, 21, and 225 times w.r.t CNM, WT and SP respectively. DCAM1 is faster (resp. slower) than FG on the first eight (resp. EER, GEB, TMT) datasets.

Table 3 CPU time (in second) of each algorithm

Data	DCAM1	DCAM2	CNM	WT	SP	FG	VP	LP
KAR	**0.000**	**0.000**	0.116	**0.000**	0.015	**0.000**	18.51	0.69
DOL	**0.000**	**0.000**	0.223	**0.000**	0.028	**0.000**	19.29	4.53
MIS	**0.000**	**0.000**	0.551	0.015	0.038	0.000	26.13	3.98
BOK	**0.000**	**0.000**	0.367	0.048	0.066	0.015	21.14	18.45
BAL	**0.000**	**0.000**	0.394	0.000	0.096	0.000	34.95	12.79
JAZ	**0.000**	**0.000**	0.747	0.031	0.228	0.016	27.14	2.52
EMA	0.172	**0.000**	4.371	0.280	3.148	0.078	386.82	37.82
PF2	2.352	**0.782**	76.349	42.619	451.11	59.748	-	-
EER	194.05	**36.720**	231.355	-	2686.1	50.779	-	-
GEB	215.37	**80.110**	2372.0	-	-	179.93	-	-
TMT	705.43	**167.15**	4560.37	-	-	188.57	-	-
HOK	5597.21	**102.43**	-	-	-	-	-	-
AFS	3765.23	**81.97**	-	-	-	-	-	-
RG1	4895.21	**880.96**	-	-	-	-	-	-
RG2	5800.30	**1966.2**	-	-	-	-	-	-

Table 4 Results of DCAM2 with different values of c^0

	$c^0 = n$			$c^0 = \frac{n}{2}$			$c^0 = 5\sqrt{\frac{n}{2}}$		
Dataset	c*	Q	CPU	c*	Q	CPU	c*	Q	CPU
KAR	4	0.420	0.000	4	0.420	0.000	4	0.419	0.000
DOL	4	0.528	0.000	5	0.529	0.000	4	0.527	0.000
MIS	6	0.560	0.000	6	0.555	0.000	6	0.555	0.000
BOK	5	0.527	0.000	5	0.527	0.000	5	0.527	0.000
BAL	10	0.605	0.000	10	0.605	0.000	9	0.598	0.000
JAZ	4	0.445	0.000	4	0.445	0.000	4	0.445	0.000
EMA	12	0.551	0.062	12	0.551	0.061	10	0.538	0.020
PF2	10	0.558	1.981	10	0.558	1.342	10	0.558	0.650
EER	1072	0.579	62.356	1072	0.567	53.725	421	0.561	47.800
GEB	133	0.849	356.506	131	0.839	185.562	104	0.842	107.120

4.3.2 Comparison between DCAM and the two mathematical programming based algorithms VP and LP

VP and LP work only on 7 small size networks (no more than 1133 nodes). In terms of modularity, the results of DCAM2, VP and LP are comparable: they are equal on 5/7 datasets and everyone wins on one dataset. DCAM1 gives the same result as VP (resp. LP) on 3/7 (resp. 4/7) datasets, better on one dataset, and smaller on 3 (resp. 2) datasets. Note that VP and LP algorithms include a refinement procedure and they perform well only on the seven small datasets. Note also that, for these small datasets, the modularity values provided by DCAM are very close to the upper bounds given in [1] (the gap is zero on one dataset, 0.001 on 5/7 datasets and 0.002 on one dataset).

As for running time, on the 7 datasets which can be handled by VP and LP, the CPU times of VP and LP varies from 0.69 to 386.82 seconds while those of DCAM2 (resp. DCAM1) are always smaller than 0.001 (resp. 0.6) second.

4.3.3 Comparison between the two versions of DCAM

Thanks to the initial procedure, DCAM2 is better than DCAM1 (with random initial points) on both quality and rapidity, in particular for large size networks. The gain ratio on running time is up to 55 times (HOK data). Meanwhile, even with a random starting point, DCAM1 is an efficient algorithm: it can handle large scale problems in a reasonable time.

4.3.4 The influence of c^0 on DCAM

DCAM is robust: c^* is quite stable when c^0 varies (except for EER data), and the difference of modularity values varies from 0.0 to 0.024. As for CPU time, not surprisingly, smaller c^0 is, shorter running time is.

5 Conclusion

We have studied the problem of detecting community structure in networks via the maximization of the modularity. By exploiting the special structure of the considered problem we have developed a fast and scalable DCA scheme (DCAM) that requires only matrix-vector product at each iteration. The implementation is performed by an incremental approach taking into account the sparsity of the modularity matrix. Our algorithm is neither a hierarchical partition, nor a $k-$ partition approach, and do not require any refinement: starting with a very large number of communities, DCAM furnishes, as output results, an optimal partition together with the optimal number of communities c^*, i.e., the number of communities is discovered automatically during DCA's iterations. The number of clusters is updated at each iteration when empty clusters are found, by the way computational efforts are considerably reduced. Thanks to the above elements, our algorithm can handle large size networks having up to $4,194,304$ nodes and $30,359,198$ edges, say the initial optimization problem has $6,073,352,192$ variables and $6,077,546,496$ constraints (as we start with $c^0 = 1448$ and the number of variables and of constraints are, respectively, nc and $nc + n$).

The DCAM has been compared to six reference algorithms on a variety of real-world network datasets. Experimental results show that it outperforms reference algorithms not only on quality and rapidity but also on scalability, and it realizes a very good trade-off between the quality of solutions and the running time.

In a future work we plan to investigate DCA for optimizing new modularity measures to identify overlapping communities or small communities which can not be revealed by the original modularity maximization problem.

References

1. Agarwal, G., Kempe, D.: Modularity-maximizing graph communities via mathematical programming. The European Physical Journal B 66(3), 409 418 (2008). DOI 10.1140 epjb/e2008-00425-1.
2. Aloise, D., Cafieri, S., Caporossi, G., Hansen, P., Perron, S. and Liberti, L. (2010) Column generation algorithms for exact modularity maximisation in networks. Phys. Rev. E 82, 046112.
3. Arenas, A., Fernandez, A., Gomez, S.: Analysis of the structure of complex networks at different resolution levels. NEW J.PHYS. 10, 053,039 (2008). URL doi:10.1088/ 1367-2630/10/5/053039
4. Barber, M.J., Clark, J. W., Detecting network communities by propagating labels under constraints, Phys. Rev. E 80 (2009) 026129.
5. Blondel, V.D., Guillaume, J., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. (2008). DOI 10.1088/1742-5468/2008/10/P10008.
6. Boccaletti, S., Ivanchenko, M., Latora, V., Pluchino, A. and Rapisarda, A.: "Detecting complex network modularity by dynamical clustering," Physical Review E, vol. 75, no. 045102(R), 2007.
7. Bradley, B.S. & Mangasarian, O.L. (1998). Feature selection via concave minimization and support vector machines. In J.Shavlik, editor, Machine Learning Proceedings of the Fifteenth International Conferences (ICML'98), 82–90, San Francisco, California, MorganKaufmann.
8. Brandes, U., Delling, D., Gaertler, M., Grke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: "On modularity clustering," IEEE Transactions on Knowledge and Data Engineering, vol. 20, no. 2, pp. 172–188, 2008.
9. Cafieri, S., Hansen, P., Liberti, L., A Locally Optimal Heuristic for Modularity Maximization of Networks, Phys. Rev. E 83, 056105 (2011)
10. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Phys. Rev. E 70(6), 066,111 (2004). DOI 10.1103/PhysRevE.70.066111.
11. Collobert, R., Sinz, F., Weston, J. & Bottou, L. (2006). Trading Convexity for Scalability. ICML06, 23rd International Conference on Machine Learning.
12. Danon, L., Duch, J., Diaz-Guilera, A., Arenas, A.: Comparing community structure identification (2005). URL doi:10.1088/1742-5468/2005/09/P09008.

13. Dempster, A.P., Laird, N.M. & Rubin, D.B. (1977). Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, Serie B, 39(1), 1–38.
14. Dinh, T.N.; Thai, M.T., Community Detection in Scale-Free Networks: Approximation Algorithms for Maximizing Modularity, Selected Areas in Communications, IEEE Journal on , vol.31, no.6, pp.997,1006, June 2013.
15. Djidjev, H.: "A scalable multilevel algorithm for graph clustering and community structure detection," Lecture Notes in Computer Science, vol. 4936, 2008.
16. Duch, J., Arenas, A.: Community detection in complex networks using extremal opti- mization. Phys. Rev. E 72(2), 027,104 (2005). DOI 10.1103/PhysRevE.72.027104
17. Emprise, Y.K.C., Dit-Yan, Y.: A Convex Formulation of Modularity Maximization for Community Detection, proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, pp. 2218–2225, IJCAI/AAAI (2011).
18. Evans, T.S., Lambiotte, R.: Line graphs, link partitions, and overlapping communities. Phys. Rev. E 80(1), 016,105 (2009). DOI 10.1103/PhysRevE.80.016105
19. Fortunato, S., Barthelemy, M.: Resolution limit in community detection. in proc. Natl.Acad.Sci.USA 104, 36 (2007). URL doi:10.1073/pnas.0605965104.
20. Fortunato, S. (2010) Community detection in graphs. Physics Reports 486, 75-174.
21. Guimera, R., Amaral, L.: Functional cartography of complex metabolic networks. Na- ture 433, 895 (2005). URL http://www.citebase.org/abstract?id=oai:arXiv.org: q-bio/0502035
22. Harel, D., Koren, Y.: On clustering using random walks. In: in proc. 21st conf. on Foundations of Software Technology and Theoretical Computer Science, vol. 2245, pp. 18–41. Springer-Verlag (2001)
23. Jonathan, Q.J., Lisa, J.M.:, Modularity functions maximization with nonnegative relaxation facilitates community detection in networks, Physica A: Statistical Mechanics and its Applications, Volume 391, Issue 3, ISSN 0378-4371, pp. 854-865 (2012).
24. Juyong, L., Steven ,P.G., Jooyoung, L., Mod-CSA: Modularity optimization by conformational space annealing, eprint arXiv:1202.5398 (2012)
25. Lai, D., Lu, H., Nardini, C.: Enhanced modularity-based community detection by random walk network preprocessing. Phys. Rev. E 81(6), 066,118 (2010). DOI 10.1103/PhysRevE.81.066118
26. Lambiotte, R., Delvenne, J., Barahona, M.: Laplacian dynamics and multiscale mod- ular structure in networks (2008). URL http://www.citebase.org/abstract?id=oai: arXiv.org:0812.1770
27. Laura, B., Songsong, L., Lazaros, G.P., Sophia, T., A Mathematical Programming Approach to Community Structure Detection in Complex Networks, Proceedings of the 22nd European Symposium on Computer Aided Process Engineering, pp. 1387–1391, (2012)
28. Le Thi, H.A. DC Programming and DCA. `http://lita.sciences.univ-metz.fr/~lethi/index.php/en/dca.html`.
29. Le Thi, H. A, Pham Dinh, T.: The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems. Annals of Operations Research pp. 23–46 (2005).
30. Le Thi, H.A., Belghiti, T. & Pham Dinh, T. (2006). A new efficient algorithm based on DC programming and DCA for Clustering. Journal of Global Optimization, 37, 593–608.
31. Le Thi, H.A., Le, H.M & Pham Dinh, T. (2006). Optimization based DC programming and DCA for Hierarchical Clustering. European Journal of Operational Research, 183, 1067–1085.
32. Le Thi, H.A., Le, H.M. & Pham Dinh, T. (2007). Fuzzy clustering based on nonconvex optimisation approaches using difference of convex (DC) functions algorithms. Journal of Advances in Data Analysis and Classification, 2, 1–20.
33. Le Thi, H.A., Le, H.M., Nguyen, V.V & Pham Dinh, T. (2008). A DC programming approach for Feature Selection in Support Vector Machines learning. Journal of Advances in Data Analysis and Classification, 2(3), 259–278.
34. Le Thi, H.A., Nguyen, V.V. & Ouchani (2008). Gene selection for cancer classification using DCA, Adv. Dat. Min. Appl. LNCS, 5139, 62–72.
35. Liu, Y., Shen, X. & Doss, H. (2005). Multicategory ψ-Learning and Support Vector Machine:Computational Tools. Journal of Computational and Graphical Statistics, 14, 219–236.
36. Liu, Y. & Shen, X. (2006). Multicategory ψ-Learning. Journal of the American Statistical Association, 101, 500–509.
37. Lehmann, S., Hansen, L.: Deterministic modularity optimization. Eur. Phys. J. B 60(1), pp. 83–88 (2007). DOI 10.1140/epjb/e2007-00313-2.
38. Li, Z., Zhang, S., Wang, R.S., Zhang, X.S., Chen, L.: Quantitative function for community detection, Phys. Rev. E, 77 (2008) 036109.
39. Mariá C.V.N., Leonidas, S.P.: Community detection by modularity maximization using GRASP with path relinking, Computers & Operations Research, Available online 19 March 2013, ISSN 0305-0548 (2013).
40. Massen, C., Doye, J.: "Identifying communities within energy landscapes," Physical Review E, vol. 71, no. 046101, 2005.
41. Medus, A., Acuna, G. and Dorso, C.: "Detection of community structures in networks via global optimization," Physica A, vol. 358, pp. 593–604, 2005.
42. Nadakuditi, R.R., Newman, M.E.J.: Graph Spectra and the Detectability of Community Structure in Networks, Phys. Rev. Lett. 108, 188701 (2012).
43. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Phys. Rev. E 69(6), 066,133 (2004). DOI 10.1103/PhysRevE.69.066133

44. Newman, M.E.J.: Finding community structure in networks using the eigenvectors of matrices. Phys. Rev. E 74(3), 036,104 (2006). DOI 10.1103/PhysRevE.74.036104
45. Newman, M.E.J.: Modularity and community structure in networks. Proc. Natl. Acad. Sci. USA 103(23), 85778582 (2006)
46. Newman, M.E.J.: Networks: An Introduction. Oxford University Press (2010)
47. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E 69(2), 026,113 (2004). DOI 10.1103/PhysRevE.69.026113
48. Neumann, J., Schnrr, C. & Steidl, G.: SVM-based Feature Selection by Direct Objective Minimisation. Pattern Recognition, Proc. of 26th DAGM Symposium, 212–219 (2004)
49. Ong, C.S. & Le Thi, H.A. (2011). Learning with sparsity by Difference of Convex functions Algorithm. J. Optimization Methods and Software, in Press 27 February 2012, DOI:10.1080/10556788.2011.652630, 14 pages
50. Pham Dinh, T., Le Thi, H.A.: Convex analysis approach to d.c programming: Theory, algorithms and applications. In: Acta Mathematica Vietnamica, pp. 289–355 (1997)
51. Pham Dinh, T., Le Thi, H.A.: DC optimization algorithms for solving the trust region subproblem. SIAM J. Optimization pp. 476–505 (1998)
52. Pons, P., Latapy, M.: Computing communities in large networks using random walks. J. of Graph Alg. and App. 10, pp. 284–293 (2004)
53. Raghavan, U. N., Albert, R., Kumara, S., Near linear time algorithm to detect community structures in large-scale networks, Phys. Rev. E 76 (2007) 036106.
54. Richardson, T., Mucha, P. and Porter, M.: "Spectral tripartitioning of networks," Physical Review E, vol. 80, p. 036111, Sep 2009.
55. Schuetz, P. and A. Caflisch, A. "Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement," Physical Review E, vol. 77, no. 046112, 2008.
56. Shen, X., Tseng, G.C., Zhang, X. & Wong, W.H. (2003). ψ-Learning. Journal of American Statistical Association, 98, 724–734.
57. Yuille, A.L. & Rangarajan, A. (2002). The Convex Concave Procedure (CCCP). Advances in Neural Information Processing System, 14, Cambrige MA, MIT Press.
58. Weber, S., Schle, T. & Schnrr, C. (2005). Prior Learning and Convex-Concave Regularization of Binary Tomography. Electr. Notes in Discr. Math., 20, 313–327.
59. Sun, Y., Danila, B., Josic, K., Bassler, K.E.: "Improved community structure detection using a modified fine-tuning strategy," Europhysics Letters, vol. 86, no. 28004, 2009.
60. Tasgin, M., Herdagdelen, A. and Bingol, H.:"Community detection in complex networks using genetic algorithms," arXiv:0711.0491, 2007.
61. Twan, V.L, Elena, M.: Graph clustering with local search optimization: The resolution bias of the objective function matters most, Phys. Rev. E 87, 012812 (2013).
62. Van Dongen, S.: Graph clustering by flow simulation. Ph.D. thesis, University of Utrecht (2000)
63. Wakita, K., Tsurumi, T.: Finding Community Structure in Mega-scale Social Networks, arXiv e-print cs/0702048 (2007)
64. White, S., Smyth, P.: A spectral clustering approach to finding communities in graph, in: H. Kargupta, J. Srivastava, C. Kamath, A. Goodman (Eds.), Proceedings of the 5th SIAM International Conference on Data Mining, Society forIndustrial and Applied Mathematics, Philadelphia, 2005, pp. 274–285.
65. Xu, G., Tsoka, S. and Papageorgiou, L.: "Finding community structures in complex networks using mixed integer optimization," Eur. Physical Journal B 60, 231-239 (2007).

2.2 Hierarchical Community Analysis approach

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

Mod-Müllner: An Efficient Algorithm for Hierarchical Community Analysis in Large Networks

Brieuc CONAN-GUEZ · Manh Cuong NGUYEN

Submitted version to the International Journal of Social Network Mining (IJSNM).
Received: date / Accepted: date

Abstract In this work, we propose a new efficient algorithm for hierarchical clustering analysis (HCA) of large networks. This algorithm, called Mod-Müllner, is an adaptation of an existing algorithm proposed by Müllner in 2011 and initially dedicated to HCA of pairwise dissimilarities. Mod-Müllner performs a greedy optimization of the modularity, a widely used measure for network partitioning. We show that thanks to adapted data structures, Mod-Müllner achieves lower running times than state of the art algorithms, while producing the same solutions.

Keywords graph, network partitioning, community detection, modularity, hierarchical clustering analysis.

1 Introduction

In recent years, the study of complex systems has attracted a great deal of interest in many disciplines, including physics, computer science, biology or social science. As explained in [1], networks are natural and powerful models to analyze such systems. We can cite as representative examples acquaintance networks, collaboration networks, citation networks, biochemical networks, the Internet...

One network property that has received a considerable amount of attention is the so-called community structure [2]: the whole network can be divided into separate meaningful communities/modules. A community is a densely connected sub-network with only sparser connections to other communities. Detection of communities is of significant practical importance as it allows the analysis of large networks at a mesoscopic scale: the identification of related web pages in the World Wide Web, the uncovering of communities in social networks, the decomposition of metabolic networks into functional modules.

There exists various definitions of a network community [3], but one in particular is widely used: it is the one based on the modularity measure Q proposed by Girvan and Newman in 2004 [4]. Such measure is used to quantify the quality of a given partition $\mathcal{P}$ of a network in separate communities. Thanks to this quantitative measure, the problem of community detection can be recast as a combinatorial optimization problem: by maximizing $Q(\mathcal{P})$, one can discover meaningful communities. Since its introduction, this measure has become a central tool for network analysis, and many works have studied its properties [5].

Since global maximization of the modularity measure Q is computationally intractable even for very small networks (this problem is NP-hard [6]), several heuristic algorithms have been developed to address the problem of large network partitioning [7, 8, 9, 10, 11, 12]. One method

Brieuc CONAN-GUEZ · Manh Cuong NGUYEN
Laboratory of Theoretical and Applied Computer Science - LITA EA 3097,
University of Lorraine, Ile du Saulcy, 57045 Metz, France.
brieuc.conan-guez@univ-lorraine.fr
manh-cuong.nguyen@univ-lorraine.fr

which is widely accepted by network scientists is the classical hierarchical clustering analysis (HCA) [13], which is based on a greedy optimization of Q. In the case of sparse hierarchical networks, a naïve implementation of HCA has a worst-case run-time complexity of $O(n^2)$ [14], where n denotes the number of vertices. Different works have focused on reducing the run-time complexity and the effective running time of the HCA thanks to adapted data structures and careful implementations. For instance, we can cite the proposal of [15] with a worst-case complexity of $O(n \ln^2(n))$.

In this paper, we propose a new efficient algorithm, called Mod-Müllner, for HCA of sparse networks. This algorithm is an adaptation of an existing algorithm proposed by Müllner in 2011 [16]. Müllner's algorithm allows fast HCA of plain dissimilarity matrices thanks to classical linkage criteria (Ward's criterion for instance). Our adaptation to sparse networks uses the significance as linkage criterion, a slightly modified version of the modularity linkage criterion. Moreover, in order to speed up the algorithm, an efficient data structure is used to represent the sparse linkage matrix. We show on simulated and real world data sets that our proposal achieves lower running times than state of the art algorithms on very large networks while producing the same solutions.

2 Greedy modularity optimization

2.1 Modularity

As explained in the introduction, the modularity measure Q is a quantitative measure, which evaluates the quality of a particular vertex clustering in a network. For non valued network, this measure is defined as the fraction of intra-community edges minus its expected value from the null model, a randomly rewired network with the same degree assignments [4].

Let's define the modularity measure in the more general framework of valued networks [17]. We consider an undirected valued network $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega)$, where $\mathcal{V} = \{1, \ldots, n\}$ is the vertex set, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$, is the edge set. We denote m the number of edges. Finally, we consider the weight function $\omega : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^+$, which is non negative. The degree of vertex u is defined as $d(u) = \sum_{v \in \mathcal{V}} \omega(u, v)$. In order to keep concise notations all along this paper, we generalize the weight function and degrees to subsets of vertices. For all community pairs $(\mathcal{C}_i, \mathcal{C}_j)$, we have $\omega(\mathcal{C}_i, \mathcal{C}_j) = \sum_{u \in \mathcal{C}_i, v \in \mathcal{C}_j} \omega(u, v)$. Moreover, $d(\mathcal{C}) = \omega(\mathcal{C}, \mathcal{V})$.

The modularity of community partition $\mathcal{P} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ is given by:

$$Q(\mathcal{P}) = \sum_{\mathcal{C} \in \mathcal{P}} \left(\frac{\omega(\mathcal{C}, \mathcal{C})}{\omega(\mathcal{V}, \mathcal{V})} - \frac{d(\mathcal{C})^2}{\omega(\mathcal{V}, \mathcal{V})^2} \right)$$

The first fraction is the actual community weight ratio, whereas the second one is the expected value of that ratio for a network with the same vertex set and in which edge weights have been distributed randomly independently of partition $\mathcal{P}$ while preserving vertex degrees.

The modularity measure Q takes values in the interval $[-1, 1]$, values approaching $Q = 1$ indicate networks with a strong community structure. One important property of this measure is that Q can be used to select the optimal number of communities k, by finding the value k for which Q is maximized.

2.2 Significance: a linkage criterion

It is straightforward to compute the modularity increase which results from the merge of two disjoint communities [17]:

$$\Delta Q(\mathcal{C}_i, \mathcal{C}_j) = 2 \frac{\omega(\mathcal{C}_i, \mathcal{C}_j)}{\omega(\mathcal{V}, \mathcal{V})} - 2 \frac{d(\mathcal{C}_i) d(\mathcal{C}_j)}{\omega(\mathcal{V}, \mathcal{V})^2}$$

A major property of this linkage criterion is that merge of a pair of unconnected communities can't lead to a modularity increase. As we will see, this fact greatly speeds-up community detection algorithms.

ΔQ is usually used as linkage criterion during HCA to select the next pair of communities to merge. However, several works report [18, 17] that this natural criterion suffers from a significant drawback. Indeed, it tends to favor the growth of a small number of communities, the "super communities" as called by Aaron Clauset. As a consequence, the community size distribution is highly right-skewed. This distribution is not representative of the graph structure, but is an artefact produced by the algorithm and the modularity linkage criterion.

This phenomenon affects the results in two ways. First this leads to sub-optimal partitions: the modularity value is not as high as it could be. Secondly, the dendrogram, the tree which represents the order of the merges, is heavily unbalanced. In this case, if we denote d the dendrogram height, the usual approximation $d \sim \ln(n)$ doesn't hold, and is replaced by $d \sim n$. As a direct consequence, the run-time complexity of the HCA is increased: the almost linear run-time complexity of state of the art algorithms is replaced by a super quadratic complexity, and computation times are dramatically impacted.

To speed up HCA and to improve partition quality, some authors have proposed to use modified versions of ΔQ as linkage criterion, which are called merge prioritizes. We can cite [19, 18]. In this work, we will focus on one of them: the significance S proposed in [17].

$$S(\mathcal{C}_i, \mathcal{C}_j) = \frac{\Delta Q(\mathcal{C}_i, \mathcal{C}_j)}{\sqrt{d(\mathcal{C}_i)d(\mathcal{C}_j)}}$$

The significance is a normalized version of ΔQ. This normalization cures the two flaws exposed above (partition quality and running time) by favoring merges of communities of similar sizes. In a general study [17], authors compare different merge prioritizes, and finally recommend to use systematically the significance as a replacement of the modularity increase ΔQ for HCA.

2.3 Agglomerative hierarchical clustering of networks

HCA [13] are usually applied to individuals described thanks to plain dissimilarity matrices. At the beginning, HCA considers the partition of singletons. At each iteration of the algorithm, the pair of clusters that are the most similar, in the sense of a given linkage criterion, are merged. Successive merges produce a sequence of nested partitions. After the final merge, the last partition is only composed of a single set: the set of all the individuals. The whole process produces a data structure: the dendrogram. A naïve implementation of this algorithm leads to a run-time complexity of $O(n^3)$ for plain dissimilarity matrices.

In the context of network analysis, some specific points have to be outlined:

- as the modularity increases only for pairs of connected communities, search of the best pair has a complexity of $O(m)$ for a naïve implementation. And for sparse networks, that is networks in which the number of edges m scales with the number of vertices n, this complexity is reduced to $O(n)$;
- at the beginning of the HCA, whatever the linkage criterion used is, ΔQ or S, each merge leads to a modularity increase. However, at a certain stage of the dendrogram building, modularity increase is not possible anymore(ΔQ are all negatives). At this point, the merging process is ended (dendrogram is not complete), and the best partition is obtained. In this sense, the modularity measure provides the number of communities in a network.

To obtain a fast implementation of HCA algorithms, a data structure is used during the course of the algorithm to store the values of the linkage criterion S. This data structure represents the sparse linkage matrix. This approach saves numerous computations during the search of the best pair to merge. If we merge communities $\mathcal{C}_i, \mathcal{C}_j$, the linkage matrix update can be done efficiently thanks to the following equations (see [15] for the case of ΔQ):

– if $\mathcal{C}_k$ is connected to both $\mathcal{C}_i$ and $\mathcal{C}_j$:

$$S(\mathcal{C}_i \cup \mathcal{C}_j, \mathcal{C}_k) = \sqrt{\frac{d(\mathcal{C}_i)}{d(\mathcal{C}_i) + d(\mathcal{C}_j)}} S(\mathcal{C}_i, \mathcal{C}_k) + \sqrt{\frac{d(\mathcal{C}_j)}{d(\mathcal{C}_i) + d(\mathcal{C}_j)}} S(\mathcal{C}_j, \mathcal{C}_k) \tag{1}$$

– if $\mathcal{C}_k$ is connected to $\mathcal{C}_i$, but not to $\mathcal{C}_j$:

$$S(\mathcal{C}_i \cup \mathcal{C}_j, \mathcal{C}_k) = \sqrt{\frac{d(\mathcal{C}_i)}{d(\mathcal{C}_i) + d(\mathcal{C}_j)}} S(\mathcal{C}_i, \mathcal{C}_k) - 2\sqrt{\frac{d(\mathcal{C}_k)}{d(\mathcal{C}_i) + d(\mathcal{C}_j)}} \frac{d(\mathcal{C}_j)}{\omega(\mathcal{V}, \mathcal{V})^2}$$

– if $\mathcal{C}_k$ is connected to $\mathcal{C}_j$, but not to $\mathcal{C}_i$:

$$S(\mathcal{C}_i \cup \mathcal{C}_j, \mathcal{C}_k) = \sqrt{\frac{d(\mathcal{C}_j)}{d(\mathcal{C}_i) + d(\mathcal{C}_j)}} S(\mathcal{C}_j, \mathcal{C}_k) - 2\sqrt{\frac{d(\mathcal{C}_k)}{d(\mathcal{C}_i) + d(\mathcal{C}_j)}} \frac{d(\mathcal{C}_i)}{\omega(\mathcal{V}, \mathcal{V})^2}$$

Thanks to pre-computations of the different coefficients, these update formulae can be calculated very efficiently.

2.4 A brief overview of existing algorithms

Several works have been devoted to speeding up the agglomerative hierarchical clustering of networks:

The seminal work was published by Newman in 2004 [14]. The author describes the classical HCA adapted to greedy modularity optimization. The only optimization proposed in this work consists in searching the best pair to merge amongst connected communities. The complexity of this naïve algorithm is $O((m+n)n)$, or $O(n^2)$ for sparse networks.

In the same year, Clauset, Newman and Moore [15] proposed a new algorithm for HCA, denoted *CNM*. The CNM principle is a major sophistication of the proposal of Day and Edelsbrunner [20] for the pairwise dissimilarity case. CNM has a complexity of $O(md\ln(n))$, where d is the height of the dendrogram. For sparse networks with a hierarchical structure ($d \sim \ln(n)$), the complexity can be expressed as $O(n\ln^2(n))$. This improvement over the naïve algorithm is obtained thanks to adapted data structures: each row of the linkage matrix is represented at the same time by a binary heap and by a balanced binary tree. The balanced binary tree allows for fast merge of two rows, while the max heap is used to quickly obtain the nearest neighbor of each community. Finally, a global max heap, which contains all the communities and their nearest neighbors, is used to select the best pair to merge.

In 2007, Wakita and Tsurumi [18] proposed a new efficient algorithm for HCA, denoted *WT*. Each community stores its neighboring communities in a linked list, which is ordered on the community ID. Moreover, each community maintains a reference to its nearest community. Finally, as in CNM, a global max heap is used to order pairs consisting of one community and its nearest neighbor. The complexity, as computed in [17], is $O(mdn)$, or $O(n^2\ln(n))$ for sparse hierarchical networks.

Finally, in 2011, Francisco and Oliveira [21] proposed an algorithm, denoted *FO*, which is presented as the fastest algorithm so far. For instance, authors report that FO is at least two times faster than CNM. The linkage matrix is represented thanks to a cross-linked adjacency list data structure (see Mod-Müllner data structure in figure 1). Unlike WT, lists are not ordered. In order to keep merge operations as fast as possible, each merge begins by the indexing of one of both rows. This indexing allows to merge rows in linear time. It is worth noticing that this algorithm takes advantage of the symmetry of the linkage matrix. It only stores the upper triangular part of the matrix, which saves numerous updates of the data structure. Another peculiarity of the algorithm is that it uses only one binary heap, which orders all the community pairs (the structure size is m). Finally, the complexity of FO is $O(md\ln(m))$ and for sparse hierarchical networks $O(n\ln^2(n))$.

3 Mod-Müllner: an algorithm for HCA of large networks

3.1 Müllner's algorithm for HCA of dissimilarity data

In 2011 Müllner proposed a new algorithm for HCA of plain dissimilarity matrices [16]. This algorithm is a sophistication of a former algorithm proposed by Anderberg [13]. This new proposal has a worst case complexity of $O(n^3)$ for a general linkage criterion, where n denotes the number of individuals. Although there exists other algorithms that exhibit lower worst case complexities [20], Müllner's algorithm is faster in practice, as shown by Müllner on various benchmarks [16].

Unlike the modularity optimization problem, we consider in this part a linkage criterion, denoted L, which has to be minimized. For instance, we can consider the Ward linkage criterion, or the single linkage criterion. Müllner's algorithm can be described succinctly as follows:

Clusters are represented by unique IDs: a list of consecutive integers $\{1, \ldots, n\}$. Since the linkage matrix is symmetric, the nearest cluster searches are done in the upper triangular par of this matrix. For instance, let us consider cluster i, the search for its nearest cluster, cluster j, is performed amongst clusters k such that $i > k$. Obviously, the best pair to merge, that is the one which has the lowest linkage criterion value, is part of these set of pairs cluster/nearest cluster.

At all time, Müllner's algorithm maintains two arrays: the first one is an array of *nearest cluster candidates*, denoted $ncc[.]$. The term "candidate" means that for a given cluster i, the true nearest cluster of i can be different from the candidate $ncc[i]$. This array of integers stores cluster IDs. The second array, denoted $lb[.]$, contains lower bounds on the linkage criterion values between each cluster i and all cluster k, such that $i < k$. More precisely, for all cluster i, the algorithm ensures that $lb[i] \leq L(\mathcal{C}_i, \mathcal{C}_k)$ for all $k > i$.

The efficiency of Müllner's algorithm relies on the following remark: thanks to these two arrays, nearest neighbor searches can often be avoided or are postponed to a later stage. As the data size is decreasing during the course of the algorithm (successive merges decrease the number of rows and columns of the linkage matrix), delayed searches can be done more quickly.

Here is what Müllner's algorithm does: first it initializes the two arrays $ncc[.]$ and $lb[.]$ thanks to a complete computation of nearest clusters. Then, at each of the $n-1$ iterations of the hierarchy construction, the cluster with the lowest lower bound is searched. Let us denote i this cluster with the lowest lower bound $lb[i]$. Two cases have to be considered at this point:

- in the first case, $lb[i]$ is strictly lower than the linkage criterion between i and $ncc[i]$, its nearest cluster candidate. We have $lb[i] < L(\mathcal{C}_i, \mathcal{C}_{ncc[i]})$. This implies that we have no guarantee that the nearest cluster candidate $ncc[i]$ is actually the true nearest cluster. In such case, the algorithm is forced to recompute the true nearest cluster of i thanks to a complete linear search. $ncc[i]$ and $lb[i]$ are updated with the values of the true minimum. As the new value of $lb[i]$ may now be greater than the lowest lower bound, the pair $(i, ncc[i])$ may not be the best pair to merge. So the whole process has to be restarted: the cluster with the lowest lower bound is searched once again.
- in the alternative case, $lb[i]$ is equal to the linkage criterion between i and $ncc[i]$. We have $lb[i] = L(\mathcal{C}_i, \mathcal{C}_{ncc[i]})$. As we have a realization of the lower bound, this means that the nearest cluster candidate $ncc[i]$ is the true nearest cluster. At this point, the two clusters i and $ncc[i]$ are merged as a new cluster $j = ncc[i]$ (cluster i is removed). And the data structures are updated: the linkage matrix, and the two arrays $ncc[.]$ and $lb[.]$.

During this linkage matrix update, the column/row of the deleted cluster i is removed and the column/row of the newly created cluster $j = ncc[i]$ is updated. Let us consider updates associated to some cluster k. If the nearest cluster candidate $ncc[k]$ was cluster i, it is replaced by another cluster (as $k < j$, the new cluster j seems a very sensible choice). Thereafter, if the new value $L(\mathcal{C}_k, \mathcal{C}_j)$ is lower than $lb[k]$, $ncc[k]$ is updated by j, and $lb[k]$ is updated by $L(\mathcal{C}_k, \mathcal{C}_j)$. Finally, for the new cluster j, $ncc[j]$ and $lb[j]$ are updated thanks to a complete linear search.

In order to further speed up the algorithm, Müllner cached lower bounds $lb[.]$ in a priority queue, implemented thanks to a binary min heap. Thanks to this priority queue, lowest bound

extractions are done in $O(\ln(n))$. However, care must be taken now to update this additional data structure during the linkage matrix update.

3.2 Mod-Müllner: an adaptation of Müllner's algorithm for greedy modularity optimization

3.2.1 Adaptation principles

In this section, our adaptation of Müllner's algorithm to modularity optimization is described[1]. We call this adaptation *Mod-Müllner*. Figure 1 describes states of Mod-Müllner algorithm before and after the merge of two communities.

This adaptation can be summarized in four points:

- the linkage criterion S (or ΔQ) has now to be maximized. So the lower bound array is replaced by an upper bound array $ub[.]$, and the binary min heap is replaced by a binary max heap which orders the upper bounds. Updates are done accordingly.
- as usual for modularity optimization of sparse networks, only pairs of connected communities have to be considered. Therefore, iterations over data structures have to be adapted. Especially, upper bound computations have to be slightly modified to ensure that these bounds are greater than all linkage criterion values, even those of non neighboring communities. This can be obtained simply by forcing upper bounds to be greater or equal to 0.
- as the nearest community is sought amongst communities with greater ID (upper triangular par of the linkage matrix), it may happen that a community i has no such neighbors. In such case, the upper bound is once again fixed to 0 and the nearest community candidate $ncc[i]$ is fixed to a special value (denoted NA in figure 1). This has no consequences for the success of the algorithm, as it stops as soon as the linkage criterion is null (if the linkage criterion S is null, this implies that the modularity can't be increased any more).
- finally the major adaptation doesn't apply to Müllner's algorithm itself, but focuses on the data structure which represents the linkage matrix. As explained below, we propose to use a slightly modified implementation of FO data structure [21]: the sparse linkage matrix is represented thanks to a cross-linked adjacency list data structure (see figure 1). Each node of this structure represents a weighted edge of the coarsened graph of communities. Each value of the linkage matrix, the significance, is stored in a node. This data structure implies a modification on the nature of the nearest community candidate array $ncc[.]$. In the original Müllner's algorithm, this array contains cluster IDs, whereas in the Mod-Müllner adaptation this array contains references to nodes of the linkage matrix. This gives direct access to the node which contains the two communities to merge.

3.2.2 Representing the sparse linkage matrix thanks to an efficient data structure

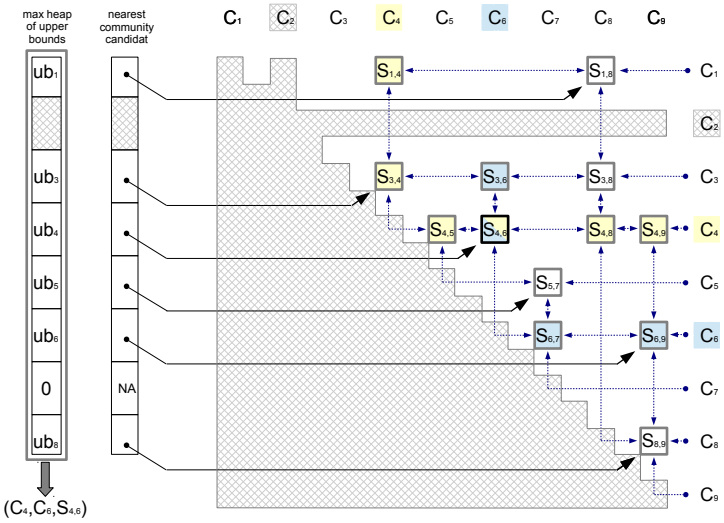
Our adaptation only needs the upper triangular part of the linkage matrix. As explained in the previous section, we use the same cross-linked adjacency list data structure as the FO algorithm. Each node contains seven pieces of information: two integers for the IDs of the two communities, one real number for the linkage criterion, and four references to neighboring nodes.

When a nearest community candidate $ncc[i]$ has to be recomputed, the search has to be performed only amongst nodes (i, k) for which community IDs k are greater than i. If nodes are inserted in the data structure carelessly, as in the FO algorithm, iterations have to be done over all neighboring communities k, and therefore those such that $k < i$ have to be skipped. This is inefficient. To avoid this problem, we propose to insert nodes such that $k > i$ at the front of the doubly linked list, and those such that $k < i$ at the back of the list. This allows to iterate efficiently on nodes such that $k > i$.

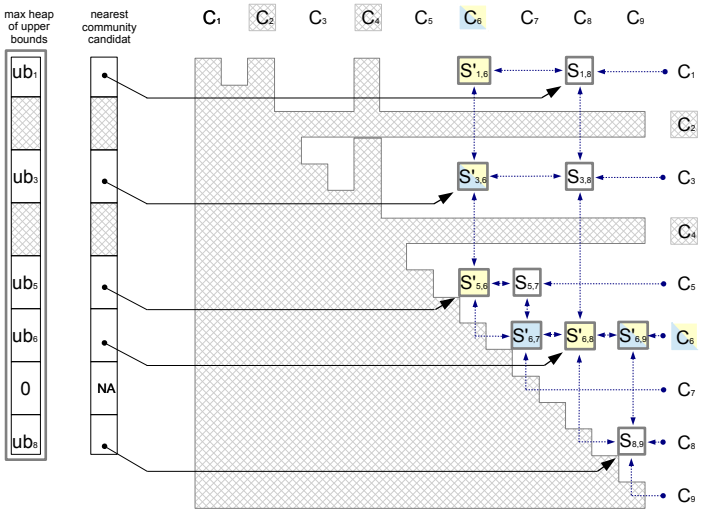
Finally, we recall that during the merge of two communities, the algorithm must find neighbors which are shared by both communities (see formula 1). As lists are not ordered on community IDs (unlike the WT algorithm), a naïve implementation leads to a quadratic complexity:

[1] Available implementation: `https://www.dropbox.com/s/vvrpaehknp0og3o/networkPartitioning.zip`

Fig. 1 States of Mod-Müllner algorithm before and after a merge (for the sake of clarity, lists appear ordered by community IDs. It is not required by the algorithm).



Before the merge of communities C_4 and C_6



After the merge of communities C_4 and C_6

two nested loops are needed. To avoid this bottleneck, FO uses a simple indexing strategy: at the beginning of each merge, neighbors of the first community are indexed thanks to an array. Then a second loop processes each neighbor of the second community. Thanks to the indexing, the question of whether one neighbor of the second community is also a neighbor of the first community is resolved in constant time. A third and final loop is used to process residual neighbors of the first community: the ones which haven't been treated by the second loop. Thanks to these three successive loops, the merge operation has linear complexity (updates of additional data structures, for instance the binary max heap, are not taken into account in this complexity analysis). We use the same indexing strategy as FO in Mod-Müllner.

3.2.3 Formal description of Mod-Müllner algorithm

In this section, we present a formal description of Mod-Müllner algorithm. To keep algorithm 1 as concise as possible, some notations have been used:

- function Mod-Müllner accepts as entries n, the number of vertices, and $weightedPair[.]$, which is an array of triplets $(communityID1, communityID2, edgeWeight)$ describing the network. Self-loops and multi-edges are not allowed. The return value is a list of merges;
- a *node* structure represents an edge. It contains: $node.r$, the community ID of the row, $node.c$, the community ID of the column, $node.S$, the significance, and four references to neighboring nodes of the doubly linked lists. This structure requires that $node.r < node.c$;
- when *node* is inserted into the data structure, it is part of two doubly linked lists. As explained in the previous section, *node* is inserted at the front of list $edgeList[node.r]$, and at the back of $edgeList[node.c]$. We denote $row(edgeList[i])$ the subset of list $edgeList[i]$ such that $node.r = i$. It represents the first nodes of the list;
- $refTo(node)$ indicates that some kind of pointer on the structure *node* is stored;
- $neighborID(nodeU, u)$ returns the community which forms with community u the edge represented by the $nodeU$ structure. This is the other stub of the edge;
- *index* is an array of node references. Thanks to this array, merges of two communities can be done in linear time. This array should be allocated only once at the beginning of the algorithm (its length is n). Before and after each merge, this array references no nodes (It contains values NA).

Algorithm 1

function MOD-MÜLLNER(n, $weightedPair[.]$)
  $ub[i] \leftarrow 0,\ \forall i = 1, \ldots, n-1$ ▷ *an upper bound is never lower than 0*
  $ncc[i] \leftarrow NA,\ \forall i = 1, \ldots, n-1$
  $edgeList[.] \leftarrow$ CREATELINKAGEMATRIX$(n, weightedPair[.], ub[.], ncc[.])$
  $mergeList \leftarrow \{\}$ ▷ *this list will contain merges:* $(ID1, ID2)$
  $PQ \leftarrow createPriorityQueue(ub[.])$ ▷ *community IDs are ordered by* $ub[.]$
  while *PQ not empty* **do**
    repeat
      $u \leftarrow PQ.pop()$ ▷ *extract the community ID with the greatest upper bound*
      if $ub[u] \leq 0$ **then** ▷ *the modularity can't be increased anymore*
        return $mergeList$
      end if
      $isPairFound \leftarrow ub[u] = ncc[u].S$ ▷ *a boolean*
      if $not(isPairFound)$ **then**
        $ncc[u] \leftarrow refTo(\arg\max_{node \in row(edgeList[u])} node.S)$ ▷ *linear search*
        $ub[u] \leftarrow ncc[u].S$
        $PQ.push(\ (u, ub[u])\ )$
      end if
    until $isPairFound$
    $v \leftarrow ncc[u].c$ ▷ *pair* (u, v)
    $mergeList.pushBack(\ (u, v)\ )$
    UPDATELINKAGEMATRIX$(u, v, edgeList[.], ub[.], ncc[.])$
  end while
  return $mergeList$
end function

function CREATELINKAGEMATRIX$(n, weightedPair[.],$ **byRef** $ub[.],$ **byRef** $ncc[.])$
  $edgeList[i] \leftarrow \{\},\ \forall i = 1, \ldots, n$
  for all $pair \in weightedPair$ **do**
    $node \leftarrow createNode(pair)$ ▷ $node.r < node.c$ *and* $node.S \leftarrow Significance$
    $edgeList[node.r].pushFront(node)$
    $edgeList[node.c].pushBack(node)$

```
32:         if node.S > ub[node.r] then                          ▷ significance comparison
33:             ub[node.r] ← node.S
34:             ncc[node.r] ← refTo(node)
35:         end if
36:     end for
37:     return edgeList
38: end function

39: procedure UpdateLinkageMatrix(u, v, byRef edgeList[.], byRef ub[.], byRef ncc[.])
Require: index[i] = NA, ∀ i = 1, ..., n
40:     edgeList.removeNode(ncc[u])                  ▷ from edgeList[u] and from edgeList[v]
41:     index[neighborID(nodeU, u)] ← refTo(nodeU), ∀ nodeU ∈ edgeList[u]
42:     ncc[u] ← NA                                  ▷ ub[u] and ncc[u] won't be used any more
43:     ub[v] ← 0
44:     ncc[v] ← NA
45:     for all nodeV ∈ edgeList[v] do               ▷ second loop of the merge operation
46:         nodeU ← index[neighborID(nodeV, v)]
47:         if nodeU = NA then                       ▷ neighbor of v is not connected to u
48:             nodeV.updateSignificance()
49:         else                                     ▷ neighbor of v is also connected to u
50:             nodeV.updateSignificance(nodeU)
51:             if ncc[nodeU.r] = nodeU then         ▷ ncc[nodeU.r] = NA for nodeU.r = u
52:                 ncc[nodeU.r] ← refTo(nodeV)
53:             end if
54:             index[neighborID(nodeV, v)] ← NA
55:             edgeList.removeNode(nodeU)
56:         end if
57:         if nodeV.S > ub[nodeV.r] then
58:             ub[nodeV.r] ← nodeV.S
59:             ncc[nodeV.r] ← refTo(nodeV)
60:             if nodeV.r ≠ v then          ▷ PQ update is done at the end for nodeV.r = v
61:                 PQ.update( (nodeV.r, ub[nodeV.r]) )
62:             end if
63:         end if
64:     end for
65:     for all nodeU ∈ edgeList[u] do               ▷ third loop of the merge operation
66:         nodeV ← createNode(v, neighborID(nodeU, u), nodeU.S)
67:         nodeV.updateSignificance()
68:         if ncc[nodeU.r] = nodeU then             ▷ ncc[nodeU.r] = NA for nodeU.r = u
69:             ncc[nodeU.r] ← refTo(nodeV)
70:         end if
71:         index[neighborID(nodeU, u)] ← NA
72:         edgeList.removeNode(nodeU)               ▷ loop iterations have to be done carefully
73:         edgeList[nodeV.r].pushFront(nodeV)
74:         edgeList[nodeV.c].pushBack(nodeV)
75:         if nodeV.S > ub[nodeV.r] then
76:             ub[nodeV.r] ← nodeV.S
77:             ncc[nodeV.r] ← refTo(nodeV)
78:             if nodeV.r ≠ v then          ▷ PQ update is done at the end for nodeV.r = v
79:                 PQ.update( (nodeV.r, ub[nodeV.r]) )
80:             end if
81:         end if
82:     end for
83:     PQ.update( (v, ub[v]) )                      ▷ update is done if v ≠ n
84: end procedure
```

3.2.4 Run-time complexity analysis of Mod-Müllner

The analysis of the worst case complexity of the different steps of the algorithm can be summarized as follows:

- initialization: filling the linkage matrix is in $O(m)$, and the priority queue initialization is in $O(n)$;
- the inner loop (lines 8 to 19): in the worst case, the maximum number of iterations is n. If n iterations are performed, linear searches (arg max) in the worst case globally imply m computations, and managing the priority queue requires n updates ($O(\ln(n))$ for each).
- the merge operation: in the worst case, n nodes have to be considered, and each time, the priority queue may have to be updated.

The global worst case complexity is therefore $O(m + n(m + n\ln(n))) = O(nm + n^2\ln(n))$. In the case of a sparse hierarchical network, we obtain $O(n^2\ln(n))$. Mod-Müllner has therefore the same worst case complexity as WT, but exhibits a greater complexity than CNM or FO. However, in the next section, we show on simulated and real world networks, that Mod-Müllner achieves lower running times than FO, even for very large networks (experiments have been conducted on networks with more than 50 millions of edges).

4 Experiments

4.1 Experimental protocol

In this section, our goal is to show that Mod-Müllner is an efficient algorithm, which achieves lower running times than concurrent algorithms on very large networks. As Francisco and Oliveira claim that their algorithm, FO, is the best performer [21], all of our experiments have been conducted against this unique contender.

Moreover, in order to obtain a fair comparison between both algorithms, we have re-implemented the FO algorithm. We call this new implementation *MyFO*. MyFO uses the same code as the Mod-Müllner implementation for the linkage matrix. This allows us to greatly reduce implementation bias. In all of the conducted experiments, MyFO turns out to be faster than FO. This acceleration is due to the removal of several branch conditions which appear in the merge loop of the original implementation of FO.

Both algorithms use the significance S as linkage criterion. Indeed, as explained in section 2.2, significance outperforms ΔQ both in running time and in the quality of the obtained solutions. Therefore there is no practical interest to use ΔQ as linkage criterion.

All algorithms are implemented in C++/STL and compiled with GCC in 64 bit mode. For the sake of memory space, references (pointer of 64 bit size) have been replaced by simple integers (int of 32 bit size). This replacement is made possible because nodes are not allocated individually in the main memory. Instead, a large array which contains all the nodes is allocated only once at the beginning of the algorithm. This favors memory locality. Using integers (array indices) instead of pointers allows to process very large networks nevertheless. Finally, the cpu is an Intel Core i5 760 at 2.8 GHz with 8Go of main memory.

All running times were recorded once all the data were loaded into a temporary data structure in the main memory . Therefore, reported running times take into account the construction of cross-linked adjacency list data structures.

Finally, it is worth noticing that even if both algorithms apply the same principle: the community pair with the greatest significance is merged, and the solutions obtained by both methods may differ slightly in practice. This can be explained by the way ties are resolved. If two pairs exhibit the same linkage criterion value, both algorithms won't necessarily pick the same pair, which leads to different final solutions. In practice, the differences are not significant, and modularity values are very close.

4.2 Test set of simulated networks

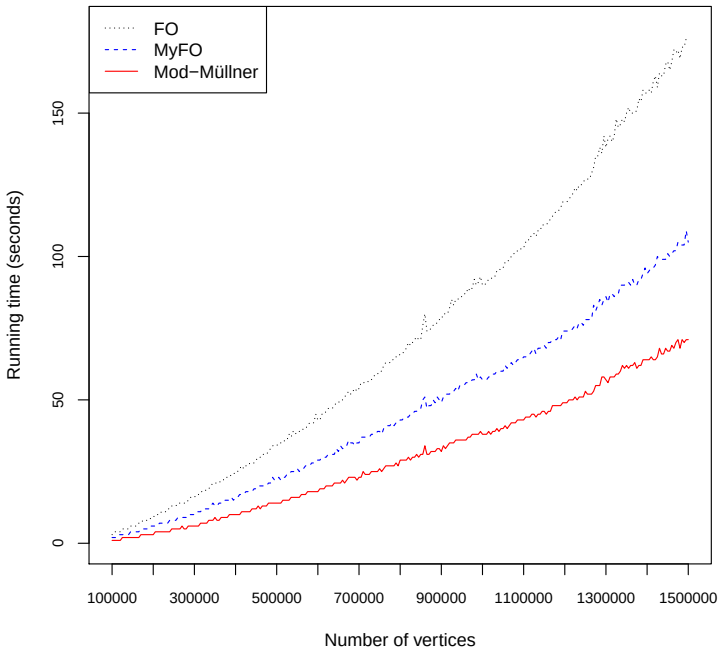
To perform numerical experiments on simulated data, we use the same generative network model as in [22]. We consider networks with $C = 100$ distinct communities. The number of intra (resp. inter) edges depends on a given probability p (resp. q) of edge occurrence. We have of course $p > q$. The ratio $f = q/p$ and vertex degrees are kept fixed: $f = 0.003$ and $d = 15$. Under these assumptions, the theoretical modularity can be computed: $Q(C, f) = 1/(1 + (C - 1)f) - 1/C = 0.76$. All these settings correspond to observed values in real world networks.

To avoid tie-breakings, networks are valued: edge weights are randomly chosen in the interval $[0.9, 1.1]$. This ensures that both algorithms find exactly the same solution (same modularity), which allows to compare running times fairly.

Finally, the number of vertices ranges from 100.000 to 1.500.000 (step=5000), and therefore the larger network has more than 11 million of edges. 280 networks were generated according to these settings.

We can note on figure 2, that Mod-Müllner is always faster than both FO and MyFO. Moreover our implementation MyFO always outperforms FO. In the case of the largest network, the one with 1.5 million of vertices, FO takes 3 minutes to reach the end, whereas MyFO takes only 1 minute and 45 seconds. Finally, Mod-Müllner takes 1 minute and 10 seconds. The acceleration factor of Mod-Müllner over MyFo is 1.5, whereas the one of MyFO over FO is 1.6.

Fig. 2 Running times of Mod-Müllner, FO and MyFO on simulated networks



4.3 Test set of real-world networks

In this section, we conduct experiments on 15 real world networks. The vertex number ranges from 0.5 million to 23 million, and the edge number ranges from 2.5 million to 50 million. For each implementation, Mod-Müllner, FO and MyFO, running times are expressed in seconds. Next

Network				Mod-Müllner				MyFO					FO		
Name	Vertex	Edge	Deg	Sec	Q	#C	H	Sec	Q	#C	H	$\frac{MyFO}{MM}$	Sec	Q	$\frac{FO}{MM}$
AMA	410 236	2 439 437	11.9	1.8	0.854	231	257	3.2	0.854	228	261	1.7	6.0	0.855	3.2
DEL	1 048 576	3 145 686	6.0	2.5	0.977	147	25	4.0	0.977	143	24	1.6	5.5	0.977	2.2
WBS	685 230	6 649 470	19.4	11.1	0.932	1 333	10 616	10.2	0.932	1 341	10 584	0.9	19.6	0.933	1.7
HUG	4 588 484	6 879 133	3.0	7.7	0.989	291	25	10.0	0.989	290	26	1.3	14.3	0.988	1.8
M6	3 501 776	10 501 936	6.0	13.3	0.982	191	26	19.0	**0.983**	200	26	1.4	27.0	0.983	2.0
3SP	3 712 815	11 108 633	6.0	12.4	**0.987**	220	26	18.6	0.986	214	26	1.5	25.1	0.987	2.0
ADA	6 815 744	13 624 320	4.0	11.1	0.987	258	25	16.9	0.987	253	26	1.5	24.5	0.987	2.2
KRO	731 706	15 888 007	43.4	997.0	0.051	121	76 449	405.3	0.051	121	76 448	0.4	865.8	0.051	0.9
RCE	14 081 816	16 933 413	2.4	42.3	0.997	1 188	39	46.6	0.997	1 193	43	1.1	54.4	0.997	1.3
SPR	1 632 803	22 301 964	27.3	113.8	0.603	22	2 681	139.6	0.603	25	2 695	1.2	254.2	0.602	2.2
AFS	1 508 065	25 582 130	33.9	4.6	0.965	92	202	10.1	**0.966**	95	204	2.2	10.0	0.965	2.2
RUS	23 947 347	28 854 312	2.4	65.8	0.998	1 562	46	73.1	0.998	1 574	47	1.1	85.4	0.998	1.3
HOK	1 498 023	29 709 711	39.6	8.6	0.919	39	265	21.5	**0.920**	42	262	2.5	45.6	0.920	5.3
WIK	3 515 067	42 375 912	24.1	172.6	**0.489**	1 179	4 973	227.1	0.488	1 187	5 559	1.3	459.1	0.491	2.7
D24	16 777 216	50 331 601	5.9	56.5	0.990	347	30	99.3	0.990	346	30	1.8	135.1	0.990	2.4

Table 1 Comparisons of Mod-Müllner, MyFO and FO on several real world networks (Deg: mean degree, Sec: running time in seconds, Q: modularity value, #C: number of communities, H: dendrogram height, $\frac{MyFO}{MM}$: acceleration factor of Mod-Müllner over MyFO).

to this information, in parentheses, the obtained modularity is reported. Finally, accelerating factors are given for Mod-Müllner compared to both MyFO and FO.

In table 1, we can note that the three implementations, Mod-Müllner, MyFO and FO reach very close modularity values. The third decimal differs only of one point in five experiments, three times in favor of MyFO. Similarly, the numbers of communities and the dendrogram heights are quite similar between Mod-Müllner and MyFO. We can conclude that even if the solutions obtained by the three implementations are not exactly the same, they are quite close to one another. A fair comparison of running times can therefore be conducted on these results.

We can note that MyFO is much faster than FO: the mean acceleration factor is 1.5. This fact corroborates results obtained on simulated networks. In the remainder of this analysis, we will focus exclusively on MyFO results.

We can note that Mod-Müllner is always faster than FO and MyFO except for two networks: WBS and KRO. For WBS, running times are quite similar, 11 seconds for Mod-Müllner and 10 seconds for MyFO. The case of the second network, KRO, is much more interesting. First, we can note that this network has the greatest mean degree $d = 43$. It is therefore legitimate to wonder whether this large number of neighbors has an impact on the relative performance of Mod-Müllner over MyFO. In studying other network results, such link between mean degree and relative performance is invalidated. For instance, networks AFS and HOK have high mean degrees too: 33.9 for AFS and 39.6 for HOK, and yet acceleration factors are quite important: 2.2 for AFS and 2.5 for HOK. Network KRO has two other peculiarities: first, we can note that HCA is unable to uncover a community structure for this network. The three implementations obtain the same modularity value of 0.051 which is very low. Indeed, in [14], Newman indicates that significant community structures lead to modularity values greater than 0.3. Secondly, the running times (and dendrogram heights) of the three implementations are very high, much higher than those obtained on much bigger networks. We can assume that KRO, and more generally networks which exhibit no community structure, badly impact optimizations and data structures used in the three implementations.

Finally, Mod-Müllner behaved very well on average: the mean acceleration factor is near 1.5, which corresponds to the value observed on the largest simulated network of the previous section. For the two largest networks, WIK (wikipedia) and D24 (delaunay), Mod-Müllner saves almost one minute for WIK and 45 seconds for D24. These are significant gains.

5 Conclusion

In this paper, we propose a new efficient algorithm, Mod-Müllner, for the hierarchical clustering analysis of large networks. We show on simulated and real world networks that Mod-Müllner achieves a 1.5 acceleration factor on average over the fastest algorithm known to date: FO. For instance, Mod-Müllner is able to uncover the community structure of a 50 million edge network in less than one minute, whereas the optimized implementation of FO takes 30 seconds more. With respect to clustering quality, Mod-Müllner produces the same solutions as FO, if we except the fine variations induced by tie-breakings. Finally, as for all hierarchical analysis algorithms, solutions could be further improved by the well known multi-level refinement schema [17]. We believe that the association of Mod-Müllner with the multi-level refinement allows to obtain very quickly quality solutions.

References

1. M. E. J. Newman. *Networks: An Introduction*. Oxford University Press, 2010.
2. M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Science*, 99:7821–7826, June 2002.
3. L. Danon, J. Duch, A. Diaz-Guilera, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 9:8, sep 2005.
4. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb 2004.

5. S. Fortunato and M. Barthelemy. Resolution limit in community detection. *in proc. Natl.Acad.Sci.USA*, 104:36, 2007.
6. U. Brandes, D. Delling, M. Gaertler, R. Grke, M. Hoefer, Z. Nikoloski, and D. Wagner. On finding graph clusterings with maximum modularity. In *proc 33rd intl workshop graph-theoretic concepts in computer science (WG07)*, 2007.
7. V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *J. Stat. Mech.*, 2008.
8. P. Pons and M. Latapy. Computing communities in large networks using random walks. *J. of Graph Alg. and App. bf*, 10:284–293, 2004.
9. M. E. J. Newman. Modularity and community structure in networks. *Proc. Natl. Acad. Sci. USA*, 103(23):85778582, 2006.
10. Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Phys. Rev. E*, 72(2):027104, Aug 2005.
11. M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74(3):036104, Sep 2006.
12. P. Schuetz and A. Caflisch. Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 77(4):046112, 2008.
13. Michael R. Anderberg. *Cluster analysis for applications*. Academic Press, New York, 1973.
14. M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69(6):066133, Jun 2004.
15. Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70(6):066111, Dec 2004.
16. Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *Lecture Notes in Computer Science*, 3918(1973):29, 2011.
17. Andreas Noack and Randolf Rotta. Multi-level algorithms for modularity clustering. In *Proceedings of the 8th International Symposium on Experimental Algorithms*, SEA '09, pages 257–268, Berlin, Heidelberg, 2009. Springer-Verlag.
18. K. Wakita and T. Tsurumi. Finding community structure in mega-scale social networks. *eprint arXiv:cs/0702048*, February 2007.
19. L. Danon, A. Díaz-Guilera, and A. Arenas. The effect of size heterogeneity on community identification in complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, 11:10, November 2006.
20. W.H.E. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1(1):7–24, 1984.
21. Alexandre P. Francisco and Arlindo L. Oliveira. On community detection in very large networks. In Luciano F. Costa, Alexandre Evsukoff, Giuseppe Mangioni, and Ronaldo Menezes, editors, *Complex Networks*, volume 116 of *Communications in Computer and Information Science*, pages 208–216. Springer Berlin Heidelberg, 2011.
22. S. Lehmann and L.K. Hansen. Deterministic modularity optimization. *Eur. Phys. J. B*, 60(1):83–88, 2007.

Chapter 3

Self-Organizing Maps by Difference of Convex functions optimization

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

Self-Organizing Maps by Difference of Convex functions optimization

Hoai An LE THI · Manh Cuong NGUYEN

Revised version to the Data Mining and Knowledge Discovery journal (DAMI).
Received: date / Accepted: date

Abstract We offer an efficient approach based on Difference of Convex functions (DC) optimization for Self-Organizing Maps (SOM). We consider SOM as an optimization problem with a nonsmooth, nonconvex energy function and investigated DC Programming and DC Algorithm (DCA), an innovative approach in nonconvex optimization framework to effectively solve this problem. Furthermore an appropriate training version of this algorithm is proposed. The numerical results on many real-world datasets show the efficiency of the proposed DCA based algorithms on both quality of solutions and topographic maps.

Keywords Self-Organizing Maps, DC Programming, DCA.

1 Introduction

The Self-Organizing Map (SOM), introduced by Kohonen in 1982 [33], and its variants are a popular artificial neural network approaches in unsupervised learning. The principal goal of an SOM is to transform an incoming signal pattern of arbitrary dimension into a low (usually one or two)-dimensional discrete map, and to perform this transformation adaptively in a topologically ordered fashion. Through their intrinsic properties, such as preserving topological relationships between input data, and sharing similarities between nearby neurons in an ordered network, the SOM is an elegant way to interpret complex data and an excellent tool for data visualization and exploratory cluster analysis. SOM is primarily a data-driven dimensionality reduction and data compression method. In data mining and machine learning, SOM is a widely used tool in the exploratory phase for various technique such as data clustering ([2], [52], [55], [66], [68], [72]), data classification ([62], [7], [65]) and graph mining ([51], [14], [71], [21], [23]).

Since its invention ([33]), the SOM is used as a standard analytical tool in many fields of sciences: statistics, data mining, signal processing, control theory, financial analyses,

Hoai An LE THI · Manh Cuong NGUYEN
Laboratory of Theoretical and Applied Computer Science - LITA EA 3097,
University of Lorraine, Ile du Saulcy, 57045 Metz, France.
hoai-an.le-thi@univ-lorraine.fr
manh-cuong.nguyen@univ-lorraine.fr

experimental physics, chemistry and medicine, and recently in organization of very large document collections, social network, biological brain function. A variety of applications and real-work problems use the SOM technique, among them image segmentation, vector quantization and image compression, density modeling, gene expression analysis, text mining and information management, data visualization (see [78] and references therein), object classification, skin detection, learning robot behaviors, learning the motion map, object recognition, map building and coordination of movements (see e.g. [1,46,4,8,56,75,77]), recommender systems (see [19,45,63,73,13]), web-based social network ([13,75,73]), etc.

The SOM algorithm was originally proposed by Kohonen on the grounds of biological plausibility and was formulated as a simple iterative learning procedure based on a neural network representation ([33,34]). Using a set of m neurons, often arranged in a 2-dimensional rectangular or hexagonal grid, each of them is associated with a position (node) and a weight vector in the same space as input data, the SOM algorithm aims to form a discrete topological mapping of an input data. A topology is defined on this neurons set, through a $m \times m$ neighborhood matrix and via a decreasing function ϕ^t of the distance between positions computed in terms of a temperature parameter σ at the learning step t, e.g.

$$\phi^t_{l,k} = \exp\left(-\frac{\left\|x^t_l - x^t_k\right\|^2}{2\sigma^2}\right). \tag{1}$$

Starting by attributing small random numbers for the weight vectors x_l associated to neuron l, the SOM algorithm iteratively modifies the weight vectors until the map convergence. The original SOM algorithm [33] is given by (x^t_l stands for the weight vector x_l at the step t)

$$x^{t+1}_l = x^t_l + \alpha_t \phi^t_{l,k^*}(a_t - x^t_l), \quad k^* = \arg\min_{l=1\ldots m}\left\{\|a_t - x_l\|^2\right\} \tag{2}$$

where t is a learning step, a_t is a data point, the neighborhood term ϕ^t_{l,k^*} and the learning rate α_t decrease during the learning procedure; k^* denotes the winning node, i.e. a_t nearest neuron in terms of weight.

Since its introduction, various SOM algorithms have been developed. Two main approaches are often used - sequential and batch, in SOM algorithms. In the traditional sequential training, samples are presented to the map one at a time, and the algorithm gradually moves the weight vectors towards them. In the batch training, the data set is presented to the SOM as a whole, and the new weight vectors are weighted averages of the data vectors. The sequential approach can provides a good topographic maps. However, it strongly depends on input order of the data and is expensive from a computational point of view. Most of recent works are Batch SOM algorithms and based on Simulated Annealing. For example, Hiroshi Dozono et al. [15] presented a Batch SOM algorithm using Simulated Annealing in the batch update phase and applied in sequence analysis to extract features of DNA sequences. Fiannaca et al. [16] proposed a fast learning method for Batch SOM based on Simulate Annealing to adapt the learning parameters in order to avoid local minima. This training technique is believed to be faster than the standard batch training. Haruna Matsushita and Yoshifumi Nishio [53] proposed a version of Batch-Learning SOM with weighted connections to avoid false-neighbor effects.

An alternative way to learning SOM is based on optimization techniques. The idea is to choose an energy function of which the minimizer yields the optimal set of parameters corresponding to the desired topographic mapping.

Graepel et al. [6] proposed an efficient deterministic annealing scheme called Soft Topographic Vector Quantization (STVQ) to minimize this energy function. Later, in [20] these authors considered an alternative mathematical formulation that is

$$\min_{\substack{u_{ik}\in\{0,1\}\\ i=1,...,n,\\ k=1,...,m}} \left\{ \sum_{i=1}^{n}\sum_{k=1}^{m} u_{ik} \sum_{l=1}^{m} \phi_{l,k}^{t} \|x_k - a_i\|^2 : \sum_{k=1}^{m} u_{ik} = 1, i = 1...,n \right\}, \tag{3}$$

where u_{ik} are binary assignment variables which take the value $u_{ik} = 1$ if the data vector a_i is assigned to node k and $u_{ik} = 0$ otherwise, and the neighborhood function obeys $\sum_{l=1}^{m} \phi_{l,k}^{t} = 1$ $\forall k$. They proposed a STVQ algorithm and a generalization of STVQ using kernel functions for solving(3).

Heskes [26] showed that the learning rule (2) cannot be derived from an energy function when the data follow a continuous distribution, and proposed an energy function that leads to a slightly different map, which still fulfills the topographic clustering aims. Later, in [27] Heskes explored the links between SOM, vector quantization, and mixture modeling and derived an EM based algorithm for the mixture model of Batch SOM.

Let $A \in \mathbb{R}^{n\times d}$ be a matrix whose ith row is the vector $a_i \in \mathbb{R}^d$, for $i = 1,...,n$ and let $\mathscr{X} := \{x_1,...,x_m\}$ be the set of m weight vectors in $\mathbb{R}^d$. A is the dataset of SOM and x_i ($i \in [1..m]$) are the weight vectors associated with map nodes. Let ϕ^t be a neighborhood function that specifies the influence of nodes at a learning step t. A measure of compatibility between the observation a_i and the weight vector x_l is defined as follows:

$$\delta_{a_i,l}^{t} = \sum_{k=1}^{m} \phi_{l,k}^{t} \|x_k - a_i\|^2, \tag{4}$$

and the distances d_i from observation a_i to the best matching neuron is given by

$$d_i = \min_{l=1...m} \delta_{a_i,l}^{t}.$$

Here the winning node is not only nearest neighbor as in the original SOM, but it reflects the resemblance to neighbor nodes.

The standard approaches of Batch SOM proposed to train network by minimizing, at each step t, the energy function which is the sum of all the distances d_i from observation a_i to the best matching neuron:

$$\text{(BSOM)} \quad \min_{X\in\mathbb{R}^{m\times d}} \left\{ E^t(X) := \frac{1}{m} \sum_{i=1}^{n} \min_{l=1...m} \sum_{k=1}^{m} \phi_{l,k}^{t} \|x_k - a_i\|^2 \right\}. \tag{5}$$

Actually, the SOM is trained iteratively with a monotonous decrease of σ. A cooling schedule is used to indicate how σ decreases in time. This approach is included in the family of Batch SOM algorithms and the model (5) is named Batch SOM model or BSOM in short.

Since its introduction, several optimization methods have been investigated for solving BSOM. The objective function of BSOM is nonsmooth and nonconvex, hence it is very difficult to handle BSOM, in particular in large-scale setting. Due to the nonconvexity of BSOM, the algorithms often fall into local minima and sometimes provide the inferior topology.

As mentioned in [17], comparing with the original SOM, the BSOM has several advantages: it is simple in computation, fast, and gives a better final distortion. Moreover, there is no adaptation parameter to tune, and the results are "deterministic reproducible".

Meanwhile, the BSOM has some drawbacks : the results are quite sensitive to the initialization map, as a consequence, the algorithm can conduct to a bad organization map, and too unbalanced classes. On the contrary, the original SOM is more or less insensitive to the initialization, at least in terms of organization and neighborhood relations.

While several heuristic methods have been investigated, there is a few deterministic approaches for solving BSOM. In this paper, we tackle BSOM by an innovative optimization approach in nonconvex programming framework, namely Difference of Convex functions (DC) Programming and DC Algorithm (DCA). Our work is 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 for which they provide quite often a global solution and proved to be more robust and efficient than the standard methods (see, e.g. [37] - [47], [9], [67], [76] and the list of references in [36]).

Capitalizing on DCA, we aim to design a fast and scalable DCA based algorithm for solving the BSOM problem so that the algorithm gives not only a good classification but also a well organized map. In other words, it should overcome the drawbacks of both original SOM et BSOM algorithms and realizes a good trade-off between the rapidity and scalability on one hand and the quality of map as well as of the classification on another hand.

Our contributions are two-fold. Firstly, using elegant matrix representations we reformulate the BSOM problem as a DC program and develop a simple DCA scheme, named DCASOM, which is very computationally inexpensive: DCASOM requires only sums and scalar multiplications of vectors. Secondly, we propose a training procedure for the DCASOM with an appropriate cooling schedule. According to the efficiency of DCASOM, our training DCASOM needs only a few iterations to get the map convergence, hence we adapt a suitable cooling schedule to reduce rapidly the temperature.

The rest of the paper is organized as follows. Section 2 is devoted to the DC programming and DCA for solving BSOM while Section 3 presents a training version of DCASOM. Numerical experiments are reported in Section 4 and Section 5 concludes the paper.

2 Solving the BSOM model by DC Programming and DCA

2.1 Outline of DC Programming and DCA

DC programming and DCA which have been introduced by Pham Dinh Tao in 1985 and extensively developed by Le Thi Hoai An and Pham Dinh Tao since 1994 (see e.g. [38],[60]) constitute the backbone of smooth/nonsmooth nonconvex programming and global optimization. They address the problem of minimizing a function f which is the difference of two convex functions on the whole space $\mathbb{R}^d$ or on a convex set $C \subset \mathbb{R}^d$.

A function $f : \mathbb{R}^n \longrightarrow \mathbb{R} \cup \{+\infty\}$ is lower semicontinuous (lsc) at x_0 if $\forall \varepsilon > 0\ \exists \eta > 0$ such that $\|x - x_0\| \leq \eta \Longrightarrow f(x) > f(x_0) - \varepsilon$. f is lsc on $\mathbb{R}^n$ if f is lsc at every point of $\mathbb{R}^n$. Denote by $\Gamma_0(\mathbb{R}^d)$ the "convex cone" of all proper lower semicontinuous convex functions on $\mathbb{R}^d$. Generally speaking, a DC program is an optimization problem of the form :

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

with $g, h \in \Gamma_0(\mathbb{R}^d)$. Such a function f is called a DC function, and $g - h$ a DC decomposition of f while g and h are the DC components of f. The convex constraint $x \in C$ can be incorporated in the objective function of (P_{dc}) by using the indicator function on C denoted by χ_C which is defined by $\chi_C(x) = 0$ if $x \in C$, and $+\infty$ otherwise:

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

$$= \inf\{\chi_C(x) + g(x) - h(x) : x \in \mathbb{R}^d\}.$$

The vector space of DC functions on $\mathbb{R}^d$, denoted by $\mathscr{DC}(\mathbb{R}^d) := \Gamma_0(\mathbb{R}^d) - \Gamma_0(\mathbb{R}^d)$,is quite large to contain almost real life objective functions and is closed under all the operations usually considered in optimization.

Let us denote $\langle x, y\rangle$ the Euclidean inner product of vectors x and y. Let

$$g^*(y) := \sup\{\langle x, y\rangle - g(x) : x \in \mathbb{R}^d\}$$

be the conjugate function of a convex function g. Then, the following program is called the dual program of (P_{dc}):

$$\alpha_D = \inf\{h^*(y) - g^*(y) : y \in \mathbb{R}^d\}. \qquad (D_{dc})$$

One can prove that $\alpha = \alpha_D$, and there is the perfect symmetry between primal and dual DC programs: the dual to (D_{dc}) is exactly (P_{dc}) ([59]).

For a convex function θ, the subdifferential of θ at $x_0 \in \text{dom } \theta := \{x \in \mathbb{R}^d : \theta(x_0) < +\infty\}$, denoted by $\partial\theta(x_0)$, is defined by

$$\partial\theta(x_0) := \{y \in \mathbb{R}^d : \theta(x) \geq \theta(x_0) + \langle x - x_0, y\rangle, \forall x \in \mathbb{R}^d\}. \qquad (6)$$

The subdifferential $\partial\theta(x_0)$ generalizes the derivative in the sense that θ is differentiable at x_0 if and only if $\partial\theta(x_0) \equiv \{\bigtriangledown_x\theta(x_0)\}$. Recall the well-known property related to subdifferential calculus of a convex function θ:

$$y_0 \in \partial\theta(x_0) \Longleftrightarrow x_0 \in \partial\theta^*(y_0) \Longleftrightarrow \langle x_0, y_0\rangle = \theta(x_0) + \theta^*(y_0). \qquad (7)$$

The complexity of DC programs resides, of course, in the lack of practical global optimality conditions. Local optimality conditions are then useful in DC programming.

A point x^* is said to be a local minimizer of $g - h$ if $g(x^*) - h(x^*)$ is finite and there exists a neighbourhood U of x^* such that

$$g(x^*) - h(x^*) \leq g(x) - h(x), \ \ \forall x \in \mathscr{U}. \qquad (8)$$

The necessary local optimality condition for (primal) DC program (P_{dc}) is given by

$$\emptyset \neq \partial h(x^*) \subset \partial g(x^*). \qquad (9)$$

The condition (9) is also sufficient (for local optimality) in many important classes of DC programs (see [38]).

A point x^* is said to be a critical point of $g - h$ if

$$\partial h(x^*) \cap \partial g(x^*) \neq \emptyset. \qquad (10)$$

The relation (10) is in fact the generalized KKT condition for (P_{dc}) and x^* is also called a generalized KKT point.

2.1.1 Philosophy of DCA

Based on local optimality conditions and duality in DC programming, the DCA consists in constructing of two sequences $\{x^k\}$ and $\{y^k\}$ of trial solutions of the primal and dual programs respectively, such that the sequences $\{g(x^k)-h(x^k)\}$ and $\{h^*(y^k)-g^*(y^k)\}$ are decreasing, and $\{x^k\}$ (resp. $\{y^k\}$) converges to a primal feasible solution x^* (resp. a dual feasible solution y^*) satisfying local optimality conditions and

$$x^* \in \partial g^*(y^*), \quad y^* \in \partial h(x^*). \tag{11}$$

It implies, according to (7) that $x^*$and y^* are critical points of $g-h$ and h^*-g^* respectively.

The main idea behind DCA is to replace in the primal DC program (P_{dc}), at the current point x^k of iteration k, the second component h with its affine minorization defined by

$$h_k(x) := h(x^k) + \langle x-x^k, y^k\rangle, \;\; y^k \in \partial h(x^k)$$

to give birth to the primal convex program of the form

$$(P_k) \;\; \inf\{g(x)-h_k(x) : \; x\in \mathbb{R}^n\} \Leftrightarrow \inf\{g(x)-\langle x, y^k\rangle : \; x\in \mathbb{R}^d\}$$

the optimal solution of which determines x^{k+1}.

Dually, a solution x^{k+1} of (P_k) is then used to define the dual convex program (D_{k+1}) obtained from (D_{dc}) by replacing g^* with its affine minorization defined by

$$(g^*)_k(y) := g^*(y^k) + \langle y-y^k, x^{k+1}\rangle, \;\; x^{k+1} \in \partial g^*(y^k)$$

to obtain the convex program

$$(D_{k+1}) \quad \inf\{h^*(y) - [g^*(y^k) + \langle y-y^k, x^{k+1}\rangle] : y\in \mathbb{R}^d\}$$

the optimal solution of which determines y^{k+1}. The process is repeated until convergence.

DCA performs so a double linearization with the help of the subgradients of h and g^*. According to relation (7) it is easy to see that the optimal solution set of (P_k) (resp. (D_{k+1})) is nothing but $\partial g^*(y^k)$ (resp. $\partial h(x^k)$). Hence, we can say that DCA is an iterative primal-dual subgradient method that yields the next scheme: (starting from given $x^0 \in \text{dom}\, \partial h$)

$$y^k \in \partial h(x^k); \quad x^k \in \partial g^*(y^k), \; \forall k \geq 0. \tag{12}$$

A deeper insight into DCA has been described in [38]. The generic DCA scheme is shown below.

DCA scheme

Initialization: Let $x^0 \in \mathbb{R}^d$ be a guess, $k \leftarrow 0$.

Repeat

- Calculate $y^k \in \partial h(x^k)$
- Calculate $x^{k+1} \in \partial g^*(y^k)$, which is equivalent to

$$x^{k+1} \in arg\min\{g(x)-h(x^k)-\langle x-x^k, y^k\rangle : x\in \mathbb{R}^d\} \quad (P_k)$$

- $k \leftarrow k+1$

Until convergence of $\{x^k\}$.

2.1.2 DCA's convergence properties

Convergence properties of DCA and its theoretical basis can be found in ([37,38,60]). For instance it is important to mention that

- DCA is a descent method, without linesearch but with global convergence.
- If the optimal value α of problem (P_{dc}) is finite and the infinite 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^*), i.e. $\partial h(x^*)\cap\partial g(x^*)\neq\emptyset$ (resp. $\partial h^*(y^*)\cap\partial g^*(y^*)\neq\emptyset$).
- DCA has a linear convergence for general DC programs.

The construction of DCA involves DC components g and h but not the 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 infinitely many DC decompositions which have crucial impacts on the qualities (speed of convergence, robustness, efficiency, globality of computed solutions,. . .) of DCA, the search of a "good" DC decomposition is important from an algorithmic point of views. How to develop an efficient algorithm based on the generic DCA scheme for a practical problem is thus a judicious question to be studied, the answer depends on the specific structure of the problem being considered.

The search of a good DC decomposition should be based on the structure of the DC function f and the closed convex constraint set C, there is no common formula of DC decomposition for general DC programs. In this regard, let us point out below two examples of interesting DC decompositions.

2.1.3 Examples of DC decompositions

Consider the constrained DC program of the form

$$\inf\{f(x) : x \in \mathrm{C}\} \quad (P)$$

where $C \subset \mathbb{R}^d$ is a nonempty closed convex set and f is a real-valued twice continuously differentiable function on $\mathbb{R}^d$. This formulation covers many problems in Machine Learning.

Let $\lambda_1(x)$ (resp. $\lambda_n(x)$) the smallest (resp. largest) eigenvalue of $\nabla^2 f(x)$. It is easy to verify that

a) For all $\eta \geq \max\{0, \max\{-\lambda_1(x),\ x \in C\}\}$ the functions $\frac{1}{2}\eta\|x\|^2$ and $\frac{1}{2}\eta\|x\|^2 + f(x)$ are convex on C.
b) For all $\rho \geq \max 0, \max\{\lambda_n(x),\ x \in C\}\}$ the functions $\frac{1}{2}\rho\|x\|^2$ and $\frac{1}{2}\rho\|x\|^2 - f(x)$ are convex on C.

It is clear that such η and ρ exist if C is bounded. More generally, if f is differentiable with Lipschitz derivative on C, then there are also η and ρ satisfying a) and b) respectively.

Hence we have the following two DC decompositions of f:

$$g(x) := \chi_C(x) + \frac{1}{2}\eta\|x\|^2 + f(x); \quad h(x) := \frac{1}{2}\eta\|x\|^2 \tag{13}$$

and

$$g(x) := \chi_C(x) + \frac{1}{2}\rho\|x\|^2; \quad h(x) := \frac{1}{2}\rho\|x\|^2 - f(x). \tag{14}$$

The DCA applied to (P) with decomposition (13) and/or (14) can be described as follows:

Algorithm DCAP1: Let x^0 be given in $\mathbb{R}^d$. Set $k \leftarrow 0$.

Repeat

- Calculate x^{k+1} by solving the convex program

$$\min\left\{f(x)+\frac{1}{2}\eta\|x-x^k\|^2 : x\in C\right\}, \tag{15}$$

- $k \leftarrow k+1$

Until convergence of $\{x^k\}$.

One recognizes in (15) the extension of the proximal point algorithm to nonconvex programming.

Algorithm DCAP2: Let x^0 be given in $\mathbb{R}^d$. Set $k \leftarrow 0$.

Repeat

- Calculate $y^k = \nabla\left(\frac{1}{2}\rho\|.\|^2 - f(.)\right)(x^k) = \rho x^k - \nabla f(x^k)$
- Calculate x^{k+1} by solving the convex program

$$\min\left\{\frac{1}{2}\rho\|x\|^2 - \langle x, \rho x^k - \nabla f(x^k)\rangle : x\in C\right\}, \tag{16}$$

  i.e. $x^{k+1} = Proj_C(x^k - \frac{1}{\rho}\nabla f(x^k))$.
- $k \leftarrow k+1$

Until convergence of $\{x^k\}$.

Here, $Proj_C$ stands for the orthogonal projection on C. Note that the resulting convex program (P_k) is actualy given by

$$(P_k) \quad \min\{\langle x, \nabla f(x^k)\rangle + \frac{1}{2}\rho\|x\|^2 + \frac{1}{2}\rho\|x^k\|^2 - \rho\langle x, x^k\rangle + f(x^k) - \langle x^k, \nabla f(x^k)\rangle : x\in C\}.$$

That can be reformulated in the following two forms

$$\min\{\langle x, \nabla f(x^k)\rangle + \frac{1}{2}\rho\|x-x^k\|^2 + f(x^k) - \langle x^k, \nabla f(x^k)\rangle : x\in C\}, \tag{17}$$

that is equivalent to (15) and

$$\min\{\frac{1}{2}\rho\|x - (x^k - \frac{1}{\rho}\nabla f(x^k))\|^2 + f(x^k) - \frac{1}{2\rho}\|\nabla f(x^k)\|^2 : x\in C\}, \tag{18}$$

which is equivalent to (16).

Hence DCAP2 is exactly the proximal regularized subgradient algorithm (via (17)) or the subgradient projection algorithm (via (18)) for nonconvex programming.

As indicated above, we are greatly interested in the choice of DC decompositions: what is "the best" among (13) and (14) ? The answer depends on both specific structures of C and f. In fact, the performance of the DCA depends upon that of the algorithm for solving convex programs (15) and (16). For certain problems, for example, box constrained quadratic programming and ball constrained quadratic programming, Algorithm DCAP2 is greatly less expensive than Algorithm DCAP1, because the orthogonal projection onto C in

these cases is given in explicit form (see [60]). In practice, when f is differentiable and the projection on C can be inexpensively determined, the use of DCAP2 is very recommended.

For a complete study of DC programming and DCA the reader is referred to [37,38,59, 60], and the references therein. The solution of a nonconvex program (P_{dc}) by DCA must be composed of two stages: the search of an appropriate DC decomposition of f and that of a good initial point.

It is worth pointing out that, with suitable DC decompositions, DCA generates most of standard methods in convex and nonconvex programming. Again, one recovers a version of DCA in standard methods widely used by the research community in machine learning. As mentioned in [44], the three well-known methods EM (Expectation-Maximization) ([12]), SLA (Succesive Linear Approximation) ([5]) and CCCP (Convex-Concave Procedure) ([79]) are particular cases of DCA. In addition, these three methods, without proof of convergence, relate only differentiable functions.

In the last years DCA has been successfully applied in several studies in Machine Learning e.g., for SVM-based Feature Selection [40], for improving boosting algorithms [35], for implementing-learning [47,67,48], for Transductive SVMs [9], for unsupervised clustering [39,41,42,44], and for diversity data mining [43], etc (see [36] for a more complete list of references).

In this paper, the objective function E^t of (BSOM) is not differentiable and the above DC decompositions can not be used. Despite the non-differentiability of E^t, exploiting its special structure we can achieve a suitable and natural DC decomposition that results in a simple and elegant DCA scheme as shown below.

2.2 A DC formulation of (BSOM)

In what follows we will use the matrix presentation to simplify related computations in our algorithms.

The variables are then $X \in \mathbb{R}^{m\times d}$ whose i^{th} row X_i is equal to the vector $x_i \in \mathbb{R}^d$, for $i = 1,2,...,m$. Let β be a vector in $\mathbb{R}^m$ and denote by $D(\beta)$ the corresponding diagonal matrix, say the ith diagonal term of this matrix is ith component of β. The Euclidean structure of the matrix vector space $\mathbb{R}^{m\times d}$ is defined with the help of the Frobenius product:

$$\langle X,Y\rangle = Tr(X^TY) = \sum_{i=1}^{m}\langle X_i,Y_i\rangle, \tag{19}$$

and its Euclidean norm $\|X\|^2 = \sum_{i=1}^m \langle X_i,X_i\rangle = \sum_{i=1}^m \|X_i\|^2$. Hence,

$$\sum_{i=1}^{m}\beta_i\langle X_i,Y_i\rangle = \sum_{i=1}^{m}\langle X_i,\beta_iY_i\rangle = \langle X,D(\beta)Y\rangle,$$

and

$$\sum_{i=1}^{m}\beta_i\langle X_i,X_i\rangle = \langle X,D(\beta)X\rangle.$$

Let us consider now the BSOM model, for a given learning step t, in an equivalent formulation where $\frac{1}{m}$ is replaced by $\frac{1}{2}$ for simply the computations in DCA:

$$\text{(BSOM)}\quad \min_{X\in\mathbb{R}^{m*d}}\left\{F^t(X) := \frac{1}{2}\sum_{i=1}^{n}\min_{l=1...m}\sum_{k=1}^{m}\phi_{l,k}^t\|x_k-a_i\|^2\right\}. \tag{20}$$

In (BSOM) the neighborhood function ϕ^t is computed in term of t according to the initial map (at the beginning of each learning step t), say the term $\phi^t_{l,k}$ is given in each energy function $F^t(X)$ to be minimized by DCA.

We first find a DC decomposition of $F^t(X)$. For simple presentation we write the function $F^t(X)$ in the form

$$F^t(X) := \frac{1}{2}\sum_{i=1}^{n} \min_{l=1\ldots m} \delta^t_{a_i,l}(X)$$

where

$$\delta^t_{a_i,l}(X) := \sum_{k=1}^{m} \phi^t_{l,k}\|x_k - a_i\|^2.$$

Since

$$\min_{l=1\ldots m} \delta^t_{a_i,l}(X) = \sum_{r=1}^{m} \delta^t_{a_i,r}(X) - \max_{l=1\ldots m} \sum_{r\neq l} \delta^t_{a_i,r}(X),$$

we have

$$F^t(X) \ := \frac{1}{2}\sum_{i=1}^{n} \min_{l=1\ldots m} \delta^t_{a_i,l}(X) \tag{21}$$

$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{r=1}^{m} \delta^t_{a_i,r}(X) - \frac{1}{2}\sum_{i=1}^{n} \max_{l=1\ldots m} \sum_{r\neq l} \delta^t_{a_i,r}(X). \tag{22}$$

From the definition, the function $\delta^t_{a_i,r}(X)$ is convex. Consequently the functions

$$G^t(X) := \frac{1}{2}\sum_{i=1}^{n}\sum_{r=1}^{m} \delta^t_{a_i,r}(X) \tag{23}$$

and

$$H^t(X) := \frac{1}{2}\sum_{i=1}^{n} \max_{l=1\ldots m} \sum_{r\neq l} \delta^t_{a_i,r}(X) \tag{24}$$

are convex. Hence the following DC decomposition of $F^t(X)$ seems to be natural:

$$F^t(X) = G^t(X) - H^t(X). \tag{25}$$

Let ϕ^t_+ be a vector in $\mathbb{R}^m$ such that $\left(\phi^t_+\right)_k = \sum_{r=1}^m \phi^t_{r,k}$. We have, after simple calculations:

$$G^t(X) = \frac{n}{2}\langle X, D(\phi^t_+)X\rangle - n\langle X, D(\phi^t_+)\overline{A}\rangle + \frac{1}{2}\left(\sum_{k=1}^{m}\phi^t_{+k}\right)\|A\|^2, \tag{26}$$

where $\overline{A} \in \mathbb{R}^{m\times d}$ is given by $\overline{A}_i := \frac{1}{n}\sum_{j=1}^n a_j$. Hence $G^t(X)$ is a strongly convex quadratic function. This structure allows us to design a very simple DCA scheme.

Determining the DCA scheme applied to (25) amounts to computing the two sequences $\{X^s\}$ and $\{Y^s\}$ in $\mathbb{R}^{m\times d}$ such that $Y^s \in \partial H^t(X^s), X^{s+1} \in \partial(G^t)^*(Y^s)$. We shall present below the computation of $\partial H^t(X)$ and $\partial(G^t)^*(Y)$.

- **Calculation of ∂H^t**

For $i = 1,...,n$ and $l = 1,...,m$ let $H^t_{i,l}$ and H^t_i be the functions defined by

$$H^t_{i,l}(X) := \sum_{r \neq l} \delta^t_{a_i,r}(X),\ H^t_i(X) = \max_{l=1...m} H^t_{i,l}(X). \tag{27}$$

Hence

$$H^t(X) = \frac{1}{2}\sum_{i=1}^{n} H^t_i(X) \tag{28}$$

and then

$$Y \in \partial H^t(X) \Leftrightarrow Y = \frac{1}{2}\sum_{i=1}^{n} Y^{[i]} \text{ with } Y^{[i]} \in \partial H^t_i(X) \text{ for } i = 1,...,n.$$

Let $K_i(X) = \{l_i \in \{1,...,m\} : |H^t_{i,l_i}(X) = H^t_i(X) := \max_{l=1...m} H^t_{i,l}(X)\}$. Then the subdifferential of H^t_i is computed as ([28]):

$$\partial H^t_i(X) = co\{\cup_{l_i \in K_i(X)} \partial H^t_{i,l_i}(X)\}, \tag{29}$$

where *co* stands for the convex envelope.

For computing $\partial H^t_{i,l}$ we express the convex function $H^t_{i,l}(X)$ as

$$H^t_{i,l}(X) := \sum_{k=1}^{m} \left(\sum_{r \neq l} \phi^t_{r,k} \right) \|x_k - a_i\|^2 = \sum_{k=1}^{m} \overline{\phi^t_l} \|x_k - a_i\|^2, \tag{30}$$

where $\overline{\phi^t_l} \in \mathbb{R}^m$ is given by

$$\left(\overline{\phi^t_l}\right)_k := \sum_{r \neq l} \phi^t_{r,k} = \phi^t_{+k} - \phi^t_{l,k}. \tag{31}$$

Hence, $H^t_{i,l}$ is differentiable and

$$\nabla H^t_{i,l}(X) = 2D(\overline{\phi^t_l})(X - A^{[i]}), \tag{32}$$

where $A^{[i]} \in \mathbb{R}^{m \times d}$ is the matrix whose rows are all equal to a_i. According to (29) $Y^{[i]} \in \partial H^t_i(X)$ is computed by

$$Y^{[i]} = \sum_{l_i \in K_i(X)} \lambda^i_l \nabla H^t_{i,l}(X) \tag{33}$$

with $\lambda_{l_i} \geq 0$ and $\sum_{l_i \in K_i(X)} \lambda_{l_i} = 1$. In particular we can chose, for a $l^*_i \in K_i(X)$, $\lambda_{l^*_i} = 1$ and $\lambda_{l_i} = 0\ \forall \lambda_{l_i} \in K_i(X) : l_i \neq l^*_i$, and then

$$Y^{[i]} = \nabla H^t_{i,l}(X) = 2D(\overline{\phi^t_{l^*_i}})(X - A^{[i]}).$$

Finally, at each iteration s, $Y^s \in \partial H(X^s)$ is computed as

$$Y^s = \sum_{i=1}^{n} D(\overline{\phi^t_{l^*_i}})(X^s - A^{[i]}). \tag{34}$$

- **Calculation of ∂G^***

Since G^t is a strongly convex quadratic function and its conjugate $(G^t)^*$ is differentiable, we deduce from (7) that

$$\begin{aligned} X^{s+1} &= \nabla(G^t)^*(Y^s) \Leftrightarrow Y^s = \nabla G^t(X^{s+1}) \\ &\Leftrightarrow Y^s = nD(\phi_+^t)X^{s+1} - nD(\phi_+^t)\overline{A}). \end{aligned} \tag{35}$$

Therefore

$$= D\left(\frac{(\phi_+^t)^{-1}}{n}\right) Y^s + \overline{A}. \tag{36}$$

More precisely, the $k-th$ row of X^{s+1} is determined as

$$X_k^{s+1} = \frac{1}{n}(\phi_+^t)_k Y_k^s + \overline{A}_k.$$

According to (34) and (31) we have

$$X_k^{s+1} = \frac{1}{n(\phi_+^t)_k}\left[\sum_{i=1}^n \left((\phi_+^t)_k - (\phi_{l_i^*,k}^t)_j\right)(X_k^s - a_i)\right] + \frac{1}{n}\sum_{i=1}^n a_i$$

or again

$$X_k^{s+1} = X_k^s + \frac{\sum_{i=1}^n (\phi_{l_i^*,k}^t)(a_i - X_k^s)}{n(\phi_+^t)_k}. \tag{37}$$

Thus DCA for solving the BSOM problem consists in determining, at each iteration s,

$$l_i^* = \arg\max_{l=1,\ldots,m} H_{i,l}^t(X^s) = \arg\min_{l=1,\ldots,m} \sum_{k=1}^m \phi_{l,k}^t \|x_k^s - a_i\|^2, \text{ for each } i = 1,\ldots,m,$$

and then computing X^{s+1} following (37). The algorithm can be described as follows

- **DCA for BSOM model (t fixed)**

DCASOM
initializations: Let $\varepsilon > 0$ be given, X^0 be an initial point in $\mathbb{R}^{m\times d}$ and set $s := 0$.
Compute the neighborhood terms $\phi_{l,k}^t$ for $l,k = 1,\ldots,m$.
repeat
1. Determine

$$l_i^* = \arg\min_{l=1,\ldots,m} \sum_{k=1}^m \phi_{l,k}^t \|X_k^s - a_i\|^2, \text{ for each } i = 1,\ldots,m,$$

2. Calculate X_k^{s+1}, for $k = 1,\ldots,m$, as

$$X_k^{s+1} = X_k^s + \frac{\sum_{i=1}^n (\phi_{l_i^*,k}^t)(a_i - X_k^s)}{n(\phi_+^t)_k}.$$

3. $s = s+1$.
until $\| X^{s+1} - X^s \| \leq \varepsilon \| X^s \|$.

Discussion: the choice of the DC decomposition $G - H$ in (25) allows us to interpret the DCASOM in an interesting point of view. First, we observe that the first step of DCASOM

algorithm is nothing else searching the winning nodes of each a_i, while the computation of X^{s+1} can be regarded as

$$\begin{aligned} X_k^{s+1} &= \left(1-\frac{\sum_{i=1}^n \phi^t_{l_i^*,k}}{n(\phi^t_+)_k}\right)X_k^s + \frac{\sum_{i=1}^n \phi^t_{l_i^*,k}a_i}{n(\phi^t_+)_k} \\ &= \left(1-\frac{\sum_{i=1}^n \phi^t_{l_i^*,k}}{n(\phi^t_+)_k}\right)X_k^s + \frac{\sum_{i=1}^n \phi^t_{l_i^*,k}}{n(\phi^t_+)_k}\cdot\frac{\sum_{i=1}^n \phi^t_{l_i^*,k}a_i}{\sum_{i=1}^n \phi^t_{l_i^*,k}} = (1-\alpha)X_k^s + \alpha X^{new}_{centre-k}, \end{aligned} \tag{38}$$

where $\alpha := \frac{\sum_{i=1}^n \phi^t_{l_i^*,k}}{n(\phi^t_+)_k} < 1$ and $X^{new}_{centre-k}$ stands for the gravity centre of the new cluster k after the step 1 (it is exactly the solution given by the step 2 of the basic Bat SOM algorithm, i.e. a K-means like algorithm for BSOM model). Thus, from (38) we see that X_k^{s+1} is in fact a convex combination of X_k^s and $X^{new}_{centre-k}$. On another hand, by expressing $X_k^{s+1} = X_k^s + \alpha(X^{new}_{centre-k} - X_k^s)$ we see that the sequence X_k^s constructed by DCASOM follows the displacement direction $(X^{new}_{centre-k} - X_k^s)$. This difference between DCASOM and a K-means like algorithm comes from the fact that, at the step of updating the new centres, instead of minimizing the objective $F^t(X)$ as K-mean, DCASOM minimizes the convex majorant function $G^t(X) - \langle Y^k, X\rangle$ of $F^t(X)$. This procedure prevents DCASOM converging prematurely to a bad solution while starting from a bad initial map.

DCASOM is very simple and inexpensive, it requires only elementary operations on vectors. The complexity of one iteration of DCASOM is determined as follows.

The computation of the neighborhood matrix ϕ^t need $O(m^2)$ operations. For the step 1, the computations of $X^s - A^{[i]}$ and $\|X_k^s - a_i\|^2$ for $k = 1,\ldots,m$, $i = 1,\ldots,n$ need $O(nmd)$ operations and the determination $l_i^* = \arg\min_{l=1,\ldots,m}\sum_{k=1}^m \phi^t_{l,k}\|X_k^s - a_i\|^2$, $i = 1,\ldots,n$ (with $\|X_k^s - a_i\|^2$ being given) requires $O(nm^2)$ operations. The calculation of X^{s+1} in the step 2 needs $O(nmd)$ operations. Thus, the complexity of DCASOM is $O(m^2) + iter_t^{DCA}\cdot(O(nmd) + O(nm^2))$, where iter_t^{DCA} denotes the number of iterations of DCASOM (for a given t).

3 A training version for DCASOM: DCASOM-t

The SOM is usually trained iteratively with a monotonous decrease of σ. A cooling schedule is used to indicate how σ decreases in time. At the t^{th} learning step, a new temperature value of σ is often computed according to the cooling schedule

$$\sigma = \sigma_{max}\left(\frac{\sigma_{min}}{\sigma_{max}}\right)^{\frac{t-1}{N_{iter}-1}}. \tag{39}$$

Here, N_{iter} is the number of learning steps and $[\sigma_{min}, \sigma_{max}]$ is the temperature variation interval. In existing SOM algorithms N_{iter} should be a very large number ($N_{iter} \geq 5000$),i.e. σ is decreasing slowly to avoid local minima and get a good map.

In this section, we propose a training version for DCASOM based on an efficient cooling schedule of σ. More precisely, DCASOM is applied N_{iter}^{DCA} times with σ being computed by

$$\sigma = \begin{cases} \sigma_{max} & \text{if } t = 1, \\ \sigma_{min} & \text{if } t = N_{iter}^{DCA}, \\ \sigma_{max}\left(\frac{\sigma_{min}}{\sigma_{max}}\right)^{\frac{t-1}{N_{iter}^{DCA}-1}} - \xi & \text{otherwise} \end{cases}, \tag{40}$$

where $\xi > 0$ is a sufficiently small number. The algorithm, called DCASOM-t, can be described as follows:

DCASOM-t
Let X^0 be an initial point in $\mathbb{R}^{m\times d}$. Set $t = 1$. Choose a positive value for $\sigma_{max}, \sigma_{min}$, ξ, N_{iter}^{DCA}, and ε_t.
repeat: (learning step t)
a. calculate σ via (40) and set $s = 0$. Compute the neighborhood matrix ϕ^t via (1).
b. **repeat:**
1. Determine

$$l_i^* = \arg\min_{l=1,\ldots,m} \sum_{k=1}^{m} \phi_{l,k}^t \|X_k^s - a_i\|^2, \text{ for each } i = 1,\ldots,m,$$

2. Calculate X_k^{s+1}, for $k = 1,\ldots,m$, as

$$X_k^{s+1} = X_k^s - \frac{\sum_{i=1}^{n} (\phi_{l_i^*,k}^t)(X_k^s - a_i)}{n(\phi_+^t)_k}.$$

3. $s = s + 1$.
until $\| X^{s+1} - X^s \| \leq \varepsilon_t \| X^s \|$.

c. $t = t + 1$.
until $T > N_{iter}^{DCA}$.

On contrary to several learning SOM algorithms, in DCASOM-t we set a very small value to N_{iter}^{DCA} ($N_{iter}^{DCA} = 3$ in our experiments). The STOP condition at each iteration t (say, the value of ε_t) may be different. The complexity of DCASOM-t is

$$N_{iter}^{DCA} \cdot (O(m^2) + iter_t^{DCA} \cdot (O(nmd) + O(nm^2))).$$

4 Numerical Experiments

We compare our algorithm with BSOM-t, a standard Batch SOM algorithm. BSOM-t performs, at each learning step, one iteration of a K-means like algorithm for the BSOM problem (5). The algorithm is described as follows.

BSOM-t
Initialization: Let X^0 be an initial point in $\mathbb{R}^{m\times d}$. Set $t = 0$ and choose a positive values for $\sigma_{max}, \sigma_{min}$, and N_{iter}.
repeat: (learning step t)
a. Calculate σ via (40) in which N_{iter}^{DCA} is replaced by N_{iter}, and calculate the neighborhood matrix ϕ^t via (1).
b. Determine

$$l_i^* = \arg\min_{l=1\ldots m} \sum_{k=1}^{m} \phi_{l,k}^t \|X_k^t - a_i\|^2, \forall i = 1,\ldots,n.$$

c. Recalculate the weight vectors X_k^{t+1}, for $k = 1,...,m$ as

$$X_k^{t+1} \in \arg\min_{X\in\mathbb{R}^{m\times d}} \frac{1}{m} \sum_{i=1}^{n} \sum_{k=1}^{m} \phi_{l_i^*,k}^t \|X_k - a_i\|^2, \text{ i.e. } X_k^{t+1} = \frac{\sum_{i=1}^{n} \phi_{l_i^*,k}^t a_i}{\sum_{i=1}^{n} \phi_{l_i^*,k}^t}.$$

d. t=t+1.
until $t > N_{iter}$.

Similar to DCASOM, in BSOM-t the computation of the neighborhood terms needs $O(m^2)$ operations, the assignment step runs in time $O(nmd) + O(nm^2)$ and the recalculation of the weight vectors runs in time $O(nd) + O(m^2d)$. Therefore, the complexity of BSOM is $O(m^2) + O(nmd) + O(nm^2) + O(m^2d)$, and that of BSOM-t is

$$N_{iter} \cdot (O(m^2) + O(nmd) + O(nm^2) + O(m^2d).$$

Since N_{iter} should be much larger than $iter_t^{DCA} N_{iter}^{DCA}$, BSOM-t is much more expensive than DCASOM-t, especially when m (the map size) is large.

All the experiments have been conducted on a Intel(R) Core(TM) i7 CPU 2 $\times$ 2.20 Ghz with 4 Gb of memory.

4.1 The datasets

Numerical experiments were performed on several real world datasets. The small datasets named Spaeth Cluster Analysis (Spaeth in short) can be found at the address http://people.sc.fsu.edu/∼jburkardt/datasets/spaeth. The datasets A1_Set, A2_Set and A3_Set can be found at http://cs.joensuu.fi/sipu/datasets while Glass, Inosphere, Letter Recognition (Letter) and Color Moments of Corel Image Features (ColorM) are taken from UCI Machine Learning Repository. Sensor dataset is extracted from the Sensor Stream Data Mining Repository in [64]. The description in details of the datasets is presented in Table 1.

Table 1 The description of datasets

dataset	note	# element	# dimension
Spaeth05	S05	55	2
Spaeth06	S06	50	2
Spaeth07	S07	52	2
Spaeth08	S08	80	2
Spaeth10	S10	49	56
Spaeth13	S13	96	5
Spaeth09	S09	112	18
Glass	GLA	214	10
Inosphere	INO	351	34
A1_Set	A1S	3,000	2
A2_Set	A2S	5,250	2
A3_Set	A3S	7,500	2
Letter	LET	20,000	16
ColorM	COL	68,040	9
Sensor	SEN	1,169,260	5

4.2 Experiment setting

We are interested in the quality of optimal solutions (to the clustering task), the quality of maps (to the visualization task) as well as the rapidity of algorithms (CPU time). For this purpose we evaluate BSOM-t and DCASOM-t via four criteria: the objective value that

reflects the quality of clustering (this is the SOM Distortion Measure), the Topographic Error [32], the Kaski-Lagus measure [30], and the CPU times (in seconds).

The Topographic Error is usually used to measure the topology preservation. For each data sample, we determine the best and second-best matching neurons. An error occurs if these neurons are not adjacent on the map lattice. The total error on all data samples is then normalized to a range form 0 to 1, where 0 corresponds to a perfect topology preservation.

The Kaski-Lagus measure combines quantization and topographic errors to evaluate the goodness of a SOM and to compare different SOMs. This measure can be used to choose maps that do not fold unnecessarily in the input space while representing the input data distribution [30]. For each data item a_i, let x_{bi} be the best matching reference vector and x_{si} be the second nearest reference vector of a_i. We compute the distance from a_i to x_{si} passing first from a_i to x_{bi}, and thereafter along the shortest path to x_{si} through a series of reference vectors. Each reference vector in the series must belong to a neuron that is an immediate neighbor of the previous neuron. Hence, it measures both the accuracy in representing the data and the continuity of the data-to-grid mapping.

The temperature parameter σ decreases from $\sigma_{max} = 10$ to $\sigma_{min} = 0.2$ for both DCASOM-t and BSOM-t. As indicate above, we set $N_{iter}^{DCA} := 3$ (i.e. DCASOM is performed 3 times). Whereas, BSOM-t is performed 5000 learning steps to get a good result (the map is very bad when $N_{iter} < 5000$).

In DCASOM-t we set $\varepsilon_t = 10^{-2}$, 10^{-6} and 10^{-8} for $t = 1,2,3$ respectively, and $\xi = 0.2$.

We test the algorithms on different map dimensions.

Our experiment is composed of three parts. In the first part we compare the two algorithms on all datasets in case of small map dimensions. In the second experiment we show how these algorithms scale with the map dimensions, and in the last experiment we study the sensitivity of DCASOM to the initial map X^0.

In the two first experiments the initial map X^0 is chosen as follows. Let $\overline{a} = (1/n)\sum_{i=1}^{n} a_i$ be the gravity center of $\{a_1,\ldots,a_n\}$. Then we set (e stands for the vector of one and $[m]$ denotes the integer part of m))

$$X_i^0 = \bar{a} - \left(\frac{[m]}{2} - i\right)\cdot\frac{1}{2}e, \text{ for } 1 \le i \le \frac{[m]}{2}, \quad X_i^0 = \bar{a} + \left(i - \frac{[m]}{2}\right)\cdot\frac{1}{2}e, \text{ for } \frac{[m]}{2} < i \le m, \tag{41}$$

i.e., the neighborhood distances between the map nodes are equal to 0.5.

4.3 Experiment 1: comparison between DCASOM-t and BSOM-t

First, as an example to show the quality of the map, on the Figure 1 we present the maps provided by DCASOM-t and BSOM-t on the dataset Spaeth_05 with map's size (5×7). We observe that, with only 3 iterations, DCASOM-t gives better results than BSOM-t on both objective value and topographic map.

In Table 2, we present the comparative results of DCASOM-t and BSOM-t in terms of objective values, Kaski Measure and Topographic Error with the map dimension (3×5) and (5×5). The CPU times of the methods are reported in the Table 4. In the tables, bold symbols correspond to the best results.

Some of these datasets present clusters where real clusters are known. Hence we can compute the Percentage of the Well Placed Objects, denoted PWPO, to evaluate the performance of the algorithms in terms of clustering task. After applying the SOM training algorithm, each output unit of SOM training is associated with a group of data samples.

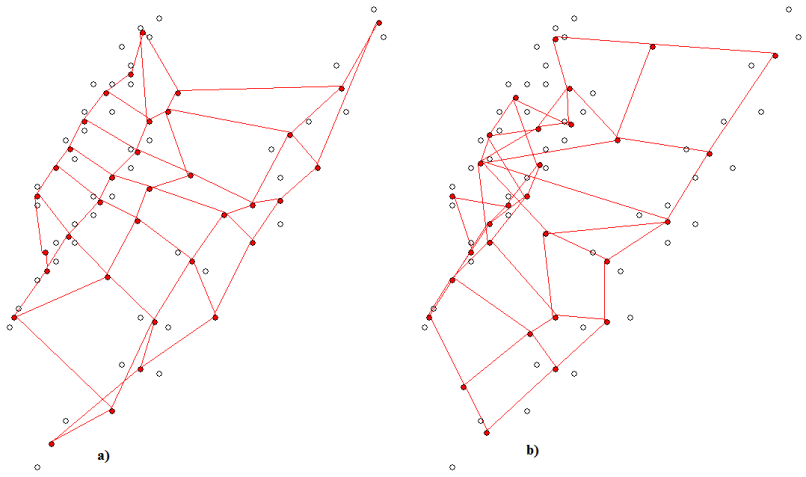


Fig. 1 Topographic map (5 x 7) of Spaeth_05; σ_{max} = 10; σ_{min} = 0.2; a) DCASOM-t: $N_{iter} = 3$, Obj. = 135.454, Kaski = 7.672; b) BSOM-t: $N_{iter} = 1000$, Obj. = 310.829, Kaski = 10.763.

These m output units are then grouped onto K clusters (K is equal to the real number of clusters in the dataset) via a K-ways clustering algorithm (here we use GKMSSC ([44])), and finally data samples are partitioned accordingly.

In Table 3 we report PWPO given by each algorithm for three cluster datasets.

For a more visible comparative results, we use the curves in Figures 2 - 5 to describe the results given in Tables 2.

Table 2 Objective value, Kaski Measure and Topographic Error of DCASOM-t (1) and BSOM-t (2) on each dataset.

Data	Map size (3 x 5)						Map size (5 x 5)					
	Obj. value		Kaski		Topo. Err.(%)		Obj. value		Kaski		Topo. Err.(%)	
	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
S05	**598.34**	655.77	**20.48**	20.58	14.55	**5.45**	**218.65**	226.13	**10.52**	10.70	16.36	**14.55**
S06	**375.54**	422.53	**16.60**	19.12	**6.00**	18.00	**156.79**	195.31	**10.23**	13.02	**22.00**	26.00
S07	**606.87**	633.73	**22.84**	24.23	**25.00**	38.46	**278.00**	287.44	**13.95**	17.55	**30.77**	38.46
S08	**930.16**	1,085.91	**24.78**	28.96	**23.75**	36.25	592.21	**581.77**	**15.95**	16.05	**17.50**	26.25
S10	**134.63**	158.77	**4.86**	6.18	**30.61**	63.27	**94.50**	129.49	**4.47**	5.09	**10.82**	57.14
S13	**1,573.59**	1,578.95	**29.86**	31.31	**40.63**	44.79	**285.19**	288.81	**12.28**	21.44	**36.46**	51.04
S09	**1,025.15**	1,073.89	**12.60**	14.01	**24.59**	28.69	**786.81**	977.95	**11.76**	14.05	**36.07**	45.08
GLA	**197.49**	217.52	**2.56**	3.36	**25.70**	29.91	**128.63**	153.50	**2.35**	2.79	**37.85**	44.39
INO	**1,445.02**	1,452.40	**6.34**	6.41	21.65	**19.37**	**1,240.07**	1,351.42	6.51	**6.32**	**33.90**	35.33
A1S	**287.39**	353.27	1.38	**1.36**	**24.93**	29.93	**135.80**	135.91	**1.03**	1.24	**23.63**	29.17
A2S	1,092.17	**1,089.11**	**1.47**	1.52	**0.65**	3.39	537.11	**514.55**	**1.05**	1.40	**11.16**	28.88
A3S	**2,544.79**	2,592.01	**1.80**	2.38	**6.16**	25.36	1,331.41	**1303.49**	**1.42**	1.52	**8.93**	16.89
LET	**748,054.0**	768,276.0	**49.37**	56.57	**46.12**	70.35	635,048.0	**627,985.0**	**45.74**	48.67	**55.63**	61.45
COL	255,782.0	**254,559.0**	8.07	**7.90**	51.28	**41.21**	212,702.0	**212,402.0**	**7.62**	8.03	**56.41**	58.96
SEN	**1,2066.9**	12,353.2	0.37	**0.34**	**13.74**	14.1	**8,793.0**	8,946.83	**0.33**	**0.33**	**39.18**	41.48

Table 3 PWPO of DCASOM-t (1) and BSOM-t (2).

Data	Map size (3 x 5)		Map size (5 x 5)	
	(1)	(2)	(1)	(2)
GLA	**86.32**	85.76	**88.89**	84.62
INO	**86.92**	85.98	**89.25**	88.79
SEN	85.95	**87.28**	**85.32**	83.56

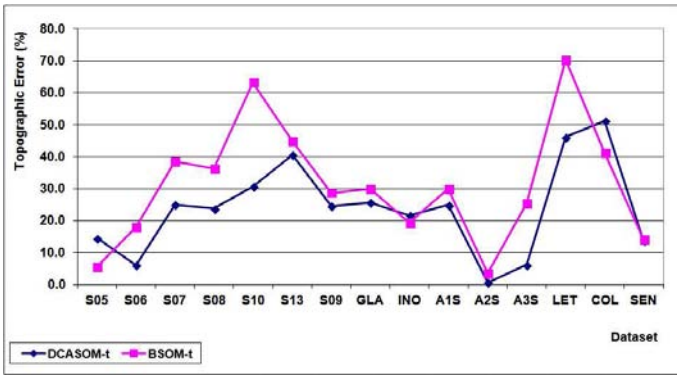


Fig. 2 Topographic Error of DCASOM-t and BSOM-t on all datasets, map size (3×5)

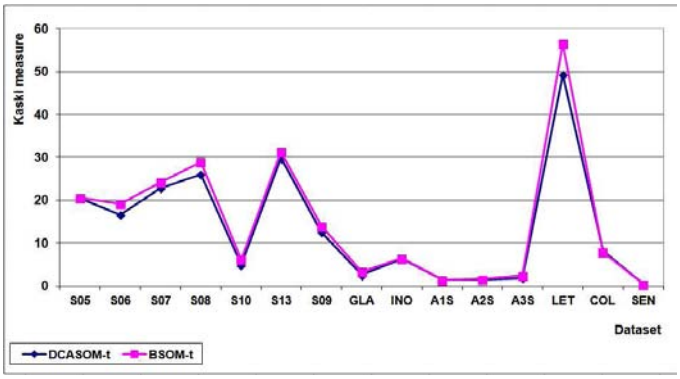


Fig. 3 Kaski Measure of DCASOM-t and BSOM-t on all datasets, map size (3×5)

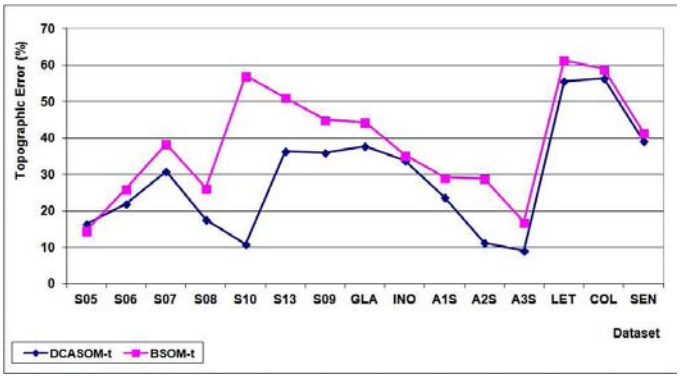


Fig. 4 Topographic Error of DCASOM-t and BSOM-t on all datasets, map size (5×5)

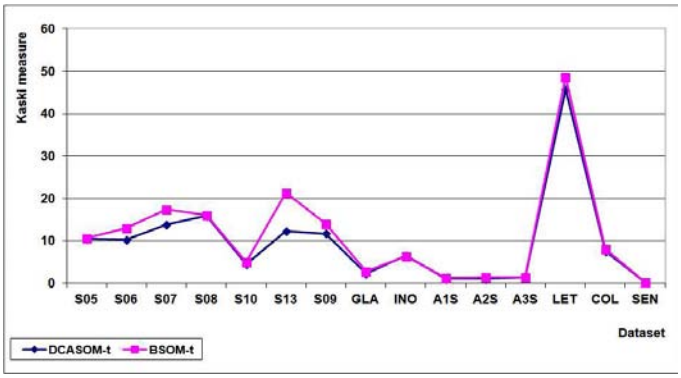


Fig. 5 Kaski Measure of DCASOM-t and BSOM-t on all datasets, map size (5×5)

Table 4 The CPU times (s) of DCASOM-t and BSOM-t on all datasets

Data	Map size (3×5)		Map size (5×5)	
name	DCASOM-t	BSOM-t	DCASOM-t	BSOM-t
S05	**0.06**	0.42	**0.09**	0.91
S06	**0.03**	0.39	**0.11**	0.85
S07	**0.06**	0.40	**0.08**	0.87
S08	**0.11**	0.59	**0.16**	1.36
S10	**0.25**	1.56	**0.42**	3.00
S13	**0.08**	0.81	**0.11**	1.71
S09	**0.23**	1.69	**0.45**	3.24
GLA	**0.28**	2.06	**0.55**	4.43
INO	**1.12**	6.67	**4.21**	13.31
A1S	**1.93**	21.09	**3.03**	43.05
A2S	**3.43**	35.62	**2.71**	80.68
A3S	**5.49**	51.16	**12.46**	114.98
LET	**35.37**	243.85	**68.43**	497.28
COL	**81.81**	646.40	**168.95**	1,359.20
SEN	**1,108.05**	6,650.70	**1,527.27**	12,308.50

We observe from the computational results that

- In terms of **Objective value and Kaski measure**: DCASOM-t gives better results on both objective value and Kaski Measure than BSOM-t in almost datasets. The objective values (resp. Kaski Measure) given by DCASOM-t are better than that of BSOM-t in $13/15$ datasets with map size 3×5 and $10/15$ datasets with map size 5×5 (resp. $12/15$ datasets with map size 3×5 and $13/15$ datasets with map size 5×5).

- In terms of **Topographic map**: the maps given by DCASOM-t have a hight topographic quality. The Topographic Errors of DCASOM-t are smaller than that of BSOM-t in almost cases: $12/15$ cases with map size 3×5 and $14/15$ cases with map size 5×5). Note that the topographic map is very important for the data visualization.

- In terms of **CPU times**: DCASOM-t is faster than BSOM-t (the ratio varies from 6 to 29.8 times). In addition, DCASOM-t is less sensitive to the cooling schedule of σ. With only 3 iterations, DCASOM-t ensures a high quality in terms of both objective value and topographic map.

- About **Clustering task**: the PWPO values of DCASOM-t are quite good (more than 85%). DCASOM-t is slightly better than BSOM-t on $5/6$ test cases (except the SEN dataset on the map 3×5).

4.4 Experiment 2: behaviors of algorithms when the map dimension varies

To study how the algorithms scale with the map dimensions we test DCASOM-t and BSOM-t on the same dataset when map dimensions increase. The medium dataset A3S is chosen for this purpose (BSOM-t does not work on the datasets with larger size). To see where are the limits of map dimensions in each algorithm we set the limit CPU time equal to 15000 seconds for each run.

The Figure 6, 7 and 8 show, respectively, the CPU time, the Kaski measure and the Topographic Error of the comparative algorithms when the map dimension varies. from (5×5) to (45×45).

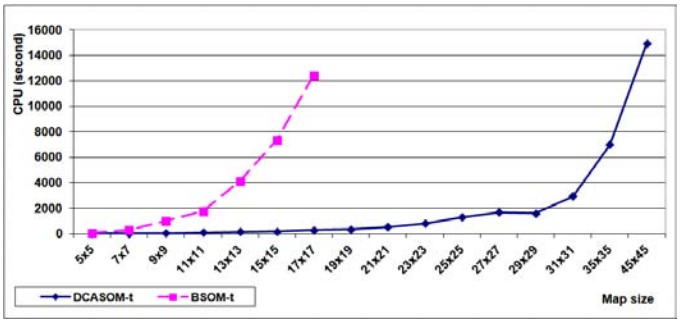


Fig. 6 The CPU time (in seconds) of two algorithms when map size increases (A3S dataset).

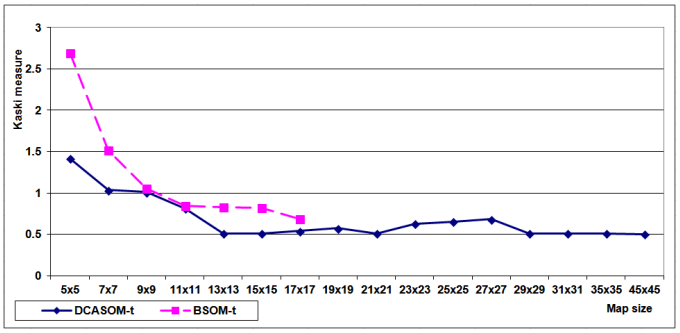


Fig. 7 The Kaski measure of two algorithms when map size increases (A3S dataset).

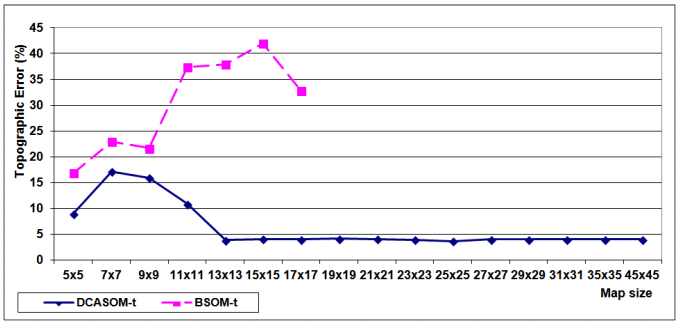


Fig. 8 The Topographic Error of two algorithms when map size increases (A3S dataset).

The numerical results show that, for the dataset A3S, the maximal map dimension on which BSOM-t (resp. DCASOM-t) works (say, CPU time until the convergence is less than 15000 seconds) is 17×17 (resp. 45×45). Moreover, we observe that the CPU time of BSOM-t increases dramatically when the map dimension increases from 5×5 to 17×17,

whereas the CPU time of DCASOM-t slightly increases. The ratio of gain of DCASOM-t versus BSOM-t varies from 6.22 to 43.20 (comparing among the cases both algorithms work). As for the Kaski measure and the Topographic Error, DCASOM-t gives always better results than BSOM-t, and DCASOM-t is more or less insensitive to the map dimension in the interval 13×13 and 45×45.

4.5 Experiment 3: the sensitivity DCASOM-t to initial points

To evaluate the sensitivity DCASOM-t to the initial map, we test the algorithm on 10 random initial solutions and on the particular solution used in the two first experiments. For randomly generating the map X^0 we first randomly choose a center a_c in the set $\{a_1, \ldots, a_n\}$ and then use a similar procedure to (41) in which $\bar{a}$ is replaced by a_c (the neighborhood distances between the map nodes are equal to 0.5). The numerical results are reported in the Table 5. For the random initial points, we present the mean and standard deviations of each criterion on 10 runs (objective values, Kaski measure, Topographic Error and CPU time. The map size is set to 7×7.

Table 5 DCASOM-t with different starting points: the center initial points (1) and the random initial points (2)

	Objective Value		Kaski Measure		Topo. Error(%)		CPU (seconds)	
	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
S05	**218.67**	249.72±60.26	**10.38**	13.58±2.28	**16.36**	29.64±4.38	0.31	0.31±0.02
S06	181.38	**155.49**±28.41	10.53	**10.31**±0.56	**24.00**	28.20±5.40	0.43	0.39±0.03
S07	**231.12**	278.83±38.88	**12.17**	12.73±0.84	28.85	**25.77**±5.38	0.30	0.31±0.01
S08	427.98	**370.14**±20.12	**12.56**	13.30±1.05	**26.25**	39.63±4.78	0.46	0.55±0.02
S10	**186.61**	197.20±7.06	**6.48**	6.83±0.39	**2.04**	12.65±15.78	0.84	0.63±0.16
S13	**285.15**	286.65±2.07	**10.64**	12.85±1.94	**33.12**	34.09±3.46	0.30	0.31±0.01
S09	**779.28**	860.59±43.20	**12.14**	13.08±0.57	**38.53**	44.26±3.46	1.33	1.30±0.04
GLA	**238.21**	251.22±35.76	**3.03**	3.14±0.23	**34.58**	43.22±9.92	1.01	0.79±0.21
INO	**1,586.88**	1,718.37±83.02	**11.09**	12.33±0.99	**39.60**	48.06±7.68	2.88	2.80±0.15
A1S	**93.83**	95.95±9.15	**0.73**	0.75±0.02	**25.53**	26.90±1.15	19.52	21.29±2.06
A2S	**328.54**	359.85±55.57	**1.09**	1.18±0.18	**34.97**	39.47±5.33	38.77	41.99±5.80
A3S	**313.78**	740.04±30.39	**1.04**	1.16±0.02	**17.13**	25.68±0.57	30.67	53.12±7.33
LET	**630,351.0**	635,966.4±2,2193.3	**45.02**	45.03±0.64	**73.68**	74.48±2.91	144.49	162.9±28.7
COL	**180,905.0**	190,606.4±5,523.3	**9.03**	10.22±0.54	**71.11**	73.07±2.68	680.14	554.9±54.9
SEN	8,808.6	**8,779.12**±24.09	**0.31**	0.35±0.04	**42.17**	47.21±4.14	3550.88	3582.1±33.6

We observe from this result that the performance of DCASOM-t with the particular procedure (1) for generating initial map is slightly better than that of the random initial maps. The objective value (resp. Kaski measure and Topographic Error) is better in $12/15$ (resp. $14/15$ and $14/15$) datasets. The gain ratio varies from 1.01 to 2.36 (resp. from 1.02 to 1.31 and from 1.01 to 6.20) times. Within the results given by 10 random initial maps, the Kaski measure and Topographic Error are relatively stable. The standard deviation values of Kaski measure (resp. Topographic Error) are less than 16.79% (resp. 23%, except on S10 dataset) of the mean results. This shows that DCASOM-t is not very sensitive to initial maps.

5 Conclusion

We have proposed a DCA based approach for the Batch Learning Self-Organizing Maps model. The nonsmooth, nonconvex optimization formulation of BSOM is reformulated as a DC program for which an efficient DCA scheme is developed. By using an elegant matrix formulation and a natural choice of DC decomposition, we get a DC program that consists in minimizing the difference of a strong convex quadratic function and a nonsmooth convex term. This nice feature is very relevant for applying DCA, it makes simpler and so much less expensive the computations in the resulting DCA. The DCASOM requires only sums and scalar multiplications of vectors. Further, it uses a simple but effective strategy to create a starting point. A training version for DCASOM with an efficient cooling schedule is investigated that consists in applying only a very few (3 in our experiments) times of DCASOM.

DCASOM is a simple and elegant scheme, and as the standard BSOM algorithm, it can be implemented on any platform. In addition, DCASOM has several advantages: it is more or less insensitive to initial solutions, the quality of solutions is good in terms of both classification and visualisation (topographic map), it requires a very few training steps and then can handle problems with large dimensions. The numerical results show that the proposed approach gives better results than the standard Batch SOM method. DCASOM-t is fast and scalable and is promising for large scale setting. In particular, thank to its efficiency and its scalability, DCASOM could be an effective approach for clustering massive dataset. On another hand, the efficiency of DCASOM suggests us to develop in future works DCA for kernel versions of BSOM and for minimizing the energy function $E^t(X)$ in which the neighborhood function $\phi_{l,k}^t$ are modified within the step t.

Acknowledgements. We are very grateful to the anonymous referees and the associate editor for their really helpful and constructive comments that helped us to improve our paper.

References

1. Amerijckx C. , Legaty J.D. , Verleysen M.: Image Compression Using Self-Organizing Maps. Systems Analysis Modelling Simulation, Vol. 43, No. 11, pp. 1529–1543 (2003).
2. Argyrou A.: Clustering Hierarchical Data Using Self-Organizing Map: A Graph-Theoretical Approach. Advances in Self-Organizing Maps, Lecture Notes in Computer Science, Volume 5629, pp. 19–27 (2009).
3. Astudillo C.A., Oommen B.J.: Topology-oriented self-organizing maps: a survey. Pattern Analysis and Applications, pp. 1–26 (2014).
4. Barreto G.A., Araúo A.F.R., Ritter H.J.: Self-Organizing Feature Maps for Modeling and Control of Robotic Manipulators. Journal of Intelligent and Robotic Systems, Volume 36, Issue 4, pp. 407–450 (2003).
5. Bradley B.S., Mangasarian O.L.: Feature selection via concave minimization and support vector machines. Machine Learning Proceedings of the Fifteenth International Conferences (ICML'98), San Francisco, California, MorganKaufmann, pp. 82–90 (1998).
6. Burger M., Graepel T., Obermayer K.: Phase transitions in stochastic self-organizing maps. Physical Review E 56, pp. 3876–3890 (1997).
7. ChandraShekar B.H., Shoba Dr.G.: Classification Of Documents Using Kohonens Self-Organizing Map. International Journal of Computer Theory and Engineering, Vol. 1, No. 5, pp. 610–613 (2009).
8. Chang L., Jun-min L., Chong-xiu Y.: Skin detection using a modified Self-Organizing Mixture Network. Automatic Face and Gesture Recognition (FG), 2013 10th IEEE International Conference and Workshops, vol. 1, no. 6, pp. 22–26 (2013).
9. Collobert R., Sinz F., Weston J., Bottou L.: Trading Convexity for Scalability. International Conference on Machine Learning ICML (2006).
10. Cottrell M., Hammer B., Hasenfß E., Villmann H.: Batch neural gas. International Workshop on Self-Organizing Maps (WSOM), pp. 275–282 (2005).

11. De A.T. de Carvalho F., Bertrand P., De Melo F.M.: Batch self-organizing maps based on city-block distances for interval variables. Hal-00706519, version 1 (2012).
12. Dempster A.P., Laird N.M., Rubin D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Stat. Soc. B. (1977).
13. Dickerson K.B., Ventura D.: Music Recommendation and Query-by-Content Using Self-Organizing Maps. Proceedings of the International Joint Conference on Neural Networks, pp. 705–710, (2009).
14. Doan N.Q., Azzag H., Lebbah M.: Self-Organizing Map and Tree Topology for Graph Summarization, Artificial Neural Networks and Machine Learning ICANN 2012, Lecture Notes in Computer Science, Volume 7553, pp. 363–370 (2012).
15. Dozono H., Tokushima H., Hara S., Noguchi Y.: An Algorithm of SOM using Simulated Annealing in the Batch Update Phase for Sequence Analysis. International Workshop on Self-Organizing Maps (WSOM), pp. 171–178 (2005).
16. Fiannaca A., Fatta G.D., Gaglio S., Rizzo R., Urso A.M.: Improved SOM Learning Using Simulated Annealing. Lecture Notes in Computer Science, Vol. 4668, pp. 279–288 (2007).
17. Fort J.C., Letremy P., Cottrell M.: Advantages and drawbacks of the Batch Kohonen algorithm. The European Symposium on Artificial Neural Networks conference - ESANN, pp. 223-230 (2002).
18. Fukui K., Numao M.: Topographic Measure Based on External Criteria for Self-Organizing Map. Advances in Self-Organizing Maps, Lecture Notes in Computer Science, Volume 6731, pp. 131–140, Springer-Verlag Berlin Heidelberg (2011).
19. Graef G., Schaefer C.: Application of ART2 Networks and Self-Organizing Maps to Collaborative Filtering. Hypermedia: Openness, Structural Awareness, and Adaptivity, Lecture Notes in Computer Science, Volume 2266, pp. 296–309 (2002).
20. Graepel T., Burger M., Obermayer K.: Self-Organizing Maps: Generalizations and New Optimization Techniques. Neurocomputing, Volume 21, Issues 13, pp. 173–190 (1998).
21. Günter S., Bunke H.: Self-organizing map for clustering in the graph domain. Pattern Recognition Letters, Volume 23, Issue 4, pp. 405–417 (2002).
22. Guo X., Wang H., Glass D.H.: Bayesian Self-Organizing map for Data Classification and Clustering. International Journal of Wavelets, Multiresolution and Information Processing 11(5), 12 pages (2013).
23. Hagenbuchner M., Sperduti A., Tsoi A.C.: Graph self-organizing maps for cyclic and unbounded graphs. Neurocomputing, Volume 72, Issues 79, pp. 1419–1430 (2009).
24. Hagenauer J., Helbich M.: Hierarchical self-organizing maps for clustering spatiotemporal data. International Journal of Geographical Information Science, Volume 27, Issue 10, pp. 2026–2042 (2013).
25. Heiss-Czedik D., Bajla I.: Using Self-Organizing Maps for object classification in Epo image analysis. MEASUREMENT SCIENCE REVIEW, Volume 5, Section 2, pp. 11–16 (2005).
26. Heskes T.: Energy functions for self organizingmaps. In Oya, S., Kaski, E., eds.:KohonenMaps. Elsevier, Amsterdam pp. 303–316 (1999).
27. Heskes T.: Self-Organization Maps, vector quantization, and mixture modeling. IEEE transactions on neural networks, Vol. 12, No. 6 (2001).
28. Hiriart Urruty J.B., Lemarechal C.: Convex Analysis and Minimization Algorithms. SpringerVerlag berlin Heidelberg (1993).
29. Ismail S., Shabri A., Samsudin R.: A hybrid model of self organizing maps and least square support vector machine for river flow forecasting. Hydrol. Earth Syst. Sci., 16, pp. 4417–4433 (2012).
30. Kaski S., Lagus K.: Comparing Self-Organizing Maps. Lecture Notes in Computer Science, Vol. 1112, pp. 809–814 (1996).
31. Khalilia M., Popescu M.: Topology Preservation in Fuzzy Self-Organizing Maps. In Advance Trends in Soft Computing, Studies in Fuzziness and Soft Computing, Volume 312, pp. 105–114 (2014).
32. Kiviluoto K.: Topology preservation in self-organizing maps. In: Proceedings of ICNN96, IEEE International Conference on Neural Networks, vol. 1, pp.294–299 (1996).
33. Kohonen T.: Analysis of a simple self-organizing process. Biol. Cybern. 44, pp. 135–140 (1982).
34. Kohonen T.: Self-Organization Maps. Springer Heidelberg (1997).
35. Krause N., Singer Y.: Leveraging the margin more carefully. International Conference on Machine Learning ICML (2004).
36. Le Thi H.A.: DC Programming and DCA. http://lita.sciences.univ-metz.fr/∼lethi.
37. Le Thi H.A and Pham Dinh T., Solving a class of linearly constrained indefinite quadratic problems by D.c. algorithms. Journal of Global Optimization, 11 (1997), pp. 253-285.
38. Le Thi H.A., Pham Dinh T.: DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems. Ann. Oper. Res.Springer-Verlag, 133, pp. 23–46 (2005).
39. Le Thi H.A., Le Hoai M., Pham Dinh T.: Fuzzy clustering based on nonconvex optimization approaches using difference of convex (DC) functions algorithms. Journal of Advances in Data Analysis and Classification 2:1-20 (2007).

40. Le Thi H.A., Le Hoai M., Nguyen V.V., Pham Dinh T.: A DC Programming approach for Feature Selection in Support Vector Machines learning. Journal of Advances in Data Analysis and Classification 2(3):259-278 (2008).
41. Le Thi H.A., Le Hoai M., Pham Dinh T., Huynh V.N.: Binary classification via spherical separator by DC programming and DCA. Journal of Global Optimization , pp. 1-15, doi:10.1007/s10898-012-9859-6 (2012).
42. Le Thi H.A., Le Hoai M., Pham Dinh T., Huynh V.N.: Block Clustering based on DC programming and DCA. NECO - Neural Computation, Volume 25, Num. 10, pp. 2776–2807 (2013).
43. Le Thi H.A., Pham Dinh T., Nguyen C.N., Le Hoai M.: DC Programming and DCA for Diversity Data Mining. to appear in Optimization.
44. Le Thi H.A., Le Hoai M., Pham Dinh T.: New and efficient DCA based algorithms for Minimum Sum-of-Squares Clustering. In press in Patter Recognition (2013).
45. Lee M., Choi P., Woo Y.: A Hybrid Recommender System Combining Collaborative Filtering with Neural Network. Adaptive Hypermedia and Adaptive Web-Based Systems, Lecture Notes in Computer Science 2347, pp. 531–534, Springer-Verlag Berlin Heidelberg (2002).
46. Lefebvre G., Zheng H., Laurent C.: Objectionable Image Detection by ASSOM Competition. Image and Video Retrieval, Lecture Notes in Computer Science, Volume 4071, pp. 201–210 (2006).
47. Liu Y., Shen X., Doss H.: Multicategory Ψ -Learning and Support Vector Machine, Computational Tools. Journal of Computational and Graphical Statistics 14:219-236 (2005).
48. Liu Y., Shen X.: Multicategory Ψ-Learning. Journal of the American Statistical Association, Vol. 101, No. 474, pp. 500–509 (2006).
49. Madalina O., Nathalie V.V., Christine C.A.: Multiple Kernel Self-Organizing Maps. ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), pp. 83–88 (2013).
50. Marc M. Van Hulle: Self-organizing Maps. Handbook of Natural Computing, pp. 585–622, Springer-Verlag Berlin Heidelberg (2012).
51. Marina R.: Graph Mining Based SOM: A Tool to Analyze Economic Stability. In Applications of Self-Organizing Maps, Magnus Johnsson edit, pp. 1–25, InTech Publisher (2012).
52. Matharage S., Alahakoon D., Rajapakse J., Huang P.: Fast Growing Self Organizing Map for Text Clustering. Neural Information Processing, Lecture Notes in Computer Science, Volume 7063, pp. 406–415 (2011).
53. Matsushita H., Nishio Y.: Batch-Learning Self-Organizing Map with Weighted Connections Avoiding False-Neighbor Effects. The 2010 International Joint Conference on Neural Networks (IJCNN), pp. 1–6 (2010).
54. Neme A., Miramontes P.: Self-Organizing Map Formation with a Selectively Refractory Neighborhood. Neural Processing Letters 39(1), pp. 1–24, 2014.
55. Ogihara M., Matsumoto H., Marumo T., Kuroda C.: Clustering of Pressure Fluctuation Data Using Self-Organizing Map. Engineering Applications of Neural Networks, Communications in Computer and Information Science, Volume 43, pp. 45–54 (2009).
56. Olteanu M., Villa-Vialaneix N., Cierco-Ayrolles C.: Multiple Kernel Self-Organizing Maps. Hal-00817920, version 1 (2013).
57. O'Connell C., Kutics A., Nakagawa A.: Layered Self-Organizing Map for Image Classification in Unrestricted Domains. In Image Analysis and Processing - ICIAP 2013, Lecture Notes in Computer Science, Volume 8156, pp. 310–319 (2013).
58. Paul S., Gupta M.: Image Segmentation By Self Organizing Map With Mahalanobis Distance. International Journal of Emerging Technology and Advanced Engineering, Volume 3, Issue 2, pp. 288–291 (2013).
59. Pham Dinh T. and Le Thi H.A., Convex analysis approach to d.c. programming: Theory, Algorithms and Applications (dedicated to Professor Hoang Tuy on the occasion of his 70th birthday), Acta Mathematica Vietnamica, 22(1997), pp. 289-355.
60. Pham Dinh T., Le Thi H.A.: DC optimization algorithms for solving the trust region sub-problem. SIAM J. Optim. 8, pp. 476–505 (1998).
61. Pölzlbauer G.: Survey and Comparison of Quality Measures for Self-Organizing Maps. In: WDA 2004. Fifth Workshop on Data Analysis, pp. 67–82. Elfa Academic Press (2004).
62. Pratiwi D.: The Use of Self Organizing Map Method and Feature Selection in Image Database Classification System. International Journal of Computer Science Issues, Vol. 9, Issue 3, No 2, pp. 377-381 (2012).
63. Roh T.H., Oh K.J., Han I.: The collaborative filtering recommendation based on SOM cluster-indexing CBR. Expert Systems with Applications, Volume 25, Issue 3, pp. 413–423, ISSN 0957-4174, http://dx.doi.org/10.1016/S0957-4174(03)00067-8 (2003).

64. Ruan X., Gao Y., Song H., Chen J.: A New Dynamic Self-Organizing Method for Mobile Robot Environment Mapping. Journal of Intelligent Learning Systems and Applications 3, pp. 249–256 (2011).
65. Saarikoski J., Laurikkala J., Järvelin K., Juhola M.: Self-Organising Maps in Document Classification: A Comparison with Six Machine Learning Methods. Adaptive and Natural Computing Algorithms, Lecture Notes in Computer Science, Volume 6593, pp. 260–269 (2011).
66. Sarlin P., Eklund T.: Fuzzy Clustering of the Self-Organizing Map: Some Applications on Financial Time Series. In Advances in Self-Organizing Maps, Laaksonen J. and Honkela T. editor, Lecture Notes in Computer Science, Volume 6731, pp. 40–50, Springer Berlin Heidelberg (2011).
67. Shen X., Tseng G.C., Zhang X., Wong W. H.: ψ -Learning. Journal of American Statistical Association 98:724-734 (2003).
68. Smith T., Alahakoon D.: Growing Self-Organizing Map for Online Continuous Clustering. Foundations of Computational Intelligence Volume 4, Studies in Computational Intelligence, Volume 204, pp. 49–83 (2009).
69. Stefanovic P., Kurasova O.: Influence of Learning Rates and Neighboring Functions on Self-Organizing Maps. Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 141–150 (2011).
70. Szymanski J., Duch W.: Self Organizing Maps for Visualization of Categories. In Neural Information Processing, Lecture Notes in Computer Science, Volume 7663, pp. 160–167 (2012).
71. Tsoi A.C., Hagenbuchner M., Sperduti A.: Self-organising Map Techniques for Graph Data Applications to Clustering of XML Documents. Advanced Data Mining and Applications, Lecture Notes in Computer Science, Volume 4093, pp. 19–30 (2006).
72. Van Laerhoven K.: Combining the Self-Organizing Map and K-Means Clustering for On-Line Classification of Sensor Data. Artificial Neural Networks ICANN 2001, Lecture Notes in Computer Science, Volume 2130, pp. 464–469 (2001).
73. Vembu S., Baumann S.: A Self-Organizing Map Based Knowledge Discovery for Music Recommendation Systems. Computer Music Modeling and Retrieval, Lecture Notes in Computer Science, Volume 3310, pp. 119–129, Springer-Verlag Berlin Heidelberg (2004).
74. Vesanto J., Alhoniemi E.: Clustering of the Self-Organizing Map. IEEE transactions on neural networks, VOL. 11, NO. 3, pp. 586–600 (2000).
75. Villa N., Boulet R.: Clustering a medieval social network by SOM using a kernel based distance measure. ESANN'2007 proceedings - European Symposium on Artificial Neural Networks Bruges (Belgium), d-side publi., ISBN 2-930307-07-2 (2007).
76. Wang J., Shen Z., Pan W.: On transductive support vector machines. Proceeding of the International Conference on Machine Learning ICML (2007).
77. Wehrens R.: Self-Organising Maps for Image Segmentation. Advances in Data Analysis, Data Handling and Business Intelligence Studies in Classification, Data Analysis, and Knowledge Organization, pp. 373–383 (2010).
78. Yin H.: The Self-Organizing Maps: Background, Theories, Extensions and Applications. Studies in Computational Intelligence (SCI) 115, pp. 715–762 (2008).
79. Yuille A.L., Rangarajan A.: The Convex Concave Procedure (CCCP). Advances in Neural Information Processing System 14, Cambrige MA: MIT Press (2002).

Chapter 4

Sparse Semi-Supervised Support Vector Machines by DC Programming and DCA

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

Sparse Semi-Supervised Support Vector Machines by DC Programming and DCA

Hoai Minh LE · Manh Cuong NGUYEN · Hoai An LE THI

Submitted.
Received: date / Accepted: date

Abstract This paper studies the problem of feature selection in the context of Semi-Supervised Support Vector Machine (S3VM). The zero norm, a natural concept dealing with sparsity, is used for feature selection purpose. Due to two nonconvex terms (the loss function of unlabeled data and the ℓ_0 term), we are faced on a NP hard optimization problem. Two continuous approaches based on DC (Difference of Convex functions) programming and DCA (DC Algorithms) are developed. The first is DC approximation approach that approximates the ℓ_0-norm by a DC function.

The second is an exact reformulation approach based on exact penalty techniques in DC programming. All the resulting optimization problems are DC programs for which DCA are investigated. Several usual sparse inducing functions are considered, and six versions of DCA are developed. Empirical numerical experiments on several Benchmark datasets show the efficiency of the proposed algorithms, in both feature selection and classification.

Keywords Semi-Supervised SVM, Feature Selection, Non-Convex Optimization, DC approximation, exact penalty, DC Programming, DCA

1 Introduction

In machine learning, supervised learning is a task of inferring a predictor function (classifier) from a labeled training dataset. Each example in training set consists of an input object and a label. The objective is to build a predictor function which can be used to identify the label of new examples with highest possible accuracy. Nevertheless, in most of real word applications, a large portion of training data are unlabeled, and supervised learning can not be used in these contexts. To deal with this difficulty, recently, in the machine learning community, there has been an attracting increasing attention in using semi-supervised learning methods. In contrast to supervised methods, semi-supervised learning methods take into account both labeled and unlabeled data to construct prediction models.

We are interested in semi-supervised classification, more precisely, in the so called Semi-Supervised Support Vector Machines (S3VM). Among the semi-supervised classification methods, the large margin approach S3VM, which extends the Support Vector Machine (SVM) to semi-supervised learning concept, is certainly the most popular ([6]-[8], [10], [14], [19], [32] [44], [45], [49]). An extensive overview of semi-supervised classification methods can be found in [51].

Hoai Minh LE · Manh Cuong NGUYEN · Hoai An LE THI
Laboratory of Theoretical and Applied Computer Science - LITA EA 3097,
University of Lorraine, Ile du Saulcy, 57045 Metz, France.
minh.le@univ-lorraine.fr
manh-cuong.nguyen@univ-lorraine.fr
hoai-an.le-thi@univ-lorraine.fr

S3VM was originally proposed by Vapnik and Sterin in 1977 ([48]) under the name of transductive support vector machine. Later, in 1999, Bennett and Demiriz ([3]) proposed the first optimization formulation of S3VM which is described as follows.

Given a training set which consists of m labeled points $\{(x_i, y_i) \in \mathbb{R}^n \times \{-1, 1\}, i = 1, \ldots, m\}$ and p unlabeled points $\{x_i \in \mathbb{R}^n, i = (m+1), \ldots, (m+p)\}$. We are to find a separating hyperplane $P = \{x \mid x \in \mathbb{R}^n, x^T w = b\}$, far away from both the labeled and unlabeled points. Hence, the optimization problem of S3VM takes the form

$$\min_{w,b} \|w\|_2^2 + \alpha \sum_{i=1}^{m} L\left(y_i(\langle w, x_i\rangle + b)\right) + \beta \sum_{i=m+1}^{m+p} L\left(|\langle w, x_i\rangle + b|\right). \tag{1}$$

Here, the first two terms define a standard SVM while the third one incorporates the loss function of unlabeled data points. The loss function of labeled and unlabeled data points are weighted by penalty parameters $\alpha > 0$ and $\beta > 0$. Usually, in classical SVM one uses the hinge loss function $L(u) = \max\{0, 1 - u\}$ which is convex. On contrary, the problem (1) is nonconvex, due to the nonconvexity of the third term.

There are two broad strategies for solving the optimization problem (1) of S3VM: the combinatorial methods (Mixed Integer Programming ([3]), Branch and Bound algorithm ([7])) and the continuous optimisation methods such as self-labeling heuristic S^3VM^{light} ([15]), gradient descent ([6]), deterministic annealing ([44]), semi-definite programming ([5]), DC programming ([10]). Combinatorial methods are not available for massive data sets in real applications (*high* dimension and *large* data set). Thus, major efforts have focused on efficient local algorithms. For more complete reviews of S3VM methods, the reader is referred to [8, 51] and references therein.

On another hand, feature selection is one of the fundamental problems in machine learning. In many areas of application such as text classification, web mining, gene expression, microarray analysis, combinatorial chemistry, image analysis, etc, data sets contain a large number of features, many of which are irrelevant or redundant. Feature selection is often applied to high dimensional data prior to classification learning. The main goal is to select a subset of features of a given data set while preserving or improving the discriminative ability of a classifier. Several feature-selection methods for SVMs have been proposed in the literature (see e.g. [4, 12, 16, 17, 25, 26, 29, 33, 38, 41, 52, 53]).

This paper deals with the feature selection in the context of S3VM. We are to find a separating hyperplane far away from both the labeled and unlabeled points, that uses the least number of features. As standard approaches for feature selection in SVM, here we use the zero norm, a natural concept dealing with sparsity for feature selection purpose. The zero norm of a vector, denoted ℓ_0 or $\|.\|_0$, is defined as the number of its nonzero components. Similar to SVM, we replace the term $\|w\|_2^2$ in (1) by the ℓ_0-norm and then formumate the feature selection S3VM problem as follows:

$$\min_{w,b} \|w\|_0 + \alpha \sum_{i=1}^{m} L\left(y_i(\langle w, x_i\rangle + b)\right) + \beta \sum_{i=m+1}^{m+p} L\left(|\langle w, x_i\rangle + b|\right). \tag{2}$$

While S3VM has been widely studied, there exist few methods in the literature for feature selection in S3VM. Due to the discontinuity of the ℓ_0 term and the non convexity of the third term, we are facing "double" difficulties in (2) (it is well known that the problem of minimizing the zero-norm is NP-Hard ([1])).

During the last two decades, research is very active in models and methods optimization involving the zero norm. Works can be divided into three categories according to the way to treat the zero norm: convex approximation (the ℓ_0-norm is replaced by a convex function, for instance the ℓ_1-norm [46] or the conjugate function [36]), nonconvex approximation (a continuous nonconvex function is used instead to the ℓ_0-norm, usual sparse inducing functions are introduced in [4, 53, 12, 37, 28]), and nonconvex exact reformulation (with the binary variables $u_i = 0$ if $w_i = 0$ and $u_i = 1$ otherwise, the original problem is formulated as a combinatorial optimization problem which is equivalently reformulated as a continuous nonconvex program via

exact penalty techniques, see [29,47]). An extensive overview of these approaches can be found in [29]. When the objective function (besides the ℓ_0-term) is convex, convex approximation techniques result to a convex optimization problem which is so far "easy" to solve. Unfortunatly, for S3VM, the problem (2) remains nonconvex with any approximation - convex or nonconvex, of the ℓ_0-norm.

Since convex approximation approaches have been shown to be, in certain cases, inconsistent for variable selection and biased [50], we tackle the problem (2) by nonconvex approaches. The core of our work is DC (Difference of Convex functions) programming and DCA (DC Algorithms), an efficient approach in nonconvex programming framework. This approach has been succesfully developed in a variety of works in Machine Learning (see e.g. [10,18,23,24,31, 40,42] and the list of reference in [30]), in particular to feature selection in SVM ([25,26,29,33, 38]). These successes of DCA suggest us to investigate it for solving the hard problem (2).

Paper's contributions. We develop an unified approach based on DC programming and DCA for solving the non convex optimization problem (2). Firstly, several DC approximations of ℓ_0-norm will be used to (2): logarithm function [53], Smoothly Clipped Absolute Deviation (SCAD) [12], piecewise exponential [4], DC polyhedral [37] and piecewise linear approximation [28]. Secondly, we inspire the same technique in [29,47] to equivalently formulate (2) as a combinatorial optimization problem. Then, thanks to the new result on exact penalty techniques recently developed in [27], we reformulate the resulting problem as a continuous optimization problem and investigate DCA to solve it. Finally, we provide an empirical experimentation, on several Benchmark datasets, of all proposed algorithms to study their efficiency in both feature selection and classification.

The remainder of the paper is organized as follows. DC programming and DCA are briefly presented in Section 2 while Section 3 is devoted to the development of DCA for solving the feature selection S3VM problem (2). Computational experiments are reported in Section 4 and finally Section 5 concludes the paper.

2 Outline of DC programming and DCA

DC programming and DCA constitute the backbone of smooth/nonsmooth nonconvex programming and global optimization. They address the problem of minimizing a function f which is the difference of two convex functions on the whole space $\mathbb{R}^d$ or on a convex set $C \subset \mathbb{R}^d$. Generally speaking, a DC program is an optimization problem of the form :

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

where g, h are lower semi-continuous proper convex functions on $\mathbb{R}^d$. Such a function f is called a DC function, and $g - h$ a DC decomposition of f while g and h are the DC components of f. The convex constraint $x \in C$ can be incorporated in the objective function of (P_{dc}) by using the indicator function on C denoted by χ_C which is defined by $\chi_C(x) = 0$ if $x \in C$, and $+\infty$ otherwise:

$$\inf\{f(x) := g(x) - h(x) : x \in C\ \} = \inf\{\chi_C(x) + g(x) - h(x) : x \in \mathbb{R}^d\}.$$

A convex function θ is called *convex polyhedral* if it is the maximum of a finite family of affine functions, i.e.

$$\theta(x) = \max\{\langle a_i, x\rangle + b : i = 1, ...p\}, a_i \in \mathbb{R}^d.$$

Polyhedral DC optimization occurs when either g or h is polyhedral convex. This class of DC optimization problems, which is frequently encountered in practice, enjoys interesting properties (from both theoretical and practical viewpoints) concerning local optimality and the convergence of DCA ([22]).

Let

$$g^*(y) := \sup\{\langle x, y\rangle - g(x) : x \in \mathbb{R}^d\}$$

be the conjugate function of a convex function g. Then, the following program is called the dual program of (P_{dc}):

$$\alpha_D = \inf\{h^*(y) - g^*(y) : y \in \mathbb{R}^d\}. \qquad (D_{dc})$$

One can prove that $\alpha = \alpha_D$, and there is the perfect symmetry between primal and dual DC programs: the dual to (D_{dc}) is exactly (P_{dc}).

For a convex function θ, the subdifferential of θ at $x_0 \in \text{dom}\ \theta := \{x \in \mathbb{R}^d : \theta(x_0) < +\infty\}$, denoted by $\partial\theta(x_0)$, is defined by

$$\partial\theta(x_0) := \{y \in \mathbb{R}^d : \theta(x) \geq \theta(x_0) + \langle x - x_0, y\rangle, \forall x \in \mathbb{R}^d\}. \qquad (3)$$

The subdifferential $\partial\theta(x_0)$ generalizes the derivative in the sense that θ is differentiable at x_0 if and only if $\partial\theta(x_0) \equiv \{\bigtriangledown_x\theta(x_0)\}$. Recall the well-known property related to subdifferential calculus of a convex function θ:

$$y_0 \in \partial\theta(x_0) \Longleftrightarrow x_0 \in \partial\theta^*(y_0) \Longleftrightarrow \langle x_0, y_0\rangle = \theta(x_0) + \theta^*(y_0). \qquad (4)$$

The complexity of DC programs resides, in the lack of practical optimal globality conditions. Local optimality conditions are then useful in DC programming.

A point x^* is said to be *a local minimizer* of $g - h$ if $g(x^*) - h(x^*)$ is finite and there exists a neighborhood $\mathcal{U}$ of x^* such that

$$g(x^*) - h(x^*) \leq g(x) - h(x), \quad \forall x \in \mathcal{U}. \qquad (5)$$

The necessary local optimality condition for (primal) DC program (P_{dc}) is given by

$$\emptyset \neq \partial h(x^*) \subset \partial g(x^*). \qquad (6)$$

The condition (6) is also sufficient (for local optimality) in many important classes of DC programs, for example, when (P_{dc}) is a polyhedral DC program with h being polyhedral convex function, or when f is locally convex at x^* (see [22]).

A point x^* is said to be *a critical point* of $g - h$ if

$$\partial h(x^*) \cap \partial g(x^*) \neq \emptyset. \qquad (7)$$

The relation (7) is in fact the generalized KKT condition for (P_{dc}) and x^* is also called a generalized KKT point.

Based on local optimality conditions and duality in DC programming, the DCA consists in constructing of two sequences $\{x^l\}$ and $\{y^l\}$ of trial solutions of the primal and dual programs respectively, such that the sequences $\{g(x^l) - h(x^l)\}$ and $\{h^*(y^l) - g^*(y^l)\}$ are decreasing, and $\{x^l\}$ (resp. $\{y^l\}$) converges to a primal feasible solution x^* (resp. a dual feasible solution y^*) satisfying local optimality conditions:

$$x^* \in \partial g^*(y^*), \quad y^* \in \partial h(x^*). \qquad (8)$$

It implies, according to (4) that $x^*$and y^* are critical points (KKT points) of $g - h$ and $h^* - g^*$ respectively.

The main idea behind DCA is to replace in the primal DC program (P_{dc}), at the current point x^l of iteration l, the second component h with its affine minorization defined by

$$h_l(x) := h(x^l) + \ \langle x - x^l, y^l\rangle, \quad y^l \in \partial h(x^l)$$

to give birth to the primal convex program of the form

$$(P_l) \quad \inf\{g(x) - h_l(x) :\ x \in \mathbb{R}^n\} \Longleftrightarrow \inf\{g(x) - \langle x, y^l\rangle\ : x \in \mathbb{R}^d\}$$

whose an optimal solution is taken as x^{l+1}.

Dually, a solution x^{l+1} of (P_l) is then used to define the dual convex program (D_{l+1}) by replacing in (D_{dc}) g^* with its affine minorization defined by

$$(g^*)_l(y) := g^*(y^l) + \langle y - y^l, x^{l+1}\rangle, \quad x^{l+1} \in \partial g^*(y^l)$$

to obtain

$$(D_{l+1}) \quad \inf\{h^*(y) - [g^*(y^l) + \langle y - y^l, x^{l+1}\rangle] y \in \mathbb{R}^d\}$$

whose an optimal solution is taken as y^{l+1}. The process is repeated until convergence.

DCA performs so a double linearization with the help of the subgradients of h and g^*. According to relation (4) it is easy to see that the optimal solution set of (P_l) (resp. (D_{l+1})) is nothing but $\partial g^*(y^l)$ (resp. $\partial h(x^{l+1})$). Hence, we can say that DCA is an iterative primal-dual subgradient method that yields the next scheme: (starting from given $x^0 \in \text{dom } \partial h$)

$$y^l \in \partial h(x^l); \quad x^{l+1} \in \partial g^*(y^l), \ \forall l \geq 0. \tag{9}$$

The generic DCA scheme is shown below.

DCA scheme

Initialization: Let $x^0 \in \mathbb{R}^d$ be a best guess, $l = 0$.

Repeat

- Calculate $y^l \in \partial h(x^l)$
- Calculate $x^{l+1} \in \arg\min\{g(x) - h(x^l) - \langle x - x^l, y^l\rangle : x \in \mathbb{R}^d\} \quad (P_l)$
- $l = l + 1$

Until convergence of $\left\{x^l\right\}$.

Convergence properties of DCA and its theoretical basic can be found in [22,35]. It is worth mentioning that

i) DCA is a descent method (*without line search*): the sequences $\{g(x^l) - h(x^l)\}$ is decreasing.
ii) If $g(x^{l+1}) - h(x^{l+1}) = g(x^l) - h(x^l)$, then x^l is a critical point of $g - h$. In such a case, DCA terminates at l-th iteration.
iii) If the optimal value α of problem (P_{dc}) is finite and the infinite sequences $\{x^l\}$ is bounded then every limit point x^* of the sequences $\{x^l\}$ is a critical point of $g - h$.
iv) DCA has a *linear convergence* for general DC programs.
v) DCA has a *finite convergence* for polyhedral DC programs. Moreover, if h is polyhedral and h is differentiable at x^* then x^* is a local optimizer of (P_{dc}).

A deeper insight into DCA has been described in [22]. For instant it is worth to mention the main feature of DCA: DCA is constructed from DC components and their conjugates but not the DC function f itself which has infinitely many DC decompositions, there are as many DCA as there are DC decompositions. Such decompositions play an extremely critical role in determining the speed of convergence, stability, robustness, and globality of sought solutions. It is important to study various equivalent DC forms of a DC problem. This flexibility of DC programming and DCA is of particular interest from both a theoretical and an algorithmical point of view.

Note that with appropriate DC decompositions and suitably equivalent DC reformulations, DCA permits to recover most of standard methods in convex and nonconvex programming. For a complete study of DC programming and DCA the reader is referred to [20,22,35].

3 Feature selection in S3VM by DC programming and DCA

Assume that the m labeled points and p unlabeled points are represented by the matrix $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times n}$, respectively. D is a $m \times m$ diagonal matrix where $D_{i,i} = y_i, \forall i = 1, \ldots, m$. Denote by e the vector of ones in the appropriate vector space. For each labeled point x_i $(i = 1, \ldots, m)$, we introduce a new variable ξ_i which represents the misclassification error. Similarly, for each unlabeled point x_i $(i = (m+1), \ldots, (m+p))$, we define r_i and s_i for two possible misclassification errors. Let r and s be vectors in $\mathbb{R}^p$ who i^{th} component are r_i and s_i respectively. Then, the final class of unlabeled x_i corresponds to the one that has minimal misclassification. The feature selection in S3VM problem (2) can be rewritten as follows:

$$\min\left\{F_0(w, b, \xi, r, s) := \alpha\langle e, \xi\rangle + \beta\langle e, \min\{r, s\}\rangle + \|w\|_0 : (w, b, \xi, r, s) \in K\right\}, \tag{10}$$

where K is polyhedral convex set defined by

$$\left\{ \begin{array}{l} (w,b,\xi,r,s) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R}^p : D(Aw - eb) + \xi \geq e, \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad Bw - eb + r \geq e, \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad -Bw + eb + s \geq e, \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \xi, r, s \geq 0. \end{array} \right\}. \tag{11}$$

3.1 DC approximation functions of ℓ_0

Let us define the step function $\pi : \mathbb{R} \to \mathbb{R}$ by

$$\pi(x) = 1 \text{ for } x \neq 0 \text{ and } \pi(x) = 0 \text{ for } x = 0.$$

Hence

$$\|w\|_0 = \sum_{i=1}^{n} \pi(w_i).$$

3.1.1 A piecewise exponential approximation

The first non convex approximation was introduced by Bradley and Mangasarian in [4] where the step function s is approximated by:

$$\pi(x) \simeq 1 - \varepsilon^{-\lambda x}, \lambda > 0, \tag{12}$$

and then the zero-norm $\|w\|_0$ is approximated by $\|w\|_0 \simeq \sum\limits_{i=1}^{n}(1 - \varepsilon^{-\lambda w_i})$. The SLA (Successive Linear Approximation) method proposed in [4], for solving the resulting Feature Selection concaVe minimization problem (FSV), consists of solving at each iteration one linear program (note that SLA is a special DCA scheme applied to FSV).

In [25], the authors proposed another DC formulation of the step function π. The comparative results on several datasets show the superiority of this approach over SLA proposed in [4] (this shows the nice effect of DC decompositions for DCA) as well as SVMs using ℓ_1-norm. Motivated by the efficiency of the DC decomposition of proposed in [25], we investigate a similar DC formulation for the approximate problem of (10).

For $x \in \mathbb{R}$, let η_1 be the function defined by ($\lambda > 0$)

$$\eta_1(x, \lambda) = 1 - \varepsilon^{-\lambda|x|}. \tag{13}$$

In what follows, for a given λ, we will use $\eta_1(x)$ instead of $\eta_1(x, \lambda)$. It is clear that for moderate values of λ, one can obtain a very adequate approximation of s. Then the approximation of zero-norm $\|w\|_0$ is given by: $\|w\|_0 \simeq \sum_{i=1}^{n} \eta_1(w_i)$. We first express η_1 as a DC function as follows: $\eta_1(x) = g_1(x) - h_1(x)$, where

$$g_1(x) := \lambda|x| \text{ and } h_1(x) := \lambda|x| - 1 + \varepsilon^{-\lambda|x|}. \tag{14}$$

With this approximation, we approximate the objective function of (10) by the following

$$F_1(w,b,\xi,r,s) = G_1(w,b,\xi,r,s) - H_1(w,b,\xi,r,s),$$

where

$$G_1(w,b,\xi,r,s) = \alpha\langle e, \xi\rangle + \sum_{i=1}^{n} g_1(w_i) \tag{15}$$

and

$$H_1(w,b,\xi,r,s) = -\beta\langle e, \min\{r,s\}\rangle + \sum_{i=1}^{n} h_1(w_i) \tag{16}$$

are convex functions (especially, G_1 is polyhedral convex). Hence, the approximate problem of (10) is a polyhedral DC program of the form

$$\min \{G_1(w,b,\xi,r,s) - H_1(w,b,\xi,r,s) : (w,b,\xi,r,s) \in K\}. \tag{17}$$

According to general DCA scheme in Section 2, DCA applied on (17) consists of computing the two sequences $\{(w^l,b^l,\xi^l,r^l,s^l)\}$ and $\{(\bar{w}^l,\bar{b}^l,\bar{\xi}^l,\bar{r}^l,\bar{s}^l)\}$ such that

$$(\bar{w}^l,\bar{b}^l,\bar{\xi}^l,\bar{r}^l,\bar{s}^l) \in \partial H_1(w^l,b^l,\xi^l,r^l,s^l)$$

and $(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1})$ is an optimal solution of the convex problem

$$\min \left\{ \sum_{i=1}^{n} \max\{\lambda w_i, -\lambda w_i\} + \alpha\langle e,\xi\rangle - \langle (\bar{w}^l,\bar{b}^l,\bar{\xi}^l,\bar{r}^l,\bar{s}^l),(w,b,\xi,r,s)\rangle \;:\; (w,b,\xi,r,s) \in K \right\}. \tag{18}$$

Following the computation of a subgradient of H_1, we can take $(\bar{w}^l,\bar{b}^l,\bar{\xi}^l,\bar{r}^l,\bar{s}^l) \in \partial H_1(w^l,b^l,\xi^l,r^l,s^l)$ as follows:

$$\bar{w}_i^l = \lambda(1-\epsilon^{-\lambda w_i^l}) \text{ if } w_i^l \geq 0,\ -\lambda(1-\epsilon^{\lambda w_i^l}) \text{ if } w_i^l < 0, \quad \forall i = 1,\ldots,n, \tag{19}$$

$$\bar{r}_i^l = -\beta \text{ if } r_i^l < s_i^l,\ 0 \text{ if } r_i^l > s_i^l,\ -\beta\mu \text{ if } r_i^l = s_i^l, \quad \forall i = 1,\ldots,p,\ \mu \in [0,1] \tag{20}$$

$$\bar{s}_{l_i} = 0 \text{ if } r_i^l < s_i^l,\ -\beta \text{ if } r_i^l > s_i^l,\ -\beta(1-\mu) \text{ if } r_i^l = s_i^l, \quad \forall i = 1,\ldots,p,\ \mu \in [0,1] \tag{21}$$

and

$$\bar{b}^l = 0,\ \bar{\xi}^l = 0 \tag{22}$$

Furthermore, solving (18) amounts to solving the following linear program:

$$\begin{cases} \min\ \langle e,t\rangle + \alpha\langle e,\xi\rangle - \langle (\bar{w}^l,\bar{b}^l,\bar{\xi}^l,\bar{r}^l,\bar{s}^l),(w,b,\xi,r,s)\rangle, \\ s.t.\ \ (w,b,\xi,r,s) \in K \\ \qquad t_i \geq \lambda w_i,\ t_i \geq -\lambda w_i, \forall i = 1,\ldots,n, \end{cases} \tag{23}$$

Therefor, DCA applied to (17) can be described as follows:

Algorithm 1 S3VM-PiE

initializations: let τ be a tolerance sufficiently small, set $l = 0$.
Choose $(w^0,b^0,\xi^0,r^0,s^0) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R}^p$.
repeat
1. Set $(\bar{w}^l,\bar{b}^l,\bar{\xi}^l,\bar{r}^l,\bar{s}^l) = (\bar{w}^l,0,0,\bar{r}^l,\bar{s}^l)$ following (19), (20) and (21).
2. Solve the linear program (23) to obtain $(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1})$.
3. $l = l+1$.

until $\|(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1}) - (w^l,b^l,\xi^l,r^l,s^l)\| \leq \tau(\|(w^l,b^l,\xi^l,r^l,s^l)\|+1)$.

3.1.2 A DC polyhedral approximation

In [37], the author proposed a DC polyhedral approximation which has been successfully applied in [38]. We will now incorporate this approximation to our problem (10).

For $x \in \mathbb{R}$, let η_2 be the function defined by defined as follows

$$\eta_2(x) := \min\{1,\lambda|x|\} = 1 + \lambda|x| - \max\{1,\lambda|x|\}. \tag{24}$$

Hence, $\|w\|_0$ is approximated by: $\|w\|_0 \simeq \sum_{i=1}^n \eta_2(w_i)$. It is easy to see that η_2 is a DC function with following DC decomposition $\eta_2(x) = g_2(x) - h_2(x)$ where

$$g_2(x) = 1 + \lambda|x| \text{ and } h_2(x) = \max\{1,\lambda|x|\}. \tag{25}$$

With this approximation function, the approximate problem of (10) can be represented as follows:

$$\min\left\{F_2(w,b,\xi,r,s) := \alpha\langle e,\xi\rangle + \beta\langle e,\min\{r,s\}\rangle + \sum_{i=1}^{n}\eta_2(w_i) : (w,b,\xi,r,s)\in K\right\}. \tag{26}$$

Clearly, $F_2(w,b,\xi,r,s)$ is a DC function:

$$F_2(w,b,\xi,r,s) = G_2(w,b,\xi,r,s) - H_2(w,b,\xi,r,s),$$

where

$$G_2(w,b,\xi,r,s) = \alpha\langle e,\xi\rangle + \sum_{i=1}^{n} g_2(w_i) \tag{27}$$

and

$$H_2(w,b,\xi,r,s) = -\beta\langle e,\min\{r,s\}\rangle + \sum_{i=1}^{n} h_2(w_i) \tag{28}$$

are convex polyhedral functions. Hence problem (26) can be expressed as:

$$\min\{G_2(w,b,\xi,r,s) - H_2(w,b,\xi,r,s) : (w,b,\xi,r,s)\in K\}. \tag{29}$$

Hence, according to the generic DCA scheme, at each iteration l, we have to compute $(\bar{w}^l,\bar{b}^l,\bar{\xi}^l,\bar{r}^l,\bar{s}^l) \in \partial H_2(w^l,b^l,\xi^l,r^l,s^l)$ and then solve the convex program

$$\min\left\{\begin{array}{l} G_2(w,b,\xi,r,s) - \langle(\bar{w}^l,\bar{b}^l,\bar{\xi}^l,\bar{r}^l,\bar{s}^l),(w,b,\xi,r,s)\rangle \\ \qquad : (w,b,\xi,r,s)\in K \end{array}\right\}. \tag{30}$$

to obtain $(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1})$.
Similarly to the computation of a subgradient of H_1 (note that H_2 differs H_1 from the second term that affects only the variable w), we have $(\bar{w}^l,\bar{b}^l,\bar{\xi}^l,\bar{r}^l,\bar{s}^l) = (\bar{w}^l,0,0,\bar{r}^l,\bar{s}^l)$ with $\bar{r}^l$ (resp. $\bar{s}^l$) being defined in (20) (resp. (21)) and

$$\overline{w}_i^l = \begin{cases} 0 & \text{if } -1/\lambda \le w_i^l \le 1/\lambda, \\ \lambda & \text{if } w_i^l > 1/\lambda, \\ -\lambda & \text{if } w_i^l < -1/\lambda, \end{cases} \qquad i = 1,\ldots n \tag{31}$$

On the other hand, solving (30) is equivalent to solving (23). Finally, the DCA applied to (29) is described as follows:

Algorithm 2 S3VM-PoDC

initializations: let τ be a tolerance sufficiently small, set $l = 0$.
Choose $(w^0,b^0,\xi^0,r^0,s^0) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R}^p$.
repeat
1. Set $(\bar{w}^l,\bar{b}^l,\bar{\xi}^l,\bar{r}^l,\bar{s}^l) = (\bar{w}^l,0,0,\bar{r}^l,\bar{s}^l)$ following (31), (20) and (21).
2. Solve the linear program (23) to obtain $(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1})$.
3. $l = l+1$.

until $\|(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1}) - (w^l,b^l,\xi^l,r^l,s^l)\| \le \tau(\|(w^l,b^l,\xi^l,r^l,s^l)\|+1)$.

3.1.3 A piecewise linear approximation

We consider another DC approximation of ℓ_0 which is a piecewise linear function ([28]). Let $\eta_3 : \mathbb{R} \to \mathbb{R}$ be the function defined by

$$\eta_3(x) = \begin{cases} 0 & \text{if } |x| \leq a, \\ 1 & \text{if } |x| \geq b, \\ \frac{|x|-a}{b-a} & \text{if } a \leq |x| \leq b. \end{cases} \tag{32}$$

where $0 \leq a < b$. Then an approximation of zero-norm $\|w\|_0$ can be: $\|w\|_0 \simeq \sum_{i=1}^{n} \eta_3(w_i)$ and the resulting approximation problem of (10) takes the form:

$$\min \left\{ F_3(w, b, \xi, r, s) := \alpha\langle e, \xi\rangle + \beta\langle e, \min\{r, s\}\rangle + \sum_{i=1}^{n} \eta_3(w_i) : (w, b, \xi, r, s) \in K \right\}. \tag{33}$$

Observe that η_3 is a DC function:

$$\eta_3(x) = \max\left(\min(1, \tfrac{|x|-a}{b-a}), 0\right) = 1 + g_3(x) - h_3(x),$$

where $g_3(x) = \frac{1}{b-a}\max(a, |x|)$ and $h_3(x) = \frac{1}{b-a}\max(b, |x|)$ are clearly convex functions. Let G_3 and H_3 be the functions defined by:

$$G_3(w, b, \xi, r, s) = \alpha\langle e, \xi\rangle + \sum_{i=1}^{n} g_3(w_i) \tag{34}$$

and

$$H_3(w, b, \xi, r, s) = -\beta\langle e, \min\{r, s\}\rangle + \sum_{i=1}^{n} h_3(w_i). \tag{35}$$

Clearly, G_3 and H_3 are polyhedral convex functions. Then the problem (33) is a DC program of the form

$$\min\{G_3(w, b, \xi, r, s) - H_3(w, b, \xi, r, s) : (w, b, \xi, r, s) \in K\}. \tag{36}$$

Hence DCA applied to (36) amounts to computing the two sequences $\{(w^l, b^l, \xi^l, r^l, s^l)\}$ and $\{(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l)\}$ such that

$$(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l) \in \partial H_3(w^l, b^l, \xi^l, r^l, s^l)$$

and $(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1})$ is an optimal solution of the following convex problem

$$\min \left\{ \begin{array}{l} \alpha\langle e, \xi\rangle + \frac{1}{b-a}\sum\limits_{i=1}^{n} \max(a, |x|) - \langle(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l), (w, b, \xi, r, s)\rangle \\ \qquad\qquad : (w, b, \xi, r, s) \in K \end{array} \right\} \tag{37}$$

which is equivalent to

$$\begin{cases} \min \ \frac{1}{b-a}\langle e, t\rangle + \alpha\langle e, \xi\rangle - \langle\overline{w}, w\rangle - \langle\overline{r}, r\rangle - \langle\overline{s}, s\rangle \\ \text{s.t.} \ \ (w, b, \xi, r, s) \in K, t_i \geq a, t_i \geq w_i, t_i \geq -w_i. \end{cases} \tag{38}$$

Similarly to the computation of a subgradient of H_1 and H_2, we have $(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l) = (\bar{w}^l, 0, 0, \bar{r}^l, \bar{s}^l)$ with $\bar{r}^l$ (resp. $\bar{s}^l$) being defined in (20) (resp. (21)) and

$$\overline{w}_i^l = \begin{cases} 0 & \text{if } |w_i^l| \leq b, \\ \frac{1}{b-a} & \text{if } w_i^l \geq 0, \\ -\frac{1}{b-a} & \text{if } w_i^l < 0 \end{cases} \qquad i = 1, ..., n. \tag{39}$$

Finally, DCA scheme applied on (36) is described as follows:

Algorithm 3 S3VM-PiL

initializations: let τ be a tolerance sufficiently small, set $l = 0$.
Choose $(w^0, b^0, \xi^0, r^0, s^0) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R}^p$.
repeat
1. Set $(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l) = (\bar{w}^l, 0, 0, \bar{r}^l, \bar{s}^l)$ following (39), (20) and (21).
2. Solve the linear program (38) to obtain $(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1})$.
3. $l = l + 1$.

until $\|(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1}) - (w^l, b^l, \xi^l, r^l, s^l)\| \leq \tau(\|(w^l, b^l, \xi^l, r^l, s^l)\| + 1)$.

3.1.4 SCAD (Smoothly Clipped Absolute Deviation) approximation

The SCAD (Smoothly Clipped Absolute Deviation) penalty function has been proposed for the first time by J. Fan and R. Li ([12]) in the context off feature selection in regression. Later, in [26], the authors reformulated the SCAD penalty function as a DC function and then developed an efficient algorithm based on DC programming and DCA (DC Algorithm) for solving the resulting optimization problem. The SCAD penalty function is expressed as follows

$$\eta_4(x) = \begin{cases} \lambda x & if\ 0 \leq x \leq \lambda, \\ -\frac{x^2 - 2\gamma\lambda x + \lambda^2}{2(\gamma - 1)} & if\ \lambda < x \leq \gamma\lambda, \\ \frac{(\gamma+1)\lambda^2}{2} & if\ x > \gamma\lambda, \\ \eta(-x) & if\ x < 0, \end{cases}$$

where $\lambda > 0$ and $\gamma > 2$ are two tuning parameters.
Let h_4 be the function given by:

$$h_4(x) = \begin{cases} 0 & \text{if } 0 \leq x \leq \lambda, \\ \frac{1}{2(\gamma-1)}(x - \lambda)^2 & \text{if } \lambda < x \leq \lambda\gamma, \\ \lambda x - \frac{(\gamma+1)\lambda^2}{2} & \text{if } x > \lambda\gamma, \\ h_2(-x) & \text{if } x < 0. \end{cases} \tag{40}$$

Clearly h_4 is a convex functions and:

$$\eta_4(x) = g_1(x) - h_4(x),$$

with g_1 being defined in (14).
Hence, the resulting approximation problem of (10) takes the form:

$$\min \left\{ F_4(w, b, \xi, r, s) := \alpha\langle e, \xi\rangle + \beta\langle e, \min\{r, s\}\rangle + \sum_{i=1}^{n} \eta_4(w_i) : (w, b, \xi, r, s) \in K \right\}. \tag{41}$$

Clearly, $F_4(w, b, \xi, r, s)$ is a DC function with the following DC decomposition $F_4(w, b, \xi, r, s) = G_1(w, b, \xi, r, s) - H_4(w, b, \xi, r, s)$, with G_1 being defined in (15) and

$$H_4(w, b, \xi, r, s) = -\beta\langle e, \min\{r, s\}\rangle + \sum_{i=1}^{n} h_4(w_i). \tag{42}$$

Then the problem (41) can be expressed as:

$$\min \left\{ G_1(w, b, \xi, r, s) - H_4(w, b, \xi, r, s) : (w, b, \xi, r, s) \in K \right\}. \tag{43}$$

Note that, since G_1 is a polyhedral convex function,(43) becomes a DC polyhedral program. In the same way to the computation of a subgradient of H_1 (as well as H_2 and H_3), we get $(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l) = (\bar{w}^l, 0, 0, \bar{r}^l, \bar{s}^l)$ with $\bar{r}^l$ (resp. $\bar{s}^l$) being defined in (20) (resp. (21)) and

$$\overline{w}_i^l = \begin{cases} 0 & \text{if } -\lambda \leq w_i^l \leq \lambda, \\ (\gamma - 1)^{-1}(w_i^l - \lambda) & \text{if } \lambda < w_i^l \leq \gamma\lambda, \\ (\gamma - 1)^{-1}(w_i^l + \lambda) & \text{if } -\gamma\lambda \leq w_i^l < -\lambda, \\ \lambda & \text{if } w_i^l > \gamma\lambda, \\ -\lambda & \text{if } w_i^l < -\gamma\lambda, \end{cases} \qquad i = 1, ...n. \tag{44}$$

Finally, DCA applied to (43) is described as follows:

Algorithm 4 S3VM-SCAD

initializations: let τ be a tolerance sufficiently small, set $l = 0$.
Choose $(w^0, b^0, \xi^0, r^0, s^0) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R}^p$.
repeat
1. Set $(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l) = (\bar{w}^l, 0, 0, \bar{r}^l, \bar{s}^l)$ following (44), (20) and (21).
2. Solve the linear program (23) to obtain $(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1})$.
3. $l = l + 1$.

until $\|(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1}) - (w^l, b^l, \xi^l, r^l, s^l)\| \leq \tau(\|(w^l, b^l, \xi^l, r^l, s^l)\| + 1)$.

3.1.5 Logarithm approximation

In [53], the authors proposed the following approximation for $\|w\|_0$:

$$\|w\|_0 \approx \sum_{j=1}^{n} \ln(\varepsilon + |w_j|), \tag{45}$$

where $0 < \varepsilon << 1$. Hence the resulting approximate problem of (10) is written as:

$$\min \left\{ F_5(w, b, \xi, r, s) := \alpha\langle e, \xi\rangle + \beta\langle e, \min\{r, s\}\rangle + \sum_{i=1}^{n} \ln(\varepsilon + |w_j|) : (w, b, \xi, r, s) \in K \right\}. \tag{46}$$

By introducing a non-negative variable $u \geq 0$ and the constraint relaxation $-u \leq w \leq u$, we obtain the following program

$$\min \begin{cases} \overline{F}_5(w, b, \xi, r, s, u) := \alpha\langle e, \xi\rangle + \beta\langle e, \min\{r, s\}\rangle + \sum\limits_{i=1}^{n} \ln(\varepsilon + u_j) \\ \qquad\qquad : (w, b, \xi, r, s, u) \in \bar{K} \end{cases} \tag{47}$$

where

$$\bar{K} := K \cap \{u \in \mathbb{R}^n : -u \leq w \leq u, u \geq 0\}. \tag{48}$$

Since $\overline{F}_5$ is a concave function, it is DC, and a DC decomposition of $\overline{F}_5$ can be immediately given by

$$\overline{F}_5(w, b, \xi, r, s, u) = G_5(w, b, \xi, r, s, u) - H_5(w, b, \xi, r, s, u) \tag{49}$$

where $G_5(w, b, \xi, r, s, u) := \chi_{\bar{K}}$ and $H_5(w, b, \xi, r, s, u) = -\overline{F}_5(w, b, \xi, r, s, u)$.

Then the problem (47) can be expressed as:

$$\min \left\{ G_5(w, b, \xi, r, s, u) - H_5(w, b, \xi, r, s, u) : (w, b, \xi, r, s, u) \in \bar{K} \right\} \tag{50}$$

which is a DC polyhedral program since G_5 is polyhedral convex. Hence, applying DCA to (50) amounts to computing, at each iteration l, the two sequences $\{(w^l, b^l, \xi^l, r^l, s^l, u^l)\}$ and $\{(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l), \bar{u}^l)\}$ such that

$$(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l, \bar{u}^l) \in \partial H_5(w^l, b^l, \xi^l, r^l, s^l, u^l),$$

and $(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1}, u^{l+1})$ is an optimal solution of the following convex problem

$$\min \left\{ -\langle(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l), (w, b, \xi, r, s)\rangle : (w, b, \xi, r, s, u) \in \bar{K} \right\}. \tag{51}$$

The computation of a subgradient of H_5 leads us to take $(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l, \bar{u}^l) = (0, 0, 0, \bar{r}^l, \bar{s}^l, \bar{u}^l)$ with $\bar{r}^l$ (resp. $\bar{s}^l$) being defined in (20) (resp. (21)) and

$$\overline{u}^l = \frac{1}{u_i^l + \varepsilon},\ i = 1, ...n. \tag{52}$$

Hence, DCA applied to (50) is described as follows:

Algorithm 5 S3VM-Log

initializations: let τ be a tolerance sufficiently small, set $l = 0$.
Choose $(w^0, b^0, \xi^0, r^0, s^0, u^0) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R}^p \times \mathbb{R}^n$.
repeat
1. Set $(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l, \bar{u}^l) = (0, 0, 0, \bar{r}^l, \bar{s}^l, \bar{u}^l)$ following (52), (20) and (21).
2. Solve the linear program (51) to obtain $(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1}, u^{l+1})$.
3. $l = l + 1$.

until $\|(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1}, u^{l+1}) - (w^l, b^l, \xi^l, r^l, s^l, u^l)\| \leq \tau(\|(w^l, b^l, \xi^l, r^l, s^l, u^l)\| + 1)$.

3.1.6 Convergence

For simplifying the presentation, we use a common function F to name the objective function of the resulting optimization problems, that is $F \in \{F_1; F_2; F_3; F_4; F_5\}$.

Theorem 1 *(Convergence properties of Algorithms S3VM-PiE, S3VM-PoDC, S3VM-PiL, S3VM-SCAD, S3VM-Log)*

(i) DCA generates a sequence $\{(w^l, b^l, \xi^l, r^l, s^l)\}$ *(resp.* $\{(w^l, b^l, \xi^l, r^l, s^l, u^l)\}$*) such that the sequence* $\{F(w^l, b^l, \xi^l, r^l, s^l)\}$ *(resp.* $\{F(w^l, b^l, \xi^l, r^l, s^l, u^l)\}$*) is monotonously decreasing.*

(ii) The sequence $\{(w^l, b^l, \xi^l, r^l, s^l)\}$ *(resp.* $\{(w^l, b^l, \xi^l, r^l, s^l, u^l)\}$*) converges to* $(w^*, b^*, \xi^*, r^*, s^*)$ *(resp.* $(w^*, b^*, \xi^*, r^*, s^*, u^*)$ *) after a finite number of iterations.*

(iii) The point $(w^*, b^*, \xi^*, r^*, s^*)/(w^*, b^*, \xi^*, r^*, s^*, u^*)$ *is a critical point of the objective function* F*.*

(iv) The point $(w^*, b^*, \xi^*, r^*, s^*)/(w^*, b^*, \xi^*, r^*, s^*, u^*)$ *is almost always a local minimizer of the corresponding problem (17), (29), (36), (43), (59). Especially,*

– *In S3VM-PoDC, if*

$$\begin{array}{ll} w_i^* \notin \{-\frac{1}{\lambda}; \frac{1}{\lambda}\} & \forall i = 1..n \\ r_i^* \neq s_i^* & \forall i = 1..p \end{array} \tag{53}$$

then $(w^*, b^*, \xi^*, r^*, s^*)$ *is a local minimizer of (29).*

– *In S3VM-PiL, if*

$$\begin{array}{ll} w_i^* \notin \{-b, b\} & \forall i = 1..n \\ r_i^* \neq s_i^* & \forall i = 1..p \end{array} \tag{54}$$

then $(w^*, b^*, \xi^*, r^*, s^*)$ *is a local minimizer of (36).*

Proof: (i) and (iii) are direct consequences of the convergence properties of general DC programs while (ii) is a convergence property of a DC polyhedral program. Only (iv) needs a proof.

In *S3VM-PoDC*, the second DC component of (29), says H_2 is a polyhedral convex function. Moreover, if the condition (53) holds then H_2 is differentiable at $(w^*, b^*, \xi^*, r^*, s^*)$. Then using the convergence property of DCA for DC polyhedral program ([22]), we can conclude that $(w^*, b^*, \xi^*, r^*, s^*)$ is a local minimizer of (29). The proof is similar for *S3VM-PiL* since its second DC component H_3 is also polyhedral convex.

In *S3VM-PiE*, G_1 is a polyhedral convex function, so is G_1^*. Hence the dual DC program of (17) is a polyhedral DC program. The necessary local optimality condition

$$\emptyset \neq \partial G_1^*(\bar{w}^*, \bar{b}^*, \bar{\xi}^*, \bar{r}^*, \bar{s}^*) \subset \partial H_1^*(\bar{w}^*, \bar{b}^*, \bar{\xi}^*, \bar{r}^*, \bar{s}^*)$$

becomes sufficient if G_1^* is differentiable at $(\bar{w}^*, \bar{b}^*, \bar{\xi}^*, \bar{r}^*, \bar{s}^*)$, the limit point of the sequence $(\bar{w}^k, \bar{b}^k, \bar{\xi}^k, \bar{r}^k, \bar{s}^k)$ generated by *S3VM-PiE*. Since G_1^* is a polyhedral function, it is differentiable everywhere except for a set of measure zero. Therefore, one can say that $(\bar{w}^*, \bar{b}^*, \bar{\xi}^*, \bar{r}^*, \bar{s}^*)$ is almost always a local solution of the dual DC program of (17).

On another hand, according to the property of transportation of local minimizers in DC programming we have the following (see [22]): let $(\bar{w}^*, \bar{b}^*, \bar{\xi}^*, \bar{r}^*, \bar{s}^*)$ be a local solution to the dual program of (17) and $(w^*, b^*, \xi^*, r^*, s^*) \in \partial G_1^*(\bar{w}^*, \bar{b}^*, \bar{\xi}^*, \bar{r}^*, \bar{s}^*)$. If H_1 is differentiable at $(w^*, b^*, \xi^*, r^*, s^*)$ then $(w^*, b^*, \xi^*, r^*, s^*)$ is a local solution of (17). Combining this property

with the facts that $(\bar{w}^*, \bar{b}^*, \bar{\xi}^*, \bar{r}^*, \bar{s}^*)$ is almost always a local solution to the dual DC program of (17) and H_1 is differentiable almost everywhere (except when $r_i^* = s_i^*$), we conclude that $(w^*, b^*, \xi^*, r^*, s^*)$ is almost always a local minimizer to Problem (17).

For *S3VM-SCAD* (resp. *S3VM-Log*), the proof is similar as its first DC component G_4 (resp. G_5) is polyhedral convex. The proof is then complete. ■

3.2 A continuous reformulation of ℓ_0 via an exact penalty technique

In [29,47], the authors reformulated ℓ_0-norm problem as a continuous nonconvex program. More precisely, the ℓ_0-norm is first equivalently formulated as a combinatorial optimization problem and then the last problem is reformulated as a DC program via an exact penalty technique ([27]). We will use the same techniche for our problem (10).

We suppose that K is bounded in the variable w, i.e. $K \subset \Pi_{i=1}^n [a_i, b_i] \times \mathbb{R}$ where $a_i, b_i \in \mathbb{R}$ such that $a_i \leq 0 < b_i$ for $i = 1, ..., n$. Let $c_i := \max\{|w_i| : w_i \in [a_i, b_i]\} = \max\{|a_i|, |b_i|\}$ for $i = 1, ..., n$. Define the binary variable $u_i \in \{0, 1\}$ as

$$u_j = \begin{cases} 1 \text{ if } w_j \neq 0, \\ 0 \text{ if } w_j = 0 \end{cases} \qquad \forall j = 1..n. \tag{55}$$

The above relation can be expressed by:

$$|w_j| \leq c_j u_j \quad \forall j = 1..n.$$

Then, the problem (10) can be written as:

$$\begin{cases} \min F_6(w, b, \xi, r, s, u) := \alpha\langle e, \xi\rangle + \beta\langle e, \min\{r, s\}\rangle + \langle e, u\rangle \\ s.t. \quad (w, b, \xi, r, s) \in K, \\ \qquad |w_j| \leq c_j u_j \quad \forall j = 1..n, \\ \qquad u \in \{0, 1\}^n \end{cases}. \tag{56}$$

Let $p(u) := \sum_{j=1}^{n} \min\{u_j, (1 - u_j)\}$ be the polyhedral concave function and L be the polyhedral convex set defined as:

$$L := \left\{ \begin{array}{l} (w, b, \xi, r, s) \in K, \\ |w_j| \leq c_j u_j \quad \forall j = 1..n, \\ u \in [0, 1]^n \end{array} \right\}. \tag{57}$$

The problem (56) is equivalent to

$$\begin{cases} \min F_6(w, b, \xi, r, s, u) \\ s.t. \quad (w, b, \xi, r, s) \in L, \\ \qquad p(u) \leq 0 \end{cases}. \tag{58}$$

Using an exact penalty technique in DC programming ([27]), we can reformulate (58) as a continuous optimization problem (κ, called penalty parameter, is sufficient large positive number):

$$\min\left\{\bar{F}_6(w, b, \xi, r, s, u) := F_6(w, b, \xi, r, s, u) + \kappa p(u) : (w, b, \xi, r, s, u) \in L\right\}. \tag{59}$$

Proposition 1 *There is $\kappa_0 \geq 0$ such that for every $\kappa > \kappa_0$ problems (10) and (59) are equivalent, in the sense that they have the same optimal value and $(w^*, b^*, \xi^*, r^*, s^*)$ is a solution of (10) iff there is $u^* \in \{0, 1\}^n$ such that $(w^*, b^*, \xi^*, r^*, s^*, u^*)$ is a solution of (59).*

Proof Direct consequences of Theorem 8 in [27].

Since F_6 and p are concave, the problem (59) is a DC program:

$$\bar{F}_6(w,b,\xi,r,s,u) = G_6(w,b,\xi,r,s,u) - H_6(w,b,\xi,r,s,u) \tag{60}$$

where

$$G_6(w,b,\xi,r,s,u) := \chi_L(w,b,\xi,r,s,u) \tag{61}$$

and

$$H_6(w,b,\xi,r,s,u) := -F_6(w,b,\xi,r,s,u) - \kappa p(u). \tag{62}$$

Furthermore, since G_6 and H_6 are polyhedral convex functions, (59) is a DC polyhedral program. According to generic DCA scheme, DCA applied to (59) can be described as follows: at each iteration l, we compute two sequences $\{(w^l,b^l,\xi^l,r^l,s^l,u^l)\}$ and $\{(\overline{w}^l,\overline{b}^l,\overline{\xi}^l,\overline{r}^l,\overline{s}^l,\overline{u}^l)\}$ such that

$$(\overline{w}^l,\overline{b}^l,\overline{\xi}^l,\overline{r}^l,\overline{s}^l,\overline{u}^l) \in \partial H_6(w^l,b^l,\xi^l,r^l,s^l,u^l) \tag{63}$$

and $(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1},u^{l+1})$ is an solution of the following convex problem

$$\min\left\{-\left\langle(\overline{w}^l,\overline{b}^l,\overline{\xi}^l,\overline{r}^l,\overline{s}^l,\overline{u}^l),(w,b,\xi,r,s,u)\right\rangle : (w,b,\xi,r,s,u) \in L.\right\} \tag{64}$$

The computation of a subgradient of H_6 leads us to take $(\overline{w}^l,\overline{b}^l,\overline{\xi}^l,\overline{r}^l,\overline{s}^l,\overline{u}^l) = (0,0,0,\overline{r}^l,\overline{s}^l,\overline{u}^l)$ with $\overline{r}$ (resp. $\overline{s}$) defined as in (20) (resp. (21)) and

$$\overline{u}_i^l = \begin{cases} +\kappa & \text{if } u_j^l \geq 0.5, \\ -\kappa & \text{if } u_j^l < 0.5 \end{cases} \quad \forall i = 1,\ldots n. \tag{65}$$

Finally, DCA applied to (59) is described as follows:

Algorithm 6 S3VM-Econ

initializations: let τ be a tolerance sufficiently small, set $l = 0$.
Choose $(w^0,b^0,\xi^0,r^0,s^0,u^0) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R}^p \times \mathbb{R}^n$.
repeat
1. Set $(\overline{w}^l,\overline{b}^l,\overline{\xi}^l,\overline{r}^l,\overline{s}^l,\overline{u}^l) = (0,0,0,\overline{r}^l,\overline{s}^l,\overline{u}^l)$ following (65), (20) and (21).
2. Solve the linear program (51) to obtain $(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1},u^{l+1})$.
3. $l = l + 1$.

until $\|(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1},u^{l+1}) - (w^l,b^l,\xi^l,r^l,s^l,u^l)\| \leq \tau(\|(w^l,b^l,\xi^l,r^l,s^l,u^l)\| + 1)$.

Theorem 2 *(Convergence properties of S3VM-Econ)*

(i) S3VM-Econ generates a sequence $\{(w^l,b^l,\xi^l,r^l,s^l,u^l)\}$ contained in $V(L)$ such that the sequence $\{F_6(w^l,b^l,\xi^l,r^l,s^l,u^l) + \kappa p(u^l)\}$ is decreasing.

(ii) For a number κ sufficiently large, if at an iteration q we have $u^q \in \{0,1\}^n$, then $u^l \in \{0,1\}^n$ for all $l \geq q$.

(iii) The sequence $\{(w^l,b^l,\xi^l,r^l,s^l,u^l)\}$ converges to $\{(w^*,b^*,\xi^*,r^*,s^*,u^*)\} \in V(L)$ after a finite number of iterations. The point $(w^*,b^*,\xi^*,r^*,s^*,u^*)$ is a critical point of Problem (59). Moreover if $u_i^* \neq \frac{1}{2}, \forall i = 1...n$ and $r_i^* \neq s_i^*, \forall i = 1..p$ then $\{(x^*,\gamma^*,\xi^*,\zeta^*,u^*)\}$ is a local solution to (59).

Proof. i) is consequence of DCA's convergence Theorem for a general DC program.

ii) Let $\kappa > \kappa_1 := \max\left\{\frac{F_6(w,b,\xi,r,s,u)-\eta}{\delta} : (w,b,\xi,r,s,u) \in V(L), p(u) \leq 0\right\}$ where $\eta := \min\{F_6(w,b,\xi,r,s,u) : (w,b,\xi,r,s,u) \in V(L)\}$ and $\delta := \min\{p(u) : (w,b,\xi,r,s,u) \in V(L)\}$. Let $\{(w^l,b^l,\xi^l,r^l,s^l,u^l)\} \subset V(L)$ $(l \geq 1)$ be generated by *S3VM-Econ*. If $V(L) \subset \{L \cap u \in \{0,1\}^n\}$, then the assertion is trivial. Otherwise, let $(w^l,b^l,\xi^l,r^l,s^l,u^l) \in \{L \cap u \in \{0,1\}^n\}$ and $(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1},u^{l+1}) \in V(L)$ be an optimal solution of the linear program (64). Then from (i) of this theorem we have

$$F_6(w^{l+1},b^{l+1},\xi^{l+1},r^{l+1},s^{l+1},u^{l+1}) + \kappa p(u^{l+1}) \leq F_6(w^l,b^l,\xi^l,r^l,s^l,u^l) + \kappa p(u^l).$$

Since $p(u^l) = 0$, it follows

$$\begin{aligned}\kappa p(u^{l+1}) &\leq F_6(w^l, b^l, \xi^l, r^l, s^l, u^l) - F_6(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1}, u^{l+1}) \\ &\leq F_6(w^l, b^l, \xi^l, r^l, s^l, u^l) - \eta.\end{aligned}$$

If $p(u^{l+1}) > 0$, then

$$\kappa \leq \frac{F_6(w^l, b^l, \xi^l, r^l, s^l, u^l) - \eta}{p(u^{l+1})} \leq \frac{F_6(w^l, b^l, \xi^l, r^l, s^l, u^l) - \eta}{\delta} \leq \kappa_1$$

which contradicts the fact that $\kappa > \kappa_1$. Therefore we have $p(u^{l+1}) = 0$.

iii) Since (59) is a polyhedral DC program, *S3VM-Econ* has a finite convergence and say, the sequence $\{(w^l, b^l, \xi^l, r^l, s^l, u^l)\}$ converges to a critical point $(w^*, b^*, \xi^*, r^*, s^*, u^*) \in V(L)$ after a finite number of iterations.

Furthermore, if $u_i^* \neq \frac{1}{2}, \forall i = 1...n$ and $r_i^* \neq s_i^*, \forall i = 1..p$ then the function H_6 is differentiable at $(w^*, b^*, \xi^*, r^*, s^*, u^*)$. Then using the convergence property of DC polyhedral program, $(w^*, b^*, \xi^*, r^*, s^*, u^*)$ is a local minimizer of (59). The proof is then complete. ■

4 Computational experiments

4.1 Datasets

Numerical experiments were performed on several real-world datasets taken from UCI Machine Learning Repository and NIPS 2003 Feature Selection Challenge. The information about data sets is summarized in Table 1 (#Att is the number of features while #Train(resp. #Test) stands for the number of points in training set (resp. test set)). For GIS, LEU and ARC datasets, training and test sets are given. For the remaining datasets, training and test sets are randomly sampled from the original set (cf. 4.3).

Table 1 Datasets

Dataset	#Att	#Train	#Test	#Total
W60	30	-	-	569
Ionosphere (INO)	34	-	-	351
Spambase (SPA)	57	-	-	2301
Internet Advertisements (ADV)	1558	-	-	3279
Gisette (GIS)	5000	6000	1000	7000
Leukemia (LEU)	7129	38	34	72
Arcene (ARC)	10000	100	100	200

4.2 Comparative algorithms

We compare our DCA based algorithms with the feature selection S3VM using ℓ_1-norm, namely

$$\min\{F_7(w, b, \xi, r, s) := \alpha\langle e, \xi\rangle + \beta\langle e, \min\{r, s\}\rangle + \|w\|_1 : (w, b, \xi, r, s) \in K\}. \tag{66}$$

Once again, problem (66) can be formulated as a DC program and then solved by DC programming and DCA. Let

$G_7(w, b, \xi, r, s) = \alpha\langle e, \xi\rangle + \|w\|_1$ and $H_7(w, b, \xi, r, s) = -\beta\langle e, \min\{r, s\}\rangle$.
The objective function of (66) can be written as:

$$F_7(w, b, \xi, r, s) = G_7(w, b, \xi, r, s) - H_7(w, b, \xi, r, s).$$

Obviously, $G_7(w, b, \xi, r, s)$ and $H_7(w, b, \xi, r, s)$ are convex polyhedral functions. Therefore (66) is a polyhedral DC program. DCA applied to (66) can be described as follows:

Algorithm 7 S3VM-ℓ_1

initializations: let τ be a tolerance sufficiently small, set $l = 0$.
Choose $(w^0, b^0, \xi^0, r^0, s^0) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R}^p$.
repeat
1. Compute $(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l) = (0, 0, 0, \bar{r}^l, \bar{s}^l)$ following (20) and (21).
2. Solve the linear following program to obtain $(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1})$:

$$\begin{cases} \min\ \langle e, t\rangle + \alpha\langle e, \xi\rangle - \langle(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l), (w, b, \xi, r, s)\rangle, \\ s.t.\ \ (w, b, \xi, r, s) \in K, \\ \qquad t_i \geq w_i,\ t_i \geq -w_i \quad \forall i = 1, \ldots, n. \end{cases} \tag{67}$$

3. $l = l + 1$.
until $\|(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1}) - (w^l, b^l, \xi^l, r^l, s^l)\| \leq \tau(\|(w^l, b^l, \xi^l, r^l, s^l)\| + 1)$.

As we have mentioned before, Bradley and Mangasarian ([4]) proposed a concave approximation of ℓ_0-norm. Later, in [43], the author used the same approximation for S3VM. As for Logarithm approximation, by introducing a non-negative variable $u \geq 0$ and the constraint relation $-u \leq w \leq u$, the resulting problem is described as follows

$$\min \left\{ F_8(w, b, \xi, r, s, u) := \alpha\langle e, \xi\rangle + \beta\langle e, \min\{r, s\}\rangle + \sum_{i=1}^{n}(1 - \varepsilon^{-\lambda u_i}) \right\} : (w, b, \xi, r, s, u) \in \bar{K}. \tag{68}$$

with $\bar{K}$ defined in (48). The objective function of (68) can be written as:

$$F_8(w, b, \xi, r, s, u) = G_8(w, b, \xi, r, s, u) - H_8(w, b, \xi, r, s, u)$$

where

$G_8(w, b, \xi, r, s, u) = \chi_{\bar{K}}$ and $H_8(w, b, \xi, r, s, u) = -F_8(w, b, \xi, r, s, u)$.

It is clear that $G_8(w, b, \xi, r, s, u)$ and $H_8(w, b, \xi, r, s, u)$ are convex functions. Therefore (68) is a DC program. Below is the description of DCA applied to (68):

Algorithm 8 S3VM-FSV

initializations: let τ be a tolerance sufficiently small, set $l = 0$.
Choose $(w^0, b^0, \xi^0, r^0, s^0, u^0, u^0) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R}^p \times \mathbb{R}^n$.
repeat
1. Compute $(\bar{w}, \bar{b}, \bar{\xi}, \bar{r}, \bar{s}, \bar{u}) = (0, 0, \bar{\xi}, \bar{r}, \bar{s}, \bar{u})$ with $\bar{r}$ (resp. $\bar{s}$) given as in (20) (resp. (21)) and

$$\begin{aligned} &\bar{\xi} = -\alpha e, \\ &\bar{u}_i = \lambda\epsilon^{-\lambda u_i} \quad \forall i = 1, \ldots, n. \end{aligned}$$

2. Solve the linear following program to obtain $(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1}, u^{l+1})$:

$$\min\left\{ -\langle(\bar{w}^l, \bar{b}^l, \bar{\xi}^l, \bar{r}^l, \bar{s}^l, \bar{u}^l), (w, b, \xi, r, s, u)\rangle : (w, b, \xi, r, s, u) \in \bar{K} \right\}. \tag{69}$$

3. $l = l + 1$.
until $\|(w^{l+1}, b^{l+1}, \xi^{l+1}, r^{l+1}, s^{l+1}, u^{l+1}, u^{l+1}) - (w^l, b^l, \xi^l, r^l, s^l, u^l, u^l)\| \leq \tau(\|(w^l, b^l, \xi^l, r^l, s^l, u^l, u^{l+1})\| + 1)$.

4.3 Set up experiments and Parameters

All algorithms were implemented in the Visual C++ 2005, and performed on a PC Intel i5 CPU650, 3.2 GHz of 4GB RAM. We stop DCA with the tolerance $\tau = 10^{-6}$. The non-zero elements of w are determined according to whether $|w_i|$ exceeds a small threshold (10^{-6}). For the W60, INO, SPA and ADV datasets, we randomly chose 60% of the whole original dataset for training, the 40% remaining data are used as test set. This procedure is repeated 10 times. On each given training set and test set, each algorithm is performed 10 times from 10 random starting points. We then report the best result, the average result and the standard deviation

over the executions. The starting point of DCA is chosen as follows: ξ^0, r^0, s^0 are set to zero and w^0 and b^0 are randomly chosen.

For S3VM_PiE, we set $\lambda = 5$ as proposed by [4]. Following Fan and Li in [12] we set the parameters λ and γ in S3VM_SCAD to, respectively, 0.4 and 3.4. For S3VM_PoDC, λ is chosen from 0.9 to 2.1. For S3VM_PiL, we take $a = 10^{-6}$ and $b \in \{10^{-4}, 10^{-3}, ..., 10^{-1}, 0.2, 0.3\}$. In S3VM_ECon, the penalty parameter κ is in the set $\{100, 1000, 2000, 3000, 4000, 5000\}$, c_i is set to a sufficient large value, say $10,000$. Finally, for S3VM_Log, ε is set to 0.1.

We are interested in the classification error and the sparsity of obtained solution as well as the rapidity of the algorithms. We measure the classification error via two criteria : the maximum sum (MS) and the accuracy (ACC) which are defined as follows:

$$MS = (SE + SP)/2, \quad ACC = (TP + TN)/(TP + TN + FP + FN),$$

where TP and TN denote true positives and true negatives while FP and FN represent false positives and false negatives. $SE = TP/(TP + FN)$ (resp. $SP = TN/(TN + FP)$) is the correctness rate of positive class (resp. negative class). The sparsity of solution is determined by the number of selected features (SF) while the rapidity of algorithms is measured by the CPU time in seconds.

4.4 Experimental results and comments

Experiment 1: In the first experiment, we are interested in the effectiveness of all algorithms when the numbers of unlabeled points varies. For this purpose, we arbitrarily choose a data set (**LEU**) and then change the percentage of unlabeled points in training set from 20% to 80%. We report in Figure 2 (resp. Figure 1), the ACC (resp. SF) of all algorithms on training and test set.

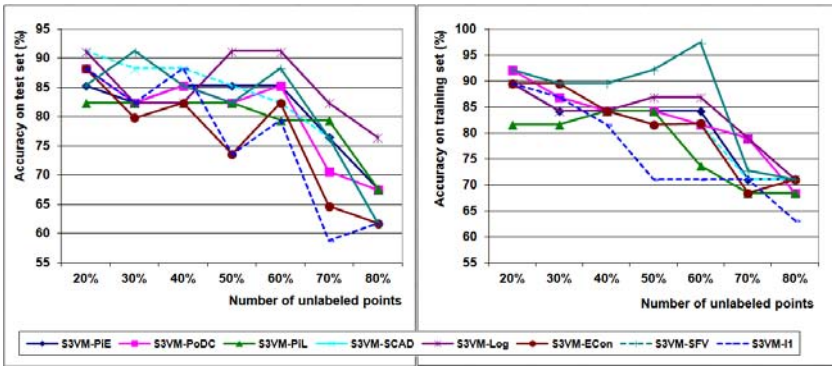


Fig. 1 Accuracy of classifiers (ACC) on training (right) and test (left) of dataset **LEU** with different numbers of unlabeled points.

We observe that:

- In most cases, the DCA based algorithms on the ℓ_0 model give better accuracy while choosing much less features than S3VM-ℓ_1 (the gain on number of selected features is up to $5,6$ times).
- When the number of unlabeled points exceeds 70%, the accuracy of all algorithms decrease dramatically, especially for S3VM-ℓ_1.

Experiment 2: In the second experiment, we compare the effectiveness of all the algorithms with a fixed percentage of unlabeled points . According to the second remark of experiment 1, 60% of training set will be set to be unlabeled points. The comparative results are reported in Table 2, Table 3, Table 4, Table 5.
From the computational results we observe that:

Table 2 Selected features (and corresponding percentages) of the algorithms

Data	S3VM-PiE		S3VM-PoDC		S3VM-PiL		S3VM-SCAD		S3VM-Log		S3VM-ECon		S3VM-SFV		S3VM-l_1	
Name	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD
W60 (#)	**2**	2.92±0.63	4	4.12±0.33	5	5.24±0.43	**2**	**2.44**±0.49	3	3.40±0.80	3	4.27±1.10	4	4.28±0.46	6	6.61±0.64
(%)	**6.67**	9.73±2.10	13.33	13.73±1.10	16.67	17.47±1.43	**6.67**	**8.13**±1.63	10.00	11.33±2.67	10.00	14.23±3.67	13.33	14.27±1.53	20.00	22.03±2.13
INO (#)	**4**	**5.5**±0.68	5	8.21±1.81	7	7.31±0.35	5	6.01±0.55	5	8.11±2.89	6	7.02±0.44	6	6.64±0.89	28	29.10±0.65
(%)	**11.76**	**16.18**±2.00	14.71	24.15±5.32	20.59	21.50±1.03	14.71	17.68±1.62	14.71	23.85±8.50	17.65	20.65 ±1.29	17.65	19.53±2.62	82.35	85.59±1.91
SPA (#)	4	9.69±3.54	**3**	6.54±2.51	5	12.35±4.46	5	9.65±4.10	4	6.41±1.03	5	12.31±3.52	**3**	**6.26**±1.71	15	23.71±5.24
(%)	7.02	17.00±6.21	**5.26**	11.47±4.40	8.77	21.67±7.82	8.77	16.93±7.19	7.02	11.25±1.81	8.77	21.60 ±6.18	**5.26**	**10.98**±3.00	26.32	41.60±9.19
ADV (#)	**2**	**3.97**±1.60	14	30.93±13.84	24	45.60±14.32	15	25.28±6.62	3	4.32±0.61	3	27.24±14.99	32	38.17±6.87	265	489.45±109.66
(%)	**0.13**	**0.25**±0.10	0.90	1.99±0.89	1.54	2.93±0.92	0.96	1.62±0.42	0.19	0.28±0.04	0.19	1.75±0.96	2.05	2.45±0.44	17.01	31.42±7.04
GIS (#)	1082	1112.8±27.1	1344	1394.4±34.7	**525**	**808.3**±279.8	1334	1389.0±37.7	1122	1144.8±14.6	**525**	**808.3**±279.8	1135	1237.6±51.26	1918	2073.5±122.6
(%)	21.64	22.26±0.54	26.88	27.89±0.69	**10.5**	**16.17**±5.59	26.68	27.78±0.75	22.44	22.90±0.29	**10.5**	**16.17**±5.59	22.70	24.75±1.03	38.36	41.47±2.45
LEU (#)	8	14.40±2.87	11	16.10±2.77	10	12.60±3.01	11	17.50±3.58	**3**	**4.50**±1.28	9	13.20±2.44	15	20.40±12.93	11	18.40±5.46
(%)	0.11	0.20±0.04	0.15	0.23±0.04	0.14	0.18±0.04	0.15	0.25±0.05	**0.04**	**0.06**±0.02	0.13	0.19±0.03	0.21	0.29±0.18	0.15	0.26±0.08
ARC (#)	**10**	**10.30**±0.45	14	21.80±6.21	18	42.10±15.23	46	62.9±12.66	15	15.70±0.78	13	20.20±4.40	20	29.60±16.11	51	71.20±13.41
(%)	**0.10**	**0.10**±0.00	0.14	0.22±0.06	0.18	0.42±0.15	0.46	0.63±0.13	0.15	0.16±0.01	0.13	0.20±0.04	0.20	0.30±0.16	0.51	0.71±0.13
Average(%)	6.78	**9.39**±1.57	8.77	11.38±1.79	8.34	11.48±2.43	8.34	10.43±1.68	7.79	9.98±1.91	**6.77**	10.68±2.54	8.77	10.37±1.28	26.39	31.87±3.28

Table 3 Accuracy of classifiers of the algorithms on: (1) - test set, (2) - training set

Data	S3VM-PiE		S3VM-PoDC		S3VM-PiL		S3VM-SCAD		S3VM-Log		S3VM-ECon		S3VM-SFV		S3VM-l_1	
Name	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD
W60 (1)	**94.69**	90.94±2.61	93.36	90.11±3.48	**94.69**	**93.66**±0.87	92.06	91.88±0.36	92.92	90.09±3.09	92.92	90.53±0.77	89.42	84.78±2.68	84.13	84.01±0.13
(2)	93.29	91.35±1.32	**94.46**	**92.40**±3.36	**94.46**	90.41±3.31	92.90	91.93±1.94	91.25	89.91±2.03	93.88	91.65±1.09	86.05	83.42±2.96	85.66	83.66±1.17
INO (1)	**85.05**	76.31±2.21	84.25	74.27±1.67	78.81	75.96±1.36	75.32	74.66±0.45	77.78	**76.68**±1.10	76.92	75.87±1.47	77.78	75.47±2.84	72.89	71.69±0.33
(2)	78.52	73.89±2.44	84.77	74.98±7.49	82.24	80.88±0.71	80.55	80.04±0.09	85.90	82.56±2.36	82.24	80.86±0.41	81.2	77.66±3.18	**88.46**	**88.26**±0.41
SPA (1)	**65.36**	**63.56**±0.42	63.72	63.38±0.28	63.72	62.88±0.49	63.88	62.87±0.38	63.72	62.87±0.33	63.78	62.89±0.44	63.89	63.02±0.72	63.55	62.11±0.52
(2)	**65.68**	**63.17**±0.44	63.36	62.89±0.41	64.49	62.36±0.75	63.97	62.99±0.52	63.97	62.98±0.46	64.41	62.55±0.66	64.7	62.56±0.53	64.56	62.74±0.64
ADV (1)	95.50	93.17±1.78	95.04	**93.81**±0.83	95.04	93.48±3.23	94.36	92.44±1.23	95.04	93.61±1.15	**95.88**	93.20±1.96	92.81	91.77±0.81	93.88	92.14±0.64
(2)	95.63	93.60±1.79	**96.44**	**94.94**±1.02	95.63	94.18±0.98	95.22	93.22±1.13	95.22	93.89±0.98	96.24	93.96±1.66	95.04	93.21±0.88	95.32	94.08±1.02
GIS (1)	68.40	67.34±0.85	70.10	68.24±1.27	68.60	66.48±1.14	**73.09**	69.32±1.80	70.70	**70.00**±0.40	68.60	66.48±1.14	69.90	69.07±0.83	65.10	62.94±1.58
(2)	69.65	68.98±0.53	69.08	67.94±0.94	67.87	65.61±1.12	69.07	68.18±0.88	**70.9**	**69.73**±0.60	67.87	65.61±1.12	69.38	69.12±0.28	64.60	63.49±0.53
LEU (1)	85.29	74.41±4.56	85.29	75.88±4.88	79.41	72.06±3.54	82.35	76.77±2.77	**91.18**	**84.41**±3.49	82.35	67.65±5.09	88.24	76.47±4.92	79.41	71.18±4.32
(2)	84.21	67.89±9.83	81.58	73.95±5.94	73.68	61.84±6.89	81.79	71.84±7.26	**94.97**	**85.53**±5.16	73.68	66.05±4.15	97.37	80.00±7.82	71.06	63.68±6.64
ARC (1)	75.00	75.00±0.00	76.00	72.50±2.25	71.00	66.80±2.23	68.00	64.00±1.61	**79.00**	**75.30**±1.68	71.00	67.50±1.50	72.00	70.80±0.75	75.00	66.40±3.20
(2)	76.00	75.30±0.90	69.00	66.67±1.73	73.00	61.80±5.85	78.00	69.90±4.44	**81.00**	**78.90**±1.04	73.00	68.00±3.82	81.00	75.80±3.34	71.00	62.80±4.19
Average(1)	78.97	77.25±1.87	81.11	76.88±2.09	78.75	75.90±1.84	78.44	75.99±1.23	**81.48**	**78.99**±1.61	78.78	74.87±1.77	79.15	75.92±1.94	76.28	72.92±1.53
(2)	80.43	76.31±2.46	79.81	76.25±2.98	78.77	73.87±2.80	80.21	76.87±2.32	**83.32**	**80.50**±1.80	78.76	75.53±1.84	82.11	77.40±2.71	77.24	74.10±2.09

Table 4 Maximum Sum of the algorithms on: (1) - test set, (2) - training set

Data		S3VM-PiE		S3VM-PoDC		S3VM-PiL		S3VM-SCAD		S3VM-Log		S3VM-ECon		S3VM-SFV		S3VM-l_1	
Name		Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD	Best	Mean±SD
W60	(1)	**94.47**	90.97±2.31	91.96	91.49±0.28	94.08	**93.66**±1.22	91.74	91.44±0.61	92.47	91.24±2.32	93.15	91.47± 0.84	91.6	87.23±1.68	86.81	86.63±0.47
	(2)	94.02	91.49±1.90	94.95	92.06±2.12	94.02	**92.77**±2.71	89.52	89.37± 0.18	91.07	90.29±1.03	**94.49**	92.67±0.96	88.72	85.35±1.16	85.57	85.21±0.33
INO	(1)	**82.32**	75.01±2.69	78.89	73.71±1.88	77.54	72.91±0.35	73.33	72.66±0.14	82.14	**75.11**±2.33	75.55	72.85±0.31	76.62	74.21±2.01	69.68	69.18±0.19
	(2)	78.36	76.39±2.18	**85.67**	76.17±5.82	81.54	81.11±0.14	81.87	81.55±0.64	85.66	82.72±3.12	82.24	81.22±0.14	83.5	80.36±2.57	83.94	**83.02**±0.32
SPA	(1)	54.69	53.88±0.45	**54.88**	53.78±0.33	54.65	53.18±0.42	53.88	53.24±0.35	54.14	**53.91**±0.37	54.55	53.84±0.69	54.39	**53.91**±0.48	54.12	52.99±0.68
	(2)	53.89	53.12±0.49	54.52	**53.57**±0.38	54.88	53.56±1.17	53.98	52.19±0.71	53.89	52.87±0.36	54.96	53.11± 1.01	55.15	52.55±1.14	**54.98**	52.81±0.82
ADV	(1)	92.13	90.41±1.68	**92.52**	89.35±1.74	**92.52**	**91.31**±0.77	89.62	87.42 ±1.12	89.06	83.00±4.46	92.40	85.45±7.13	88.91	87.19±0.85	89.06	86.69±2.89
	(2)	**94.65**	92.24±0.89	94.17	92.17±1.36	94.24	93.66±0.51	93.04	89.96±1.67	88.49	83.64±3.70	94.18	87.50±7.38	94.44	93.12±0.64	94.36	**93.84**±0.38
GIS	(1)	68.40	67.34±0.85	70.10	68.24±1.27	68.60	66.48±1.14	**74.95**	69.67±2.39	70.70	**70.00**±0.40	68.60	66.48±1.14	69.90	69.07±0.83	65.10	62.94±1.58
	(2)	69.65	68.98±0.53	69.08	67.94±0.94	67.87	65.61±1.12	69.07	68.18±0.88	**70.9**	**69.73**±0.60	67.87	65.61±1.12	69.38	69.12±0.28	64.60	63.49±0.53
LEU	(1)	83.21	70.86±5.82	84.29	73.29±5.81	80.36	72.18±4.09	78.57	74.46±2.99	**90.36**	**81.29**±4.66	81.79	69.93±4.26	85.71	71.86±5.81	80.36	68.32±5.53
	(2)	86.20	65.56±11.59	75.25	69.82±6.35	66.84	58.60±6.07	87.04	69.68±8.61	90.91	**75.00**±8.92	78.79	67.22±5.57	**90.16**	66.12±12.49	68.86	58.01±7.75
ARC	(1)	74.51	**74.51**±0.00	74.68	72.16±1.61	70.94	66.17±3.48	67.78	64.79±1.16	**77.11**	73.81±1.50	70.7	66.62±1.54	71.10	69.03±1.55	74.76	66.32±3.31
	(2)	75.41	74.78±0.80	70.13	66.86±2.18	73.21	61.14±7.12	78.9	71.06±4.71	**80.36**	**78.26**±1.08	72.48	68.00±3.80	78.90	74.47±2.43	72.89	63.13±4.17
Average	(1)	78.53	74.71±1.97	78.19	74.57±1.85	76.96	73.70±1.64	75.70	73.38±1.25	**79.43**	**75.48**±2.29	76.68	72.38±2.70	76.89	73.21±1.89	74.27	70.44±2.09
	(2)	78.88	74.65±2.63	77.68	74.08±2.74	76.09	72.35±2.69	79.06	74.62±2.49	**80.18**	**76.07**±2.69	77.86	73.62±2.85	80.04	74.44±2.96	75.03	71.36±2.04

Table 5 Comparisons of the algorithms in term of CPU time

Data	S3VM-PiE	S3VM-PoDC	S3VM-PiL	S3VM-SCAD	S3VM-Log	S3VM-ECon	S3VM-SFV	S3VM-l_1
W60	0.133±0.020	0.199±0.009	0.181±0.009	0.229±0.008	0.174±0.006	**0.087**±0.011	0.255±0.006	0.149±0.017
INO	0.125±0.103	0.181±0.011	**0.101**±0.003	0.157±0.08	0.139±0.066	0.113±0.016	0.219±0.004	0.169±0.03
SPA	**0.258**±0.150	0.445±0.055	0.484±0.054	0.437±0.06	0.451±0.053	0.499±0.054	0.468±0.089	0.491±0.026
ADV	7.411±1.29	7.247±0.084	7.57±0.019	7.028 ±0.092	**5.105**±0.146	6.878±0.169	6.785±0.177	7.695±0.207
GIS	**551.55**±41.98	630.92±169.79	819.66±431.71	566.34±10.41	598.32±18.17	819.65±431.70	599.87±49.73	553.77±6.94
LEU	6.01±0.26	5.980±0.455	6.253±1.047	**5.788**±1.00	5.937±0.225	6.348±1.569	6.059±0.144	6.265±0.227
ARC	**11.68**±0.34	13.254±0.816	18.64±2.050	13.228±1.19	12.157±0.548	21.739±4.677	11.022±0.602	14.439±0.596
Average	**82.45**±6.31	94.03±24.46	121.84±62.13	84.74±1.83	88.89±2.75	122.19±62.60	89.24±7.25	83.28±1.15

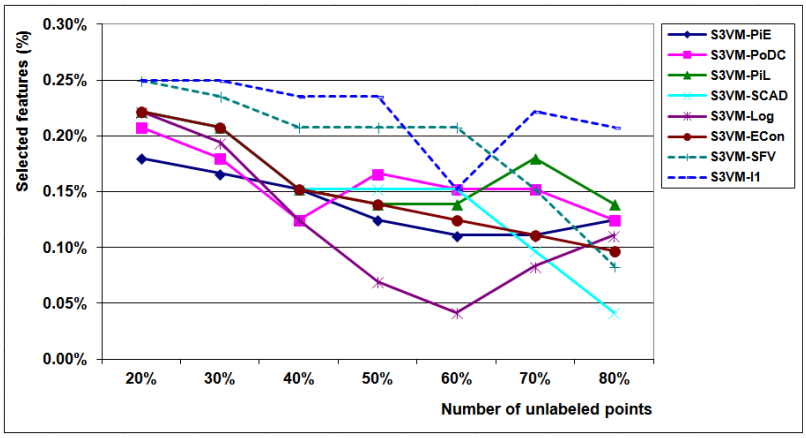


Fig. 2 Percentage of selected features on dataset **LEU** with different numbers of unlabeled points.

- All DCA based algorithms on ℓ_0 model reduce considerably the number of features (from 72.11% to 99.94%) while the accuracy of classifier is quite good (from 61.80% to 94.94%). In comparison with S3VM-ℓ_1, naturally the DCA based algorithms on ℓ_0 model suppress much more features while they always furnish better accuracy (MS/ACC). The gain of percentage of selected features (SF) with respect to S3VM_ℓ_1 is up to 123.29 times (S3VM_PiE comparing to S3VM_ℓ_1 on dataset ADV). Averagely on all datasets, S3VM_PiE is the best on term of selected features. The number of selected features of S3VM_ℓ_1 is higher than that of S3VM_PiE (resp. S3VM_PoDC, S3VM_PiL, S3VM_SCAD, S3VM_Log and S3VM_ECon) 3.39 (resp. $2.80, 2.78, 3.06, 3.19$ and 2.98) times.
- In term of ACC/MS, the quality of all five DCA based algorithms on ℓ_0 model are comparable. S3VM_Log is better than other on 5 out of 7 datasets which can be explained by the fact that S3VM_Log selects slightly more features than other algorithms in these datasets.
- Concerning the computation time, the CPU time of all DCA based algorithms is quite small: less than 22 seconds (except for dataset **GIS**).

5 Conclusion

We have intensively investigated a unified DC programming approach for feature selection in the context of Semi-Supervised Support Vector Machine. Using 5 different approximations and the continuous exact reformulation via an exact penalty technique leads us to 6 DC programs. Then we developed 6 based DCA algorithms for solving the resulting problems. Numerical results on several real datasets showed the robustness, the effectiveness of the DCA based schemes. We are convinced that DCA is a promising approach for the combined feature selection and S3VMs applications.

References

1. E. Amaldi and V. Kann. On the approximability of minimizing non zero variables or unsatisfied relations in linear systems, Theoretical Computer Science. Vol. 209:237–260 (1998).
2. M. Belkin, P. Niyogi, Semi-supervised learning on riemannian manifolds. Machine Learning, 56, 209–239 (2004).
3. Kristin P. Bennett and Ayhan Demiriz, Semi-supervised support vector machines, In Proceedings of the conference on Advances in neural information processing systems II, Cambridge, MA, USA, 368–374 (1999).
4. P. S. Bradley and O. L. Mangasarian, Feature Selection via concave minimization and support vector machines, Proceeding of ICML'98, SF, CA, USA, 82–90 (1998).
5. T.D. Bie and N. Cristianini. Convex methods for transduction. In Adv. in Neural Information Proc. Systems 16, p. 73- MIT Press (2004).

6. O. Chapelle, A. Zien A, Semi-supervised classification by low density separation. In Proc. 10th Internat. Workshop on Artificial Intelligence and Statistics, Barbados, 57–64 (2005).
7. O. Chapelle, V. Sindhwani and S. Keerthi, Branch and bound for semi-supervised support vector machines. In Advances in Neural Information Processing Systems, MIT Press, Cambridge, 17, 217–224 (2006).
8. O. Chapelle, V. Sindhwani and S. S. Keerthi Optimization Techniques for Semi-Supervised Support Vector Machines, Journal of Machine Learning Research, Vol. 9:203–233 (2008).
9. H. Choi, J. Kim, Y. Kim, A sparse large margin semi-supervised learning method, Journal of the Korean Statistical Society, Vol. 39(4), pp. 479- (2010).
10. R. Collobert, F. Sinz, J. Weston, L. Bottou, Large scale transductive SVMs. J. Machine Learn, Vol. 7:1687–1712 (2006).
11. W. Emara, M.K.M. Karnstedt, K. Sattler, D. Habich and W. Lehner, An Approach for Incremental Semi-supervised SVM, Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007), pp. 539–544 (2007).
12. J. Fan, R. Li, (2001) Variable selection via nonconcave penalized likelihood and its Oracle Properties. *Journal of the American Statistical Association 96*, pp. 1348-1360.
13. G. Fung and O. Mangasarian, Semi-supervised support vector machines for unlabeled data classification, Optimization Methods and Software, Vol.15:29-(2001).
14. F. Gieseke, A. Airola, T. Pahikkala and O. Kramer, Sparse Quasi-Newton Optimization for Semi-Supervised Support Vector Machines, Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods (ICPRAM), pp. 45–54 (2012).
15. T. Joachims, Transductive inference for text classification using support vector machines. In 16th Inter. Conf. on Machine Learning ,SF, USA, 200–209 (1999).
16. Hermes, L., Buhmann, J.M.: Feature selection for support vector machines. Proceedings. 15th International Conference on Pattern Recognition, vol.2, pp. 712–715.
17. Hui, Z.: The Adaptive Lasso and Its oracle Properties. Journal of the American Statistical Association 101:476, pp. 1418–1429 (2006).
18. N. Krause and Y. Singer, Leveraging the margin more carefully, Proceeding of ICML '04, NY, USA, 63–71 (2004).
19. Y.F. Li , J.T. Kwok and Z.H. Zhou, Semi-Supervised Learning Using Label Mean, in Proceedings of the 26th International Conference on Machine Learning, Montreal, Canada, pp. 1–8 (2009).
20. H.A. Le Thi, Contribution à l'optimisation non convexe et l'optimisation globale: Théorie, Algoritmes et Applications, Habilitation à Diriger des Recherches, Université de Rouen (1997).
21. H.A. Le Thi and T. Pham Dinh, Solving a class of linearly constrained indefinite quadratic problems by DC algorithms, Journal of Global Optimization, Vol. 11(3):253–285 (1997).
22. H.A. Le Thi and T. Pham Dinh, The DC (difference of convex functions) Programming and DCA revisited with DC models of real world nonconvex optimization problems, Annals of Operations Research, Vol. 133:23–46 (2005).
23. H.A. Le Thi, T. Belghiti and T. Pham Dinh, A new efficient algorithm based on DC programming and DCA for Clustering, Journal of Global Optimization, Vol. 37:593–608 (2006).
24. H.A.Le Thi , M. Le Hoai and T. Pham Dinh, Optimization based DC programming and DCA for Hierarchical Clustering, European Journal of Operational Research, Vol. 183:1067–1085 (2007).
25. H.A. Le Thi , M. Le Hoai, N.V. Nguyen and T. Pham Dinh, A DC Programming approach for Feature Selection in Support Vector Machines learning, Journal of Advances in Data Analysis and Classification, Vol 2(3):259–278 (2008).
26. H.A Le Thi, V.V Nguyen, S. Ouchani, Gene Selection for Cancer Classification Using DCA, Journal of Fonctiers of Computer Science and Technology, Vol.3 No.6 Sum No.15 (2009).
27. Le Thi Hoai An, Huynh Van Ngai and Pham Dinh Tao, Exact Penalty and Error Bounds in DC Programming, Journal of Global Optimization dedicated to Reiner Horst ISSN 0925-5001, DOI: 10.1007/s10898-011-9765-3.
28. H.A. Le Thi . A new approximation for the ℓ_0-norm. Research Report LITA EA 3097, University of Lorraine, France (2012).
29. H.A. Le Thi , M. Le Hoai and T. Pham Dinh, Feature Selection in machine learning: an exact penalty approach using a Different of Convex function Algorithm, submitted (2013).
30. H.A. Le Thi, DC Programming and DCA. http://lita.sciences.univ-metz.fr/∼lethi.
31. Y. Liu, X. Shen and H. Doss, Multicategory ψ-Learning and Support Vector Machine: Computational Tools, Journal of Computational and Graphical Statistics, Vol. 14:219–236 (2005).
32. J. Ma and X. Zhang, A full smooth semi-support vector machine based on the cubic spline function, 6th International Conference on Biomedical Engineering and Informatics (BMEI), pp. 650–655 (2013).
33. J. Neumann , C. Schnörr and G. Steidl, Combined SVM-based feature selection and classification, Machine Learning, Vol. 61(1–3):129–150 (2005).
34. K. Nigam, A. McCallum, S. Thrun and T. Mitchell, Text classification from labeled and unlabeled documents using EM. Machine Learning, 39, 103–134 (2000).
35. T. Pham Dinh and H.A. Le Thi, Convex analysis approach to d.c. programming: Theory, Algorithm and Applications. Acta Mathematica Vietnamica 22, pp. 289–355 (1997).
36. T. Pham Dinh, H.A. Le Thi, Recent advances on DC programming and DCA. Transactions on Computational Intelligence XIII, Lecture Notes in Computer Science Volume 8342, 2014, pp 1-37.
37. D. Peleg and R. Meir, A bilinear formulation for vector sparsity optimization. Signal Processing, 8(2), 375–389 (2008).

38. C.S Ong and H.A. Le Thi, Learning sparse classifiers with Difference of Convex functions Algorithms, Optimization Methods and Software, Vol 28:4 (2013).
39. Qi, Y. Tian, Y. Shi and X. Yu, Cost-Sensitive Support Vector Machine for Semi-Supervised Learning, Procedia Computer Science, Volume 18, pp. 1684–1689 (2013).
40. M. Thiao, T. Pham Dinh and H.A. Le Thi, DC programming approach for a class of nonconvex programs involving l_0 norm, in "Modelling, Computation and Optimization in Information Systems and Management Sciences", Communications in Computer and Information Science CCIS Volume 14, Springer, 358–367 (2008).
41. A. Rakotomamonjy, Variable Selection Using SVM-based Criteria, Journal of Machine Learning Research, Vol. 3:1357–1370 (2003).
42. C. Ronan, S. Fabian, W. Jason and B. Lé, Trading Convexity for Scalability, Proceedings of the 23rd international conference on Machine learning ICML 2006. Pittsburgh, Pennsylvania, 201–208 (2006).
43. L. Yang and L.Wang, Simultaneous Feature Selection and Classification via Semi-Supervised Models, Proceeding ICNC '07, Third International Conference on Natural Computation-Cover, 646–650 (2007).
44. V. Sindhwani, S. Keerthi, and O. Chapelle, Deterministic annealing for semi-supervised kernel machines, Proceedings of ICML'06, NY, USA, 841–848 (2006).
45. V. Sindhwani, S. Keerthi, Large scale semi-supervised linear SVMs, Proceeding SIGIR '06 Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, 477–484 (2006).
46. R. Tibshirani, Regression shrinkage and selection via the lasso. J. Roy. Stat. Soc., Vol. 46:, pp.431–439 (1996).
47. M. Thiao, T. Pham Dinh, and H.A. Le Thi A DC programming approach for Sparse Eigenvalue Problem, *Proceeding of ICML 2010*, 1063–1070 (2010)..
48. V. Vapnik and A. Sterin, On structural risk minimization or overall risk in a problem of pattern recognition. Automation and Remote Control, 10(3):1495-03 (1977).
49. R. Zhang, T.B. Liu and M.W. Zheng, Semi-Supervised Learning for Classification with Uncertainty, Advanced Materials Research, volumes 433 - 440, pp. 3584–3590 (2012).
50. H. Zou, The adaptive lasso and its oracle properties. *J. Amer. Stat. Ass.*, 101, 1418–1429 (2006).
51. X. Zhu and A.B. Goldberg, Introduction to Semi-Supervised Learning, Morgan and Claypool, ISBN:1598295470 9781598295474 (2009).
52. Wang, H., Li, G., Jiang, G.: Robust regression shrinkage and consistent variable selection via the LAD-LASSO. Journal of Business & Economics Statistics 25:3, pp. 347–355 (2007).
53. J. Weston and A. Elisseeff and B. Scholkopf and M. Tipping. Use of the Zero-Norm with Linear Models and Kernel Methods. Journal of Machine Learning Research. Vol.3 pp.1439-1461 (2003).

Chapter 5

DCA based Algorithms for Feature Selection in Multi-class Support Vector Machine

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

DCA based algorithms for Feature Selection in Multi-class Support Vector Machine

Hoai An LE THI · Manh Cuong NGUYEN

Submitted.
Received: date / Accepted: date

Abstract This paper addresses the problem of feature selection for Multi-class Support Vector Machines (MSVM). Two models involved the l_0 (the zero norm) and the l_2-l_0 regularizations are considered for which two continuous approaches based on DC (Difference of Convex functions) programming and DCA (DC Algorithms) are investigated. The first is DC approximation via several sparse inducing functions and the second is an exact reformulation approach using penalty techniques. Twelve versions of DCA based algorithms are developed on which empirical computational experiments are fully performed. Numerical results on real-world datasets show the efficiency and the superiority of our methods versus one of the best standard algorithms on booth feature selection and classification.

Keywords Feature selection, MSVM, DC programming, DCA, DC approximation, exact penalty.

1 Introduction

One of challenges of Machine Learning is the handling of the input datasets with very large number of features. Many techniques are proposed to address this challenge. The goals are to remove the irrelevant and redundant features, reduce store space and execution time, and avoid the course of dimensionality to improve the prediction performance [23].

Here, we are interested in the feature selection task for Multi-class Support Vector Machine (MSVM). The objective is to simultaneously select a subset of features (representative features) and construct a good classifier. Whereas most feature selection methods were initially developed for binary-Support Vector Machine (SVM) classification (see e.g. [1], [10], [12], [14], [22], [23], [32],[38], [42]), several extensions to feature selection for MSVM are recently investigated (see e.g. [2], [4], [5], [8], [9], [16], [13], [25] - [27], [43] - [49]). We first consider the model of MSVM proposed by Weston and Watkins [44], a direct approach (without using binary-SVM) for learning multiclass, known to be appropriate to capture correlations between the different classes, which can be described as follows.

Let $\mathcal{X}$ be a set of vectors in $\mathbb{R}^d$ and $\mathcal{Y} = \{1, ..., Q\}$ be a set of class labels. Given a training dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), .., (x_n, y_n)\} \in \mathbb{R}^{n\times(d+1)}$, where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}, i = \{1, ..., n\}$. The task is to learn a classification rule $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps an element x to a class label $y \in \mathcal{Y}$.

In a more natural way than the classical SVM based approaches for multi-classification, Weston and Watkins [44] proposed to construct a piecewise linear separation that gives the decision function:

Hoai An LE THI · Manh Cuong NGUYEN
Laboratory of Theoretical and Applied Computer Science LITA EA 3097
University of Lorraine, Ile du Saulcy, 57045 Metz, France.
E-mail: hoai-an.le-thi@univ-lorraine.fr, manh-cuong.nguyen@univ-lorraine.fr

$$f(x) = \arg\max_{1\leq i\leq Q} f_i(x), \tag{1}$$

where f_i stands for the hyperplane $f_i(x) = \langle w_i, x\rangle + b_i$, with $w_i \in \mathbb{R}^d, b_i \in \mathbb{R}$, $i = 1, ...Q$. Let $w = \left(w_1, w_2, ..., w_Q\right)$ be the vector in $\mathbb{R}^{Q\times d}$ and let $\boldsymbol{b} = (b_i)_{i=1}^Q \in \mathbb{R}^Q$. Then the MSVM model given in [44], the first "all-together" implementation of multi-class SVM, is a single optimization problem of the form:

$$\min\left\{C\sum_{i=1}^{n}\sum_{k\neq y_i}\xi_{ik} + \sum_{k=1}^{Q}\|w_k\|_2^2 : (w, b, \xi) \in \Omega\right\}, \tag{2}$$

where

$$\Omega = \left\{\begin{array}{l}(w, b, \xi) \in \mathbb{R}^{Q\times d}\times\mathbb{R}^Q\times\mathbb{R}_+^{n\times Q}: \\ \langle w_{y_i} - w_k, x_i\rangle + b_{y_i} - b_k \geq 1 - \xi_{ik}, \forall 1 \leq i \leq n, 1 \leq k \neq y_i \leq Q\end{array}\right\},$$

and $\xi \in \mathbb{R}_+^{n\times Q}$ is a slack variable. In the objective function, $C\sum_{i=1}^n\sum_{k\neq y_i}\xi_{ik}$ is the hinge loss term which presents the training classification errors. The remaining term is known as a regularization. C is a parameter that presents the trade-off between the hinge loss and the regularizer term.

For the feature selection purpose, we use a natural concept dealing with sparsity, that is the zero norm (denoted l_0 or $\|.\|_0$). The zero norm of a vector is defined as the number of its nonzero components. The function l_0, apparently very simple, is lower-semicontinuous on $\mathbb{R}^n$, but its discontinuity at the origin makes nonconvex programs involving $\|.\|_0$ challenging.

We study two models obtained from (2) by replacing the second term in the objective function with the l_0 and/or the l_2-l_0 regularization, that lead to the so called l_0-MSVM and l_2-l_0-MSVM problems which are defined respectively by

$$\min_{(w,b,\xi)\in\Omega} C\sum_{i=1}^{n}\sum_{k\neq y_i}\xi_{ik} + \sum_{k=1}^{Q}\|w_k\|_0 \quad (l_0 - MSVM) \tag{3}$$

and

$$\min_{(w,b,\xi)\in\Omega} C\sum_{i=1}^{n}\sum_{k\neq y_i}\xi_{ik} + \beta\sum_{k=1}^{Q}\|w_k\|_2^2 + \sum_{k=1}^{Q}\|w_k\|_0 \quad (l_2 - l_0 - MSVM). \tag{4}$$

Due to the l_0 term, these problems are nonsmooth and nonconvex.

During the last two decades, research is very active in models and methods optimization involving the zero norm. Works can be divided into three categories according to the way to treat the zero norm: convex approximation, nonconvex approximation, and nonconvex exact reformulation. Our work deals with the two nonconvex approaches, both are based on Difference of Convex functions (DC) programming and DC Algorithms (DCA), because our main motivation is to exploit the efficiency of DCA to solve these hard problems. DC programming and DCA were introduced by Pham Dinh Tao in their preliminary form in 1985 and have been extensively developed since 1994 by Le Thi Hoai An and Pham Dinh Tao and become now classic and more and more popular (see, e.g. [19,17,35,36], and references therein), in particular in machine learning for which they provide quite often a global solution and proved to be more robust and efficient than standard methods.

In the first approach, the l_0-norm is approximated by a DC function that leads to a DC program for which a DCA scheme is investigated. This general DCA scheme is developed to various sparse inducing DC approximation functions: the piecewise exponential function proposed in [1], the SCAD penalty function [10], the logarithm function introduced in [45], the capped-ℓ_1 function [34] and the piecewise linear function recently proposed in [18]. In the second approach, the original problem is equivalently reformulated, via an exact penalty technique in DC programming [24], as a DC program which is solved by a DCA based algorithm. Hence, using a unified DC programming framework, we unify all solution methods into DCA, and then convergence properties of our algorithms are guaranteed thanks to general convergence results

of the generic DCA scheme. We perform empirical comparative numerical experiments of 12 versions of DCA based algorithms, with various approximate functions as well as with the exact continuous reformulations.

The remainder of the paper is organized as follows. Section 2 is devoted to a brief presentation of DC Programming and DCA. The approximation approach is presented in Section 3 while the exact approach via an penalty technique is developed in Section 4. Computational experiments are reported in Section 5 and finally Section 6 concludes the paper.

2 A brief presentation of DC programming and DCA

DC programming and DCA constitute the backbone of smooth/nonsmooth nonconvex programming and global optimization. A general DC program takes the form:

$$\inf\{F(x) := G(x) - H(x) : x \in \mathbb{R}^n\}, \quad (P_{dc})$$

where G and H are lower semicontinuous proper convex functions on $\mathbb{R}^p$. Such a function F is called DC function, and $G - H$, DC decomposition of F while G and H are DC components of F. The convex constraint $x \in C$ can be incorporated in the objective function of (P_{dc}) by using the indicator function on C denoted χ_C which is defined by $\chi_C(x) = 0$ if $x \in C; +\infty$ otherwise:

$$\inf\{f(x) := G(x) - H(x) : x \in C\} = \inf\{\chi_C(x) + G(x) - H(x) : x \in \mathbb{R}^p\}.$$

A convex function θ is called *convex polyhedral* if it is the maximum of a finite family of affine functions, i.e.

$$\theta(x) = \max\{\langle a_i, x\rangle + b : i = 1, ...m\},\ a_i \in \mathbb{R}^n.$$

Polyhedral DC optimization occurs when either G or H is polyhedral convex. This class of DC optimization problems, which is frequently encountered in practice, enjoys interesting properties (from both theoretical and practical viewpoints) concerning local optimality and the convergence of DCA ([19,35]).

A point x^* is said to be *a local minimizer* of $G - H$ if $G(x^*) - H(x^*)$ is finite and there exists a neighbourhood $\mathcal{U}$ of x^* such that

$$G(x^*) - H(x^*) \leq G(x) - H(x), \quad \forall x \in \mathcal{U}. \tag{5}$$

The necessary local optimality condition for (primal) DC program (P_{dc}) is given by

$$\emptyset \neq \partial H(x^*) \subset \partial G(x^*). \tag{6}$$

The condition (6) is also sufficient (for local optimality) in many important classes of DC programs, for example, when (P_{dc}) is a DC polyhedral program with H being polyhedral convex function, or when f is locally convex at x^* (see [19,35,36]).

A point x^* is said to be *a critical point* of $G - H$ if

$$\partial H(x^*) \cap \partial g(x^*) \neq \emptyset. \tag{7}$$

The relation (7) is in fact the generalized KKT condition for (P_{dc}) and x^* is also called a generalized KKT point.

DCA is based on local optimality conditions and duality in DC programming. The main idea of DCA is simple: each iteration of DCA approximates the concave part $-H$ by its affine majorization (that corresponds to taking $y^l \in \partial H(x^l)$) and minimizes the resulting convex function.

The generic DCA scheme can be described as follows:

DCA - General scheme
Initializations: let $x^0 \in \mathbb{R}^n$ be a best guess, $l \leftarrow 0$.
Repeat
1. Calculate $y^l \in \partial H(x^l)$.

2. Calculate $x^{l+1} \in arg\min\{G(x) - H(x^l) - \langle x - x^l, y^l\rangle\} : x \in \mathbb{R}^p$.
3. $l \leftarrow l + 1$.

Untilconvergence of $\{x^l\}$.

Convergence properties of DCA and its theoretical basic can be found in [19,36,35]. It is worth mentioning that (for simplify we omit here the dual part of DCA)

i) DCA is a descent method (*without line search*): the sequences $\{G(x^l) - H(x^l)\}$ is decreasing.
ii) If $G(x^{l+1}) - H(x^{l+1}) = G(x^l) - H(x^l)$, then x^l is a critical point of $G - H$. In such a case, DCA terminates at l-th iteration.
iii) If the optimal value α of problem (P_{dc}) is finite and the infinite sequences $\{x^l\}$ is bounded then every limit point x^* of the sequences $\{x^l\}$ is a critical point of $G - H$.
iv DCA has a *linear convergence* for general DC programs, and has a finite convergence for DC polyhedral programs.
v) If H is polyhedral convex and H is differentiable at x^*, then x^* is a local minimizer of (P_{dc}).

A deeper insight into DCA has been described in [19,35–37]. For instant it is crucial to note the main features of DCA: DCA is constructed from DC components and their conjugates but not the DC function f itself which has infinitely many DC decompositions, and there are as many DCA as there are DC decompositions. Such decompositions play a critical role in determining the speed of convergence, stability, robustness, and globality of sought solutions. It is important to study various equivalent DC forms of a DC problem. This flexibility of DC programming and DCA is of particular interest from both a theoretical and an algorithmic point of view. Moreover, with suitable DC decompositions DCA generates most standard algorithms in convex and nonconvex optimization For a complete study of DC programming and DCA the reader is referred to [19,35–37] and the references therein.

In the last decade, a variety of works in Machine Learning based on DCA have been developed. The efficiency and the scalability of DCA have been proved in a lot of works (see e.g. [7, 20–23,28,32,40] and the list of reference in [17]).

3 Approximation methods based on DC programming and DCA

For simplifying the presentation, we will consider the following common optimization problem:

$$\min\left\{F(w,b,\xi) + \sum_{k=1}^{Q} \|w_k\|_0 : X = (w,b,\xi) \in \Omega\right\}, \tag{8}$$

where F stands for F_1 in the l_0-MSVM problem, and for F_2 in the l_2-l_0-MSVM problem, say

$$F_1(w,b,\xi) := C\sum_{i=1}^{n}\sum_{k\neq y_i} \xi_{ik}, \tag{9}$$

$$F_2(w,b,\xi) := C\sum_{i=1}^{n}\sum_{k\neq y_i} \xi_{ik} + \beta\sum_{k=1}^{Q} \|w_k\|_2^2. \tag{10}$$

Here F_1 is a linear function while F_2 is a quadratic convex function.

We introduce here a class of DC approximation functions of the l_0 norm. Define the step function $s : \mathbb{R} \to \mathbb{R}$ by

$$s(x) = 1 \text{ for } x \neq 0 \text{ and } s(x) = 0 \text{ for } x = 0.$$

Then for $X \in \mathbb{R}^n$ we have $\|X\|_0 = \sum_{i=1}^{n} s(X_i)$. Let $\varphi\theta : \mathbb{R} \to \mathbb{R}$ be a function depending on the parameter θ which approximates $s(x)$, say

$$\lim_{\theta\to+\infty} \varphi\theta(x) = s(x),\ \forall x \in \mathbb{R}. \tag{11}$$

Table 1 DC approximation functions

Name	Formula of $\varphi_\theta(x)$
Piecewise exponential ([1])	$1 - e^{-\alpha\|x\|}$, $\alpha > 0$
SCAD ([10])	$\begin{cases} \lambda\|x\| & \text{if } \|x\| \leq \lambda \\ -\frac{x^2 - 2\alpha\lambda\|x\| + \lambda^2}{2(\alpha-1)} & \text{if } \lambda < \|x\| \leq \alpha\lambda \\ \frac{(\alpha+1)\lambda^2}{2} & \text{if } \|x\| > \alpha\lambda \end{cases}$, $\alpha > 2, \lambda > 0$
Capped-l_1 ([33])	$\min\{1, \alpha\|x\|\}$, $\alpha > 0$
Piecewise linear ([18])	$\min\left\{1, \max\left\{0, \frac{\|x\|-a}{b-a}\right\}\right\}$, $0 < a < b$
Logarithm ([45])	$\rho\varepsilon \log\left(1 + \frac{\|x\|}{\varepsilon}\right)$, $\rho\varepsilon = \frac{1}{\log(1+\frac{1}{\varepsilon})}$

Suppose that $\varphi\theta$ can be expressed as a DC function of the form

$$\varphi\theta(x) = g(x) - h(x), \ x \in \mathbb{R} \tag{12}$$

where g and h are convex functions. Using this approximation, the l_0 term in (8) can be written as

$$\sum_{k=1}^{Q} \|w_k\|_0 \approx \sum_{k=1}^{Q}\sum_{j=1}^{d} \varphi_\theta(w_{kj}) = \sum_{k=1}^{Q}\sum_{j=1}^{d} g(w_{kj}) - \sum_{k=1}^{Q}\sum_{j=1}^{d} h(w_{kj}), \tag{13}$$

and the problem (8) can be represented as follows:

$$\min\left\{G(X) - H(X) : X \in \mathbb{R}^{Q\times d} \times \mathbb{R}^{Q} \times \mathbb{R}^{n\times Q}\right\}, \tag{14}$$

where

$$G(X) = \chi_\Omega(X) + F(X) + \sum_{k=1}^{Q}\sum_{j=1}^{d} g(w_{kj}); \quad H(X) = \sum_{k=1}^{Q}\sum_{j=1}^{d} h(w_{kj}).$$

Since the functions F, g and h are convex, G and H are convex too. Therefore (14) is a DC program. Thanks to the general DCA scheme given in Section 2, DCA applied on (14) can be described as follows.

DCA-dcApp
Initializations: Let τ be a tolerance sufficiently small, set $l = 0$. Let $X^0 = (w^0, b^0, \xi^0)$ be a guess.
Repeat

1. Compute $\overline{w}_{kj}^l \in \partial h(w_{kj}^l)$ $\forall k = 1, \ldots, Q, j = 1, \ldots, d$ and set $Y^l = (\overline{w}^l, 0, 0)$.
2. Compute $X^{l+1} = (w^{k+1}, b^{k+1}, \xi^{k+1})$ by solving the convex optimization problem

$$\min\left\{F(X) + \sum_{k=1}^{Q}\sum_{j=1}^{d} g(w_{kj}) - \sum_{k=1}^{Q}\sum_{j=1}^{d} \overline{w}_{kj}^l w_{kj} : X = (w, b, \xi) \in \Omega\right\}. \tag{15}$$

3. $l \leftarrow l + 1$.

Until$\| X^{l+1} - X^l \| \leq \tau(1 + \| X^l \|)$.

We consider now usual sparse inducing DC approximation functions φ_θ and develop the corresponding **DCA-dcApp** to solve the resulting optimization problems. The list of approximation functions is reported in Table 1. Obviously, these functions verify the conditions (11).

First, let us consider the piecewise exponential (PiE) function [1], the SCAD function [10], the Capped-ℓ_1 ([34]), and the piecewise linear (PiL) approximation recently proposed in [18]. The definition of these functions are presented in Table 1. They are all DC functions with DC decompositions given in Table 2.

The implementation of Algorithm **DCA-dcApp** according to each specific function φ_θ differs from one of others by the computation of $\overline{w}_{kj}^l \in \partial h(w_{kj}^l)$ in the step 1, and the subproblem

Table 2 DC decomposition $\varphi = g - h$ and calculation of ∂h.

Name	$g(x)$	$h(x)$	$\overline{x} \in \partial h(x)$
PiE	$\alpha\|x\|$	$\alpha\|x\| - 1 + e^{-\alpha\|x\|}$	$\text{sgn}(x)(\alpha\|x\| - e^{-\alpha\|x\|})$
SCAD	$\lambda\|x\|$	$\begin{cases} 0 & \text{if } \|x\| \le \lambda \\ \frac{(\|x\|-\lambda)^2}{2(\alpha-1)} & \text{if } \lambda \le \|x\| \le \alpha\lambda \\ \lambda\|x\| - \frac{(\alpha+1)\lambda^2}{2} & \text{otherwise} \end{cases}$	$\begin{cases} 0 & \text{if } \|x\| \le \lambda \\ \text{sgn}(x)\frac{\|x\|-\lambda}{\alpha-1} & \text{if } \lambda < \|x\| < \alpha\lambda \\ \text{sgn}(x)\lambda & \text{otherwise} \end{cases}$
Capped-l_1	$\alpha\|x\|$	$\max\{1, \alpha\|x\|\} - 1$	$\begin{cases} 0 & \text{if } \|x\| \le \frac{1}{\alpha} \\ \text{sgn}(x)\alpha & \text{otherwise} \end{cases}$
PiL	$\frac{\max\{a,\|x\|\}}{b-a}$	$\frac{\max\{b,\|x\|\}}{b-a} - 1$	$\begin{cases} 0 & \text{if } \|x\| \le b \\ \frac{\text{sgn}(x)}{b-a} & \text{otherwise} \end{cases}$

(15) in the step 2. The computation of $\overline{w}_{kj}^l \in \partial h(w_{kj}^l)$ is given in Table 2. The subproblem, for instance, in case of l_0-MSVM with the piecewise exponential (PiE) function, is written as follows (to simplify the presentation we use φ instead of φ_θ):

- l_0-**DCA-PiE** ($\varphi =$ PiE, $F = F_1$)

$$\min\left\{ C\sum_{i=1}^{n}\sum_{k\neq y_i}\xi_{ik} + \sum_{i=1}^{Q}\sum_{j=1}^{d}\max(\alpha w_{ij}, -\alpha w_{ij}) - \sum_{k=1}^{Q}\sum_{j=1}^{d}\overline{w}_{kj}^l w_{kj} : X=(w,b,\xi)\in\Omega \right\}$$

$$\Longleftrightarrow \left\{ \begin{array}{ll} \min\limits_{(w,b,\xi,t)} & C\sum\limits_{i=1}^{n}\sum\limits_{k\neq y_i}\xi_{ik} + \sum\limits_{k=1}^{Q}\sum\limits_{j=1}^{d} t_{kj} - \sum\limits_{k=1}^{Q}\sum\limits_{j=1}^{d}\overline{w}_{kj}^l w_{kj} \\ s.t. & (w,b,\xi)\in\Omega, t_{kj}\ge\alpha w_{kj}, t_{kj}\ge -\alpha w_{kj}, \forall k=1,\ldots,Q, j=1,\ldots,d. \end{array} \right. \tag{16}$$

Similarly, the subproblem in other algorithms is given by:

- $l_2 - l_0$-**DCA-PiE** ($\varphi =$ PiE, $F = F_2$)

$$\left\{ \begin{array}{ll} \min\limits_{(w,b,\xi,t)} & \beta\sum\limits_{k=1}^{Q}\|w_k\|_2^2 + C\sum\limits_{i=1}^{n}\sum\limits_{k\neq y_i}\xi_{ik} + \sum\limits_{k=1}^{Q}\sum\limits_{j=1}^{d} t_{kj} - \sum\limits_{k=1}^{Q}\sum\limits_{j=1}^{d}\overline{w}_{kj}^l w_{kj} \\ s.t. & (w,b,\xi)\in\Omega, t_{kj}\ge\alpha w_{kj}, t_{kj}\ge -\alpha w_{kj}, \forall k=1,\ldots,Q, j=1,\ldots,d. \end{array} \right. \tag{17}$$

- l_0-**DCA-SCAD** ($\varphi =$ SCAD, $F = F_1$)

$$\left\{ \begin{array}{ll} \min\limits_{(w,b,\xi,t)} & C\sum\limits_{i=1}^{n}\sum\limits_{k\neq y_i}\xi_{ik} + \sum\limits_{k=1}^{Q}\sum_{j=1}^{d} t_{kj} - \sum\limits_{k=1}^{Q}\sum\limits_{j=1}^{d}\overline{w}_{kj}^l w_{kj} \\ s.t. & (w,b,\xi)\in\Omega,\ t_{kj}\ge\lambda w_{kj}, t_{kj}\ge -\lambda w_{kj}, \forall k=1,\ldots,Q, j=1,\ldots,d. \end{array} \right. \tag{18}$$

- $l_2 - l_0$-**DCA-SCAD** ($\varphi =$ SCAD, $F = F_2$)

$$\left\{ \begin{array}{ll} \min\limits_{(w,b,\xi,t)} & \beta\sum\limits_{k=1}^{Q}\|w_k\|_2^2 + C\sum\limits_{i=1}^{n}\sum\limits_{k\neq y_i}\xi_{ik} + \sum_{k=1}^{Q}\sum\limits_{j=1}^{d} t_{kj} - \sum\limits_{k=1}^{Q}\sum\limits_{j=1}^{d}\overline{w}_{kj}^l w_{kj} \\ s.t. & (w,b,\xi)\in\Omega,\ t_{kj}\ge\lambda w_{kj}, t_{kj}\ge -\lambda w_{kj}, \forall k=1,\ldots,Q, j=1,\ldots,d. \end{array} \right. \tag{19}$$

- l_0-**DCA-Capped-l_1** ($\varphi =$ Capped-l_1, $F = F_1$)

$$\left\{ \begin{array}{ll} \min\limits_{(w,b,\xi,t)} & C\sum\limits_{i=1}^{n}\sum\limits_{k\neq y_i}\xi_{ik} + \sum\limits_{k=1}^{Q}\sum\limits_{j=1}^{d} t_{kj} - \sum\limits_{k=1}^{Q}\sum\limits_{j=1}^{d}\overline{w}_{kj}^l w_{kj} \\ s.t. & (w,b,\xi)\in\Omega,\ t_{kj}\ge\alpha w_{kj}, t_{kj}\ge -\alpha w_{kj}, \forall k=1,\ldots,Q, j=1,\ldots,d. \end{array} \right. \tag{20}$$

- $l_2 - l_0$-**DCA-Capped-l_1** ($\varphi =$ Capped-l_1, $F = F_2$)

$$\left\{ \begin{array}{ll} \min\limits_{(w,b,\xi,t)} & C\sum\limits_{i=1}^{n}\sum\limits_{k\neq y_i}\xi_{ik} + \beta\sum\limits_{k=1}^{Q}\|w_k\|_2^2 + \sum\limits_{k=1}^{Q}\sum\limits_{j=1}^{d} t_{kj} - \sum\limits_{k=1}^{Q}\sum_{j=1}^{d}\overline{w}_{kj}^l w_{kj} \\ s.t. & (w,b,\xi)\in\Omega,\ t_{kj}\ge\alpha w_{kj}, t_{kj}\ge -\alpha w_{kj}, \forall k=1,\ldots,Q, j=1,\ldots,d. \end{array} \right. \tag{21}$$

- l_0**-DCA-PiL** ($\varphi = \text{PiL}$, $F = F_1$)

$$\begin{cases} \min\limits_{(w,b,\xi,t)} \frac{1}{b-a} \sum\limits_{k=1}^{Q} \sum\limits_{j=1}^{d} t_{kj} + C \sum\limits_{i=1}^{n} \sum_{k\neq y_i} \xi_{ik} - \sum\limits_{k=1}^{Q} \sum\limits_{j=1}^{d} \overline{w}_{kj}^{l} w_{kj} \\ s.t. \quad (w,b,\xi) \in \Omega, t_{kj} \geq a, t_{kj} \geq w_{kj}, t_{kj} \geq -w_{kj}, \forall k = 1,\ldots,Q, j = 1,\ldots,d. \end{cases} \tag{22}$$

- $l_2 - l_0$**-DCA-PiL** ($\varphi = \text{PiL}$, $F = F_2$)

$$\begin{cases} \min\limits_{(w,b,\xi,t)} \frac{1}{b-a} \sum\limits_{k=1}^{Q} \sum\limits_{j=1}^{d} t_{kj} + \beta \sum\limits_{k=1}^{Q} \|w_k\|_2^2 + C \sum\limits_{i=1}^{n} \sum\limits_{k\neq y_i} \xi_{ik} - \sum\limits_{k=1}^{Q} \sum\limits_{j=1}^{d} \overline{w}_{kj}^{l} w_{kj} \\ s.t. \quad (w,b,\xi) \in \Omega, t_{kj} \geq a, t_{kj} \geq w_{kj}, t_{kj} \geq -w_{kj}, \forall k = 1,\ldots,Q, j = 1,\ldots,d. \end{cases} \tag{23}$$

We observe that when $F = F_1$ (resp. $F = F_2$), the subproblem (15) is a linear (resp. convex quadratic) program.

Convergence properties of the algorithms:

i) When $F = F_1$, for all the four above approximation functions, (14) is a polyhedral DC program since the first DC component G is polyhedral convex function. Thus, the corresponding DCA has finite convergence, say the sequence $\{(w^k, b^k, \xi^k)\}$ converges to a critical point (w^*, b^*, ξ^*) after a finite number of iterations. Moreover, by considering the dual problem of (14) in which the second DC component is polyhedral convex and using the DCA's convergence property v) mentioned in Section 2, we can prove that (w^*, b^*, ξ^*) is almost always a local minimizer of (14).

ii) When the approximation function φ is Capped-l_1 or PiL, (14) is a polyhedral DC program since the second DC component H is polyhedral convex (for both F_1 and F_2). Hence the corresponding DCA has finite convergence. Moreover, by property v) mentioned in Section 2, if H is differentiable at (w^*, b^*, ξ^*), then (w^*, b^*, ξ^*) is actually a local minimizer of (14). More precisely, when $\varphi = Capped{-}l_1$ (resp. $\varphi = PiL$), if $w_{kj}^* \notin \{-\frac{1}{\alpha}, \frac{1}{\alpha}\}$ (resp. $w_{kj}^* \notin \{-b, b\}$), then (w^*, b^*, ξ^*) is a local minimizer of (14).

Consider now the logarithm (Log) approximation function defined in Table 1. We introduce the variable $v \in \mathbb{R}_+^{Q\times d}$ and rewrite the problem (8) as

$$\min_{(w,b,\xi,v)\in\widetilde{\Omega}} F(w,b,\xi) + \textstyle\sum_{k=1}^{Q} \sum_{j=1}^{d} \varphi(v_{kj}), \tag{24}$$

where $\widetilde{\Omega} = \{(w,b,\xi,v) : (w,b,\xi) \in \Omega, -v \leq w \leq v\}$.

Let $g(x) = 0$ and $h(x) = -\rho\varepsilon \log(1 + x/\varepsilon)$. Clearly $\varphi(x) = g(x) - h(x)$ and g, h are convex functions on $\mathbb{R}_+$. Then we can express the problem (24) as a DC program

$$\min\left\{\widetilde{G}(\widetilde{X}) - \widetilde{H}(\widetilde{X}) : \widetilde{X} = (w,b,\xi,v) \in \mathbb{R}^{Q\times d+Q+n\times Q+Q\times d}\right\}, \tag{25}$$

where $\widetilde{G}(\widetilde{X}) = \chi_{\widetilde{\Omega}}(\widetilde{X}) + F(w,b,\xi)$, $\widetilde{H}(\widetilde{X}) = \sum_{k=1}^{Q}\sum_{j=1}^{d} h(v_{kj})$ are convex functions on $\widetilde{\Omega}$.

It can be seen that $\widetilde{H}$ is differentiable and $\nabla\widetilde{H}(w,b,\xi,v) = (0,0,0,\overline{v})$, where

$$\overline{v}_{kj} = \nabla h(v_{kj}) = -\frac{\rho\varepsilon}{v_{kj}+\varepsilon}, \quad k = 1,\ldots,Q,\ j = 1,\ldots,d. \tag{26}$$

Hence DCA applied to (25) can be described as follows.

l_0**-DCA-Log**

Initialization Let τ be a tolerance sufficiently small, set $l = 0$. Choose $\tilde{X}^0 = (w^0, b^0, \xi^0, v^0)$ be a guess.

Repeat

1. Compute $\overline{v}^l$ via (26).
2. Compute $\tilde{X}^{l+1} = (w^{k+1}, b^{k+1}, \xi^{k+1}, v^{k+1})$ by solving the linear problem

$$\min\left\{C\sum_{i=1}^{n}\sum_{k\neq y_i}\xi_{ik} - \sum_{k=1}^{Q}\sum_{j=1}^{d}\overline{v}_{kj}^{l} v_{kj} : (w,b,\xi,v) \in \widetilde{\Omega}\right\}. \tag{27}$$

3. $l \leftarrow l+1$.
Until $\|X^{l+1}-X^l\| \leq \tau(1+\|X^l\|)$.

l_2-l_0**-DCA-Log**
Replace the step 2 of l_0**-DCA-Log** by
Compute $\tilde{X}^{l+1}=(w^{k+1},b^{k+1},\xi^{k+1},v^{k+1})$ by solving the convex quadratic problem

$$\min\left\{C\sum_{i=1}^{n}\sum_{k\neq y_i}\xi_{ik}+\beta\sum_{k=1}^{Q}\|w_k\|_2^2-\sum_{k=1}^{Q}\sum_{j=1}^{d}\overline{v}_{kj}v_{kj}:(w,b,\xi,v)\in\widetilde{\Omega}\right\}. \tag{28}$$

Similarly to the property i) above, when $F=F_1$, (25) is a DC polyhedral program. So l_0-**DCA-Log** has a finite convergence. Moreover, l_0**-DCA-Log** converges almost always to a local minimizer of (25).

4 An exact reformulation approach via exact penalty techniques

In this section we reformulate equivalently the problem (8) in the form of a DC proram and develop a DCA based algorithm for solving it. Denote by e the vector of ones in the appropriate space. Let $u\in\mathbb{R}^{Q\times d}$ be the binary variable defined by:

$$u_{kj}=\begin{cases}1 \text{ if } w_{kj}\neq 0\\ 0 \text{ if } w_{kj}=0,\end{cases}\ \forall k=1,\ldots,Q, j=1,\ldots,d. \tag{29}$$

We have

$$\sum_{k=1}^{Q}\|w_k\|_0=\sum_{k=1}^{Q}\sum_{j=1}^{d}u_{kj}=\langle e,u\rangle. \tag{30}$$

Suppose that Ω is bounded in the variable w, i.e. $\Omega\subset[-\mathcal{B},\mathcal{B}]^{Q\times d}\times\mathbb{R}^Q\times\mathbb{R}_+^{n\times Q}$ for some $\mathcal{B}>0$. Then the problem (8) can be expressed as

$$\begin{cases}\min\limits_{(w,b,\xi,u)} & F(w,b,\xi)+\langle e,u\rangle\\ s.t. & (w,b,\xi)\in\Omega,\\ & |\ w_{kj}\ |\leq \mathcal{B}u_{kj},\ u_{kj}\in\{0,1\},\forall k=1,\ldots,Q;\ j=1,\ldots,d.\end{cases} \tag{31}$$

Let $\overline{p}:\mathbb{R}^{Q\times d}\times\mathbb{R}^Q\times\mathbb{R}^{n\times Q}\times[0,1]^{Q\times d}\to\mathbb{R}$ be the function defined as $\overline{p}(w,b,\xi,u)=p(u)$ with $p:\mathbb{R}^{Q\times d}\to\mathbb{R},\ p(u):=\sum\limits_{k=1}^{Q}\sum\limits_{j=1}^{d}u_{kj}(1-u_{kj})$, and let Λ be the polyhedral convex set determined by

$$\Lambda:=\left\{(w,b,\xi,u)\in\Omega\times[0,1]^{Q\times d}:|w_{kj}|\leq|\ \mathcal{B}u_{kj},\ \forall k=1,\ldots,Q,j=1,\ldots,d,\right\}. \tag{32}$$

We observe that $\overline{p}$ is concave and $\overline{p}(w,b,\xi,u)\geq 0\ \forall(w,b,\xi,u)\in\Lambda$. By dint of the exact penalty technique developed recently in [24], the problem (31) is equivalent to the following continuous optimization problem, with sufficient large positive numbers $\eta>\eta_0\geq 0$ (called penalty parameters)

$$\min\{F(w,b,\xi)+\langle e,u\rangle+\eta p(u):\mathbb{X}=(w,b,\xi,u)\in\Lambda\} \tag{33}$$

We investigate now a DCA based algorithm for solving (33). Let $\overline{G}$ and $\overline{H}$ be the functions defined by

$$\overline{G}(\mathbb{X}):=F(X)+\chi_\Lambda(\mathbb{X})$$

and

$$\overline{H}(\mathbb{X}):=\eta\sum_{k=1}^{Q}\sum_{j=1}^{d}u_{kj}^2-(\eta+1)\sum_{k=1}^{Q}\sum_{j=1}^{d}u_{kj}.$$

The problem (33) can be expressed as:

$$\min\left\{\overline{G}(\mathbb{X})-\overline{H}(\mathbb{X}):\mathbb{X}=(w,b,\xi,u)\in\mathbb{R}^{Q\times d+Q+n\times Q+Q\times d}\right\}. \tag{34}$$

Obviously, $\overline{G}$ and $\overline{H}$ are convex functions and so (34) is a DC program. DCA applied on (34) consists of computing, at each iteration l,

$$\mathbb{Y}^l\in\partial\overline{H}(\mathbb{X}^l),\ \mathbb{X}^{l+1}\in\arg\min\left\{\overline{G}(\mathbb{X})-\ \langle\mathcal{Y}^l,\mathbb{X}\rangle:\mathbb{X}\ \in\Lambda\right\}.$$

Clearly, $\overline{H}$ is differentiable and $\mathbb{Y}^l=\nabla\overline{H}(\mathbb{X}^l)$ can be computed as

$$\mathbb{Y}^l=(0,0,0,\overline{u}^l),\ \text{with}\ \overline{u}^l_{kj}=2\eta u^l_{kj}-(\eta+1),\ \forall k=1,\ldots,Q,j=1,\ldots,d. \tag{35}$$

And $\mathbb{X}^{l+1}=(w^{l+1},b^{l+1},\xi^{l+1},u^{l+1})$ is an optimal solution of the convex optimization problem

$$\min\left\{F(X)-\langle\overline{u}^l,u\rangle:\mathbb{X}\in\Lambda\right\}\ . \tag{36}$$

Hence, the DCA applied to (34) when $F=F_1$ (the l_0-MSVM problem) is described as follows:

l_0-DCA-Econ
Initializations: let $\tau>0$ be given and $\mathbb{X}^0=(w^0,b^0,\xi^0,u^0)$ be an initial point. Select $\eta,\mathcal{B},C$ and set $l=0$.
Repeat

1. Calculate $\mathbb{Y}^l$ via (35).
2. Calculate $\mathbb{X}^{l+1}$, an optimal solution of the linear program

$$\min\left\{C\sum_{i=1}^{n}\sum_{k\neq y_i}\xi_{ik}-\langle\overline{u}^l,u\rangle:\mathbb{X}=(w,b,\xi,u)\in\Lambda\right\}$$

3. $l\leftarrow l+1$.

Until $\|\mathbb{X}^{l+1}-\mathbb{X}^l\|\leq\tau\|\mathbb{X}^l\|$.

Note that since $\overline{G}$ is polyhedral convex, (34) is a polyhedral DC program and then **l_0-DCA-Econ** has a finite convergence. Furthermore, by considering the dual problem of (34) in which the second DC component is polyhedral convex and using the property v) mentioned in Section 2, we can prove that **l_0-DCA-Econ** converges almost always to a local minimizer of (34).

Similarly, the DCA applied to (34) when $F=F_2$ (the l_2-l_0-MSVM problem) is described as follows:

l_2-l_0-DCA-Econ
Replace the step 2 of **l_0-DCA-Econ** by
Calculate $\mathbb{X}^{l+1}$, an optimal solution of the convex quadratic problem

$$\min\left\{C\sum_{i=1}^{n}\sum_{k\neq y_i}\xi_{ik}+\beta\sum_{k=1}^{Q}\|w_k\|_2^2-\langle\overline{u}^l,u\rangle:\mathbb{X}=(w,b,\xi,u)\in\Lambda\right\}.$$

5 Numerical experiments

We have implemented the algorithms in the V.S C++ v6.0 environment and performed the experiments a Intel Core^{TM} I7 (2×2.2 Ghz) processor, 4 GB RAM.

5.1 Datasets

We consider eight popular datasets often used for feature selection. The Lung Cancer (LUN), Optical Recognition of Handwritten Digits (OPT), Libras Movement (MOV), Semeion Handwritten Digit (SEM), Multiple Features (MFE), CNAE-9 (CNA) and Internet Advertisement (ADV) datasets are taken from UCI Machine Learning Repository. The ADN dataset (ADN) that consists of 3186 genes, described by 60 DNA sequel elements can be found at `ftp://genbank.bio.net`. For the OPT dataset, both training set and test set are given in UCI Machine Learning Repository. The LUN dataset contains a very small number of samples, therefore the whole dataset is used as the training set as well as the test set. For the remaining datasets, training and test sets are randomly sampled from the original set with 60% for training and the remaining 40% for test (cf. 5.3). These datasets are described in details in the table 3.

Dataset	#feature	#class	#train	#test
LUN	56	3	16	16
OPT	63	10	3823	1797
ADN	60	3	1913	1273
MOV	90	15	225	135
SEM	256	10	960	633
MFE	649	10	1200	800
CNA	856	9	648	432
ADV	1558	2	1967	1312

Table 3 The description of the datasets

5.2 Experiment setting

The CPLEX 12.5 solver is used to solve linear and convex quadratic problems.

The parameters are taken as follows: for all methods, the most appropriate values of the parameter C are chosen by a five-folds cross-validation; β, the coefficient of the l_2 term is set to 0.01; the tolerance τ (in the stoping criterion of DCA) is set to 10^{-6}.

Observe that, in one hand, theoretically, the larger value of α is, the better DC approximation of l_0-norm would be, and on another hand, practically, when α is large, the algorithms give sometimes bad local minima. Hence, we use an α updating procedure during the algorithms. Starting with a small value of α_0, we increase it at each iteration l by $\alpha_{l+1} = \alpha_l + \Delta\alpha$ until a given threshold $\overline{\alpha}$. For l_0-DCA_PiE and l_2-l_0-DCA_PiE, $\alpha_0 = 1.5$ and $\Delta\alpha = 0.5$, $\overline{\alpha} = 5.5$. For l_0-DCA_Capped-l_1 and l_2-l_0-DCA_Capped-l_1, we set $\alpha_0 \in \{0.7, 0.8, 0.9\}, \Delta\alpha = 0.2$, $\overline{\alpha} = 5.5$. For l_0-DCA_PiL and l_2-l_0-DCA_PiL, we take $a = 10^{-6}$ and $b \in \{10^{-4}, 10^{-3}, ..., 10^{-1}, 0.2, 0.3\}$. The parameters α and λ in l_0-DCA_SCAD and l_2-l_0-DCA_SCAD are, respectively, set to 3.4 and 0.4 as proposed by Fan and Li in [10]. In l_0-DCA_Log and l_2-l_0-DCA_Log, ε is set to 10^{-4}. Finally, for l_0-DCA_Econ and l_2-l_0-DCA_Econ, the starting value of η is chosen in $\{10, 20, ..., 50\}$ and η is doubled at each iteration until $10,000$. The parameter $\mathcal{B}$ is set to 1000.

The starting vectors w^0, u^0 and v^0 of the DCA based algorithms are randomly chosen in $[-0.5, 0.5]^{Q\times d}$.

We compare our methods with one of the best algorithms for feature selection in MSVM, called the Adaptive Sub-Norm (ASN) method (see [47] for more details). To select relevant features, we first compute the feature ranking score $c_j, j = 1, ..., d$ for each feature ([9]) as follows

$$c_j = \sum_{i=1}^{Q} |w_{ij}|.$$

This ranking score is then normalized. Let $\varsigma = \max_j c_j$, we compute $c_j = \frac{c_j}{\varsigma}$ for each $j = 1, ..., d$. Then, we remove the features j for which c_j is smaller than a given threshold (0.01 in our

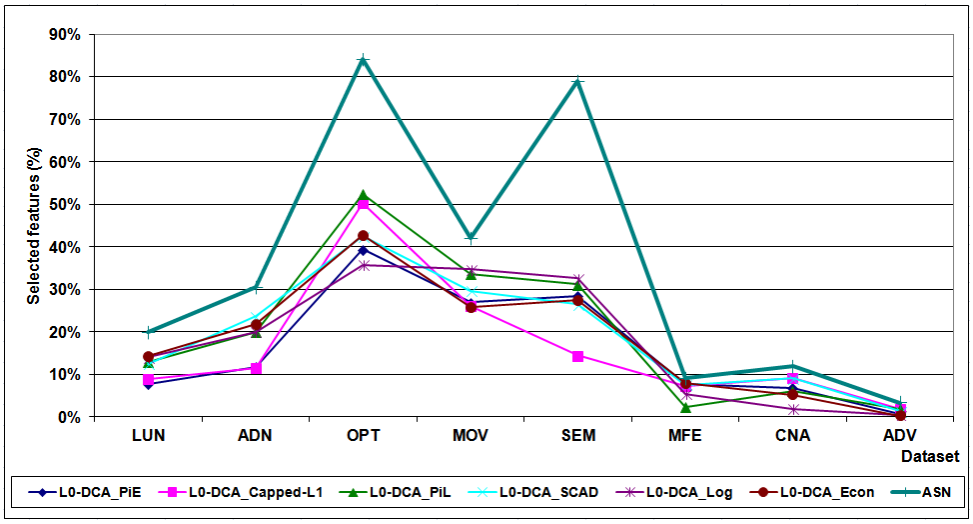


Fig. 1 Selected features (%) of l_0-DCAs and ASN

experiments). After removing features, for computing the accuracy of classifier, we apply again l_2-MSVM (2) on the new training datasets and calculate the classification's accuracy on the new test sets.

5.3 Numerical results

We are interested in the efficiency (the sparsity and the classification error) and the rapidity of the algorithms. The comparative results of the DCA based algorithms and the concurrent ASN method (resp. of the DCA based algorithms) for the l_0 model (resp. the l_2-l_0 model) are presented in Table 4 (resp. Table 5). In each table, we present the number of selected features and the corresponding percentage, the accuracy of classifiers and the CPU times of all algorithms. For the ADN, MOV, SEM, MFE, CNA and ADV datasets, we chose randomly 60% for training and the 40% remaining for test. This procedure is repeated 10 times. On each given training set and test set, each algorithm is performed 10 times and the results are reported, for each dataset, on average and its standard deviation (the best results are emphasized in bold font). For an easy observation, the number of selected features and the corresponding accuracy of classifiers on the l_0 model are shown in the figures 1 and 2.

Comments on numerical results:

Sparsity of solutions:

- All the DCA based algorithms applied on the l_0 model give a good sparsity of solutions. Their percentage of selected features vary from 0.33% to 52.49%. On all datasets, DCAs are better than ASN which selects from 3.39% to 84.13% features. On average of 8 datasets, DCA based algorithms on the l_0 model selects less than 20.52% of features while ASN selects 35.04% of features.
- On average, l_0-DCA-Capped-l_1 gives the best solutions in term of sparsity (averagely, its selected 16.28% of features).
- The percentage of selected features of the DCA based algorithms for the l_2-l_0 model are quite close to (but slightly larger than) those for the l_0 model and, on all datasets, they are better than those given by ASN. On average, the results of the l_2-l_0 DCAs vary from 16.898% to 24.30%.

Accuracy of classifiers:

- The DCA based algorithms on both l_0 and l_2-l_0 models not only provide a good performance in term of feature selection, but also give a high accuracy of classifiers (from 68.75% to 96.88%

	Dataset	l_0-DCA_PiE	l_0-DCA_Capped-l_1	l_0-DCA_PiL	l_0-DCA_SCAD	l_0-DCA_Log	l_0-DCA_Econ	ASN
Number (and percentage) of selected features	LUN	**4.38**±0.48	5.02±1.11	7.3±1.27	7.02±0.88	7.84±0.79	8.00±0.84	11.2±1.17
		7.82%±0.86%	8.96%±1.98%	13.04%±2.27%	12.54%±1.57%	14.00%±1.41%	14.29%±1.50%	20.00%±2.09%
	OPT	25.82±2.36	31.78±1.21	33.07±0.80	26.78±0.63	**22.58**±0.91	26.96±0.56	53±0.89
		40.98%±3.75%	50.44%±1.92%	52.49%±1.27%	42.51%±1.00%	**35.84**%±1.44%	42.79%±0.89%	84.13%±1.41%
	ADN	7.10±2.71	**6.98**±0.58	12.04±0.92	14.18±0.82	12.00±0.73	13.07±0.85	18.30±0.64
		11.83%±4.52%	**11.63**%±0.97%	20.07%±1.53%	23.63%±1.37%	20.00%±1.22%	21.78%±1.42%	30.50%±1.07%
	MOV	24.27±1.85	23.53±0.98	33.13±0.69	26.69±0.84	31.24±1.32	**23.27**±1.36	37.90±0.94
		26.97%±2.06%	26.14%±1.09%	36.81%±0.77%	29.66%±0.93%	34.71%±1.47%	**25.86**%±1.51%	42.11%±1.04%
	SEM	73.04±1.11	**37.16**±0.76	79.87±1.15	67.73±0.49	83.42±1.14	70.40±1.27	202.4±3.83
		28.53%±0.43%	**14.52**%±0.30%	31.20%±0.45%	26.46%±0.19%	32.59%±0.45%	27.50%±0.50%	79.06%±1.50%
	MFE	49.78±2.32	47.84±1.13	**15.91**±0.81	49.27±1.00	34.76±0.97	51.20±1.26	59±0.77
		7.67%±0.36%	7.37%±0.17%	**2.45**%±0.12%	7.59%±0.15%	5.36%±0.15%	7.89%±0.19%	9.09%±0.12%
	CNA	58.80±5.87	78.98±0.91	52.09±1.36	77.80±1.13	**16.04**±0.76	45.18±1.40	102.90±1.51
		6.87%±0.69%	9.23%±0.11%	6.09%±0.16%	9.09%±0.13%	**1.87**%±0.09%	5.28%±0.16%	12.02%±0.18%
	ADV	11.02±1.61	30.56±1.22	30.96±1.03	20.93±0.57	6.02±0.71	**5.18**±0.71	52.8±1.89
		0.71%±0.10%	1.96%±0.08%	1.99%±0.07%	1.34%±0.04%	0.39%±0.05%	**0.33**%±0.05%	3.39%±0.12%
	Average	16.42%	**16.28**%	20.52%	19.10%	18.09%	18.21%	35.04%
Accuracy of classifiers (%)	LUN	68.75±0.00	68.75±0.00	68.75±0.00	68.75±0.00	68.75±0.00	68.75±0.00	68.75±0.00
	OPT	93.55±0.17	93.95±1.01	**95.25**±0.56	94.58±0.15	92.28±0.51	94.12±0.85	93.23±0.46
	ADN	86.86±0.97	83.21±0.98	85.88±2.23	**87.07**±0.64	86.75±1.12	86.12±0.98	86.51±0.67
	MOV	**75.93**±3.95	69.29±0.36	75.68±1.02	73.49±0.53	70.12±0.98	72.22±0.54	71.02±0.89
	SEM	81.89±0.16	**90.55**±0.26	83.01±0.25	79.89±0.56	82.56±0.54	82.55±0.32	86.14±0.66
	MFE	95.73±0.15	96.15±0.42	96.35±0.55	96.22±0.27	95.33±0.15	**96.88**±0.52	95.68±0.32
	CNA	86.99±1.66	90.59±0.56	89.33±0.96	**90.68**±0.51	76.57±0.18	86.67±0.95	90.12±0.74
	ADV	**95.47**±1.01	93.64±0.68	93.25±0.45	94.28±0.77	94.52±1.15	93.66±0.51	93.42±0.54
	Average	85.65	85.77	**85.94**	85.62	83.36	85.12	85.61
CPU time (in second)	LUN	0.11±0.02	0.07±0.03	0.11±0.04	0.13±0.02	**0.05**±0.06	0.09±0.03	0.11±0.02
	OPT	89.88±1.46	**55.05**±8.12	186.57±3.52	56.88±2.23	199.86±1.02	104.35±1.65	425.98±0.77
	ADN	**1.77**±0.62	3.39±0.38	2.85±0.15	3.32±0.27	4.89±1.33	2.32±0.56	11.21±0.69
	MOV	48.52±10.82	160.35±3.35	**0.59**±0.24	45.36±11.25	0.61±0.19	6.52±0.98	25.56±0.71
	SEM	145.86±1.88	77.64±1.13	172.56±4.12	155.85±2.86	25.35±1.22	**12.35**±0.77	169.99±0.56
	MFE	300.51±18.55	**70.68**±1.56	421.26±2.89	302.56±7.54	93.26±0.31	432.55±12.25	441.32±1.01
	CNA	11.85±4.81	60.51±5.64	11.52±0.36	11.21±0.67	**3.12**±0.11	12.23±1.54	42.97±0.96
	ADV	4.11±0.18	3.55±0.58	3.21±0.32	3.35±0.15	**1.81**±0.24	2.78±0.77	13.38±0.64
	Average	75.33	53.91	99.83	72.33	**41.12**	71.65	141.32

Table 4 Comparative results of l_0-DCAs and the concurrent ASN algorithm

	Dataset	l_2-l_0-DCA_PiE	l_2-l_0-DCA_Capped-l_1	l_2-l_0-DCA_PiL	l_2-l_0-DCA_SCAD	l_2-l_0-DCA_Log	l_2-l_0-DCA_Exp
Number (and percentage) of selected features	LUN	**4.53**±0.50	6.40±1.20	7.69±0.94	7.00±1.19	5.00±0.30	9.02±1.16
	OPT	**8.09**%±0.89%	11.43%±2.14%	13.73%±1.68%	12.50%±2.13%	8.93%±0.54%	16.11%±2.07%
	ADN	25.73±1.24	32.31±1.98	34.31±0.66	46.82±0.57	**22.47**±1.02	27.87±1.20
	MOV	40.84%±1.97%	51.29%±3.14%	54.46%±1.05%	74.32%±0.90%	**35.67**%±1.62%	44.24%±1.90%
	SEM	14.72±1.77	**6.19**±1.20	18.02±0.41	14.98±0.41	12.13±0.63	13.02±0.33
	MFE	24.53%±2.95%	**10.32**%±2.00%	30.03%±0.68%	24.97%±0.68%	20.22%±1.05%	21.70%±0.55%
	CNA	32.07±4.80	25.62±0.48	34.34±0.80	32.78±1.49	31.04±0.42	**24.98**±0.54
	ADV	35.63%±5.33%	28.47%±0.53%	38.16%±0.89%	36.42%±1.66%	34.49%±0.47%	**27.76**%±0.60%
	Average	74.53±1.86	**38.2**±0.54	81.00±1.46	70.38±1.08	82.96±0.89	82.78±0.87
		29.11%±0.73%	**14.92**%±0.21%	31.64%±0.57%	27.49%±0.42%	32.41%±0.35%	32.34%±0.34%
Accuracy of classifiers (%)	LUN	51.09±1.75	48.18±0.85	51.42±1.24	49.44±1.34	**35.02**±1.02	54.47±0.83
	OPT	7.87%±0.27%	7.42%±0.13%	7.92%±0.19%	7.62%±0.21%	**5.40**%±0.16%	8.39%±0.13%
	ADN	59.08±4.26	79.67±1.25	64.09±0.69	77.91±0.69	**27.04**±1.38	44.93±1.06
	MOV	6.90%±0.50%	9.31%±0.15%	7.49%±0.08%	9.10%±0.08%	**3.16**%±0.16%	5.25%±0.12%
	SEM	11.46±2.20	30.80±1.22	14.00±0.73	31.11±1.49	6.04±0.63	**5.91**±0.63
	MFE	0.74%±0.14%	1.98%±0.08%	0.90%±0.05%	2.00%±0.10%	0.39%±0.04%	**0.38**%±0.04%
	CNA	19.22%	**16.89**%	23.04%	24.30%	17.58%	19.52%
	ADV	68.75±0.00	68.75±0.00	68.75±0.00	68.75±0.00	68.75±0.00	68.75±0.00
	Average	93.59±0.16	93.88±0.69	**95.81**±0.11	94.35±0.15	92.28±0.19	94.87±1.01
CPU time (in second)	LUN	**87.26**±0.88	83.19±1.51	85.81±2.56	87.21±0.73	85.45±3.18	86.61±3.35
	OPT	72.31±0.98	69.84±1.21	**76.58**±0.89	73.53±1.54	69.98±1.33	73.44±1.64
	ADN	82.11±0.21	**90.57**±0.56	83.15±0.11	82.51±0.63	84.56±0.25	82.88±0.52
	MOV	95.88±0.22	**96.89**±0.18	96.88±0.19	96.44±0.29	95.31±0.15	96.35±0.29
	SEM	85.44±2.13	**90.31**±0.52	89.55±0.54	90.15±0.89	84.47±1.59	86.33±0.68
	MFE	**95.94**±0.58	93.66±0.51	95.12±0.69	95.14±0.69	94.33±0.69	93.98±0.58
	CNA	85.16	85.89	**86.46**	86.01	84.39	85.40
	ADV	0.12±0.03	0.12±0.05	0.14±0.02	0.13±0.01	**0.02**±0.01	0.07±0.02
	Average	156.51±3.13	185.68±0.89	185.65±4.21	**39.98**±1.59	429.35±8.89	205.32±1.52
		42.53±4.18	95.35±10.26	30.25±1.15	21.32±1.98	19.66±0.25	**4.99**±10.23
		48.18±14.81	166.53±5.69	72.58±1.11	52.39±6.85	**0.89**±0.15	35.92±22.89
		351.25±3.32	189.65±3.25	309.25±1.99	351.35±1.89	103.73±2.35	**44.39**±1.82
		531.21±7.22	152.96±1.26	655.25±3.59	512.39±7.62	201.36±1.91	**125.39**±1.57
		55.17±9.76	61.25±5.78	**15.68**±0.52	53.22±3.12	20.55±0.32	18.56±8.95
		4.88±0.79	5.83±1.89	3.01±0.46	5.12±0.56	**1.36**±0.09	4.45±1.52
	Average	148.73	107.17	158.98	129.49	97.12	**54.89**

Table 5 Comparative results of l_2-l_0-DCAs

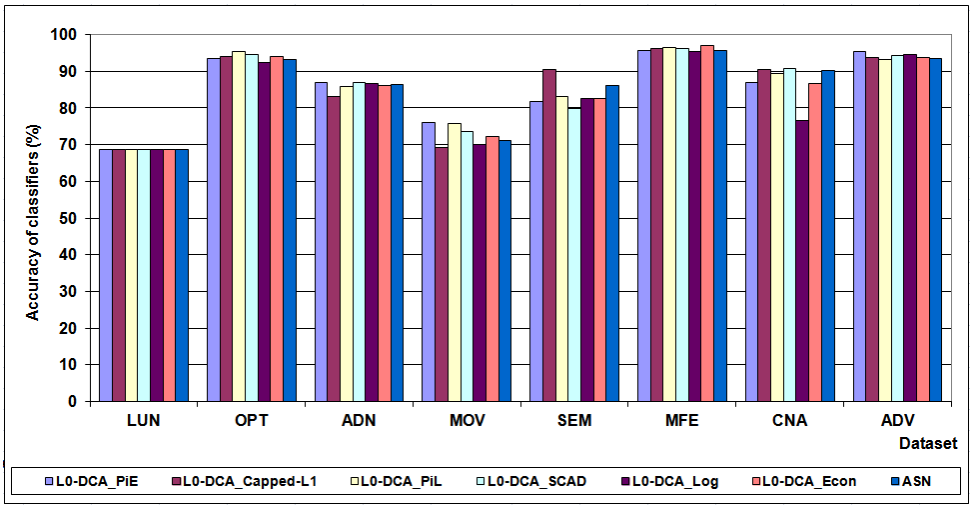


Fig. 2 Accuracy of classifiers (%) of l_0-DCAs and ASN

for the l_0 model and from 68.75% to 96.89% for the l_2-l_0 model). These results are higher than (or quite closed to) that of ASN.
- l_0-DCA_PiL and l_2-l_0-DCA_PiL give the best accuracy of classifiers. These values are, respectively, 85.94% and 86.46% and they are slightly better than that of ASN (85.61%).
- As for the comparison between the l_0 and the l_2-l_0 models: the accuracy of classifiers of l_2-l_0-DCAs are slightly better than that of l_0-DCAs.

CPU times:

- All DCA based algorithms on the l_0 model are generally faster than ASN. On average, the gain is up to 3.44 times. l_0-DCA_Log and l_0-DCA_Econ are faster than ASN on 8/8 datasets and the 4 remaining l_0-DCAs are faster than ASN on 6/8 datasets. The gain is up to 43.32 times (l_0-DCA_PiL on MOV dataset).
- l_0-DCA_Log is the fastest algorithm. On average, it takes 41.12 seconds. Whereas, the corresponding CPU time of ASN is 141.32 seconds. For the l_2-l_0 model, the l_2-l_0-DCA_Econ is the fastest algorithm (on average, it takes 54.89 seconds).
- In general, not surprisingly, the l_0-DCAs are faster than l_2-l_0-DCAs: at each iteration of l_0-DCAs we solve one linear program instead of one quadratic program in l_2-l_0-DCAs.

6 Conclusion

We have developed efficient approaches based on DC programming and DCA for feature selection in multi-class support vector machine. Based on appropriate approximation functions of zero-norm and an exact penalty technique, the l_0-MSVM and l_2-l_0-MSVM problems are reformulated as DC programs. It fortunately turns out that the corresponding DCA consist in solving, at each iteration, one linear program (in l_0 regularization) and/or one convex quadratic program (in l_2-l_0 regularization). Moreover, several DCA based algorithms converge, after a finite number of iterations, almost always to a local solution. Numerical results on several real datasets showed the robustness, the effectiveness of the DCAs based schemes. We are convinced that DCA is a promising approach for feature selection in MSVM.

References

1. Bradley, P.S., Mangasarian, O.L.: Feature selection via concave minimization and support vector machines. In: J.Shavlik, editor, Machine Learning Proceedings of the Fifteenth International Conferences (ICML'98), pp. 82–90. Morgan Kaufmann, San Francisco (1998).

2. Cai, X., Nie, F., Huang, H., Ding, C.: Multi-Class $l_{2,1}$-Norm Support Vector Machine. Data Mining (ICDM), 2011 IEEE 11th International Conference, pp. 91–100 (2011).
3. Candès, E.J., Wakin, M.B., Boyd, S.P.: Enhancing sparsity by reweighted l_1 minimization. Journal of Fourier Analysis and Applications 14, pp. 877–905 (2008).
4. Chapelle, O.: Multi-Class Feature Selection with Support Vector Machines. Technical report YR-2008-002 (2008).
5. Chen, X., Zeng, X., Alphen, D.V.: Multi-class feature selection for texture classification. Pattern Recognition Letters 27, pp. 1685–1691 (2006).
6. Chen, Y.W., Lin, C.J.: Combining SVMs with Various Feature Selection Strategies. Feature Extraction 207, Studies in Fuzziness and Soft Computing Volume, pp. 315–32.
7. Collobert, R., Sinz, F., Weston, J., Bottou, L.: Large scale transductive SVMs. J. Machine Learn. 7, pp. 1687–1712 (2006).
8. Deng, S., Xu, Y., Li, L., Li, X., He, Y.: A feature-selection algorithm based on Support Vector Machine-Multiclass for hyperspectral visible spectral analysis. Journal of Food Engineering 119, Issue 1, pp. 159–166 (2013).
9. Duan, K.B., Rajapakse, J.C, Wang, H., Azuaje, F.: Multiple SVM-RFE for Genne Selection in Cancer Classification With Expression Data. IEEE Transactions on NANOBIOSCIENCE 4, pp. 228–234 (2005)
10. Fan, J., Li, R.: Variable selection via nonconcave penalized likelihood and its Oracle Properties. Journal of the American Statistical Association 96, pp. 1348–1360 (2001)
11. Guyon, I., Elisseeff, A.: An Introduction to Variable and Feature Selection. Journal of Machine Learning Research 3, pp. 1157–1182 (2003)
12. Hermes, L., Buhmann, J.M.: Feature selection for support vector machines. Proceedings. 15th International Conference on Pattern Recognition, vol.2, pp. 712–715 (2000).
13. Hsu, C.W., Lin, C.J.: A comparison of methods for multiclass support vector machines. IEEE Transactions on Neural Networks 13:2, pp. 415–425 (2002).
14. Hui, Z.: The Adaptive Lasso and Its oracle Properties. Journal of the American Statistical Association 101:476, pp. 1418–1429 (2006)
15. Huang, J., Ma, S., Zhang, C.H.: Adaptive Lasso for sparse high-dimentional regression models. Statistica Sinica 18, pp. 1603–1618 (2008)
16. Huang, L., Zhang, H.H., Zeng, Z.B., Bushel, P.R.: Improved Sparse Multi-Class SVM and Its Application for Gene Selection in Cancer Classification. Cancer Inform 12, pp. 143–153 (2013).
17. Le Thi, H.A.: DC Programming and DCA. `http://lita.sciences.univ-metz.fr/~lethi/DCA.html`
18. Le Thi H.A., A new approximation for the ℓ_0-norm, Research Report LITA EA 3097, University of Lorraine, 2012.
19. Le Thi, H.A., Pham Dinh, T.: The DC (Difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems. Annals of Operations Research 133, pp. 23–46 (2005).
20. Le Thi, H.A., Belghiti, T., Pham Dinh, T.: A new efficient algorithm based on DC programming and DCA for Clustering. Journal of Global Optimization 37, pp. 593–608 (2006).
21. Le Thi, H.A., Le Hoai, M., Pham Dinh, T.: Optimization based DC programming and DCA for Hierarchical Clustering. European Journal of Operational Research 183, pp. 1067–1085 (2007).
22. Le Thi, H.A., Le Hoai, M., Nguyen, V.V., Pham Dinh, T.: A DC Programming approach for Feature Selection in Support Vector Machines learning. Journal of Advances in Data Analysis and Classification 2:3, pp. 259–278 (2008).
23. Le Thi, H.A., Nguyen, V.V., Ouchani, S.: Gene Selection for Cancer Classification Using DCA. In: Tang, C., Ling, C.X., Zhou, X., Cercone, N.J., Li, X. (eds.) ADMA 2008. LNCS (LNAI), vol. 5139, pp. 62–72. Springer, Heidelberg (2008).
24. Le Thi, H.A., Huynh, V.N., Pham Dinh, T.: Exact Penalty and Error Bounds in DC Programming. Journal of Global Optimization dedicated to Reiner Horst ISSN 0925-5001, DOI: 10.1007/s10898-011-9765-3 (2011).
25. Lee, Y., Kim, Y., Lee, S., Koo, J.: Structured multicategory support vector machines with analysis of variance decomposition. Biometrika 93(3), pp. 555–71 (2006)
26. Lee, Y., Lin, Y., Wahba, G.: Multicategory support vector machines, theory, and application to the classification of microarray data and satellite radiance data. Journal of the American Statistical Association 99:465, pp. 67–81 (2004).
27. Li, G.Z., Yang, J., Liu, G.P., Xue, L.: Feature Selection for Multi-Class Problems Using Support Vector Machines. PRICAI 2004: Trends in Artificial Intelligence, Lecture Notes in Computer Science 3157, pp. 292–300, Springer Berlin Heidelberg (2004).
28. Liu, Y., Shen, X.: Multicategory Ψ-Learning. Journal of the American Statistical Association 101:474, pp. 500–509, (2006).
29. Liu, Y., Zhang, H.H., Park, C., Ahn, J.: Support vector machines with adaptive L_q penalty. Computational Statistics & Data Analysis 51, pp. 6380–6394 (2007).
30. Liu, D., Qian, H., Dai, G., Zhang, Z.: An iterative SVM approach to feature selection and classification in high-dimensional datasets. Pattern Recognition 46, Issue 9, pp. 2531–2537 (2013).
31. Maldonado, S., Weber, R., Basak, J.: Simultaneous feature selection and classification using kernel-penalized support vector machines. Information Sciences 181, Issue 1, pp. 115–128 (2011).
32. Neumann, J., Schnörr, C., Steidl, G.: SVM-based Feature Selection by Direct Objective Minimisation. In: Proc. of 26th DAGM Symposium Pattern Recognition, pp. 212–219 (2004)

33. Ong, C.S., Le Thi, H.A.: Learning sparse classifiers with Difference of Convex functions Algorithms. Optimization Methods and Software 28:4 (2013).
34. Peleg, D. and Meir, R.: A bilinear formulation for vector sparsity optimization. Signal Processing, 8(2), 375–389 (2008).
35. Pham Dinh, T., Le Thi, H.A.: Convex analysis approach to d.c. programming: Theory, Algorithm and Applications. Acta Mathematica Vietnamica 22, pp. 289–355 (1997).
36. Pham Dinh, T., Le Thi, H.A.: Optimization algorithms for solving the trust region subproblem. SIAMJ. Optimization 2, pp. 476–505 (1998).
37. T. Pham Dinh, H.A. Le Thi, Recent advances on DC programming and DCA. Transactions on Computational Intelligence XIII, Lecture Notes in Computer Science Volume 8342, 2014, pp 1-37.
38. A. Rakotomamonjy, Variable Selection Using SVM-based Criteria, Journal of Machine Learning Research, Vol. 3:1357–1370 (2003).
39. Ramona, M., Richard, G., David, B.: Multiclass Feature Selection With Kernel Gram-Matrix-Based Criteria. IEEE Transactions on Neural Networks and Learning Systems 23:10, pp. 1611–1623 (2012).
40. Ronan, C., Fabian, S., Jason, W., Lé, B.: Trading Convexity for Scalability. Proceedings of the 23rd international conference on Machine learning ICML 2006, Pittsburgh, Pennsylvania, pp. 201–208 (2006).
41. Yeh, Y., Chung, Y., Lin, T., Wang, Y.: Group lasso regularized multiple kernel learning for heterogeneous feature selection. The 2011 International Joint Conference on Neural Networks (IJCNN), pp. 2570–2577 (2011).
42. Wang, H., Li, G., Jiang, G.: Robust regression shrinkage and consistent variable selection via the LAD-LASSO. Journal of Business & Economics Statistics 25:3, pp. 347–355 (2007).
43. Wang, L., Shen, X.: On l_1-norm multi-class support vector machine: methodology and theory. Journal of the American Statistical Association 102, pp. 583–594 (2003)
44. Weston, J., Watkins, C.: Support Vector Machines for Multi-Class Pattern Recognition. In: Proceedings - European Symposium on Artificial Neural Networks, ESANN 1999, pp. 219–224. D-Facto public (1999).
45. Weston, J., Elisseeff, A., Schölkopf, B.: Use of Zero-Norm with Linear Models and Kernel Methods. Journal of Machine Learning Research 3, pp. 1439–1461 (2003).
46. Wu, K., Lu, B., Uchiyama, M., Isahara, H.: A Probabilistic Approach to Feature Selection for Multi-class Text Categorization. D. Liu et al. (Eds.): ISNN 2007, Part I, LNCS 4491, pp. 1310–1317 (2007).
47. Zhang, H.H., Liu, Y., Wu, Y., Zhu, J.: Variable selection for the multicategory SVM via adaptive sup-norm regularization. Journal of Statistics 2, pp. 149–167 (2008).
48. Zhou, X., Tuck, D.P.: MSVM-RFE: extensions of SVM-RFE for multiclass gene selection on DNA microarray data. Bioinformatics 23:9, pp. 1106–1114 (2007).
49. Zhou, Y., Jin, R., Hoi, S.C.: Exclusive Lasso for Multi-task Feature Selection. In AISTATS 9 (2010).

Conclusion Générale

Cette thèse est consacrée au développement d'une approche innovante de l'optimisation non-convex: la programmation DC (Difference of Convex functions) et DCA (DC Algorithms), pour la résolution de certaines classes de problèmes en Machine Learning et Data Mining.

L'utilisation avec succès de DCA, pour la première fois, dans les deux thématiques importantes d'apprentissage non supervisée - la maximisation du critère de Modularité et la cartes auto-organisatrice (SOM) représente un grand intérêt pour la communauté des chercheurs / practiciens en Machine Learning. Grâce aux outils théoriques rigoureux, nous obtenons des schémas DCA simples, faciles à implémenter qui sont surtout non coûteux en temps de calcul et donc capables de traiter des problèmes de grande dimension. Les résultats numériques ont montré l'efficacité, la scalabilité et la supériorité de DCA par rapport aux méthodes standard.

La maximisation du critère de Modularité: est un problème d'optimisation non-convexe et NP-difficile. La première contribution pour ce problème porte sur le développement d'un algorithme basé sur la programmation DC et DCA. Le problème non convexe est reformulé en un programme DC pour lequel nous proposons une décomposition DC appropriée et un algorithme DC efficace pour le résoudre. Cet algorithme permet de partitionner de grands réseaux (plus de 4 millions de sommets et plus de 30 millions d'arêtes) en communautés disjointes dont tous les calculs sont explicites. Notre algorithme ne nécessite pas de raffinement et converge vers une bonne solution après un nombre fini d'itérations. Par ailleurs, le nombre de clusters est automatiquement détecté lors des itérations de DCA. Cette originalité de DCA constitue une contribution importante et intéressante de notre approche. Une seconde contribution pour ce problème porte sur le développement d'un nouvel algorithme d'analyse hiérarchique de la structure d'un réseau. Cet algorithme, Mod-Müllner, permet de trouver la même solution que les algorithmes hiérarchiques déjà existants avec des temps de calcul beaucoup plus courts. Un réseau de plus de 50 millions d'arêtes a ainsi pu être décomposé en moins d'une minute.

L'apprentissage du SOM a été considéré comme un problème d'optimisation avec une fonction d'énergie non smooth et non convexe. La programmation DC et DCA sont étudiés pour résoudre efficacement ce problème. Outre cette première version de l'algorithme DCA, une variante, basée sur une gestion efficace de la décroissance de la température du modèle, est aussi proposée. Dans les deux cas, nous avons obtenu des algorithmes simples et efficaces car ils ne nécessitent que des opérations simples (sommes et produits de matrices). Grâce à l'efficacité de DCA, notre algorithme d'apprentissage nécessite un nombre très faible d'étapes, alors que dans les méthodes classiques ce nombre doit être très grande pour obtenir une bonne solution. Les résultats numériques montrent que l'approche proposée donne de meilleurs résultats que la méthode standard (Batch SOM).

Ces travaux sucsitent des sujets qui peuvent être envisagés dans une future proche:

Pour le problème de la détection de communautés dans des réseaux complexes: Il nous semble plus pertinent d'utiliser des mesures bien adaptées à la structure du réseau, par la suite il est intéressant d'investir DCA aux problèmes de maximisation des nouvelles mesures. Citons, par exemple, "distance based modularity" proposé par Wangqun Lin et al. en 2012 pour des réseaux d'information incomplète dans le sens où les graphes ont un nombre faible des arrêttes, la modularité densitée proposée par Li et al. en 2008 pour les

réseaux ayant des petites communautés (les méthodes basées sur la mesure de modularité ont la tendance à fusionner de les petites communautés à une plus grande), ou encore la mesure du modularité adaptée à des graphes bipartis de Michael J. Barber en 2007.

Pour SOM: Nous souhaitons développer DCA pour les versions du noyau de BSOM et/ou pour minimiser la fonction d'énergie dans laquelle la fonction de voisinage est modifiée à chaque étape t.

Par ailleurs l'efficacité de DCA pour ces deux thématiques d'apprentissage non supervisée nous encourage à appliquer nos algorithmes aux problèmes concrets.

Si DCA est utilisé pour la première fois dans les deux précédents problèmes, il n'en est pas de même pour la sélection des variables en classification supervisée: DCA a été développé pour la sélection des variables en SVM utilisant la norme zéro dans certains articles. Notre façon de traiter la norme zéro est la même que celle dans ces travaux, mais nos problèmes sont bien plus complexes.

Pour la sélection de variables en apprentissage semi-supervisé dans les machines à vecteurs de support (S3VM): les problèmes d'optimisation sont non smooth et non convexes. Au point de vue algorithmique, la sélection des variables en S3VM est plus difficile que la sélection des variables en SVM parce qu'il y a la double difficulté venant d'une part de la norme zéro et d'autre part de la non convexité et non différentiabilité de la fonction de perte. Pour résoudre ce problème, nous utilisons 5 différentes approximations et une reformulation continue de la norme l_0 ce qui conduit à 6 problèmes non convexes. Ensuite, nous avons développé 6 schémas DCA efficaces pour les résoudre. Les résultats numériques sur plusieurs jeux de données réels ont montré la robustesse, l'efficacité de nos méthodes, aussi bien en classification qu'en sélection de variables.

Pour la sélection de variables en MSVM, nous avons considéré le modèle de SVM multi-classes proposé par J. Weston et C. Watkins. Par rapport à la sélection de variables en SVM, sa difficulté réside plutôt dans l'aspect numérique (et non en construction des schémas DCA) car la taille du problème devient beaucoup plus importante en MSVM. De manière similaire au problème de S3VM ci-dessus, nous avons développé les deux approches non convexes basées sur la programmation DC et DCA: l'approximation et la reformulation via la pénalité exacte. Les expériences numériques menées sur des données réelles, là encore, montre l'efficacité de nos méthodes par rapport au meilleur algorithme existent (Adaptive Sub-Norm). En effet, les modèles obtenus sont plus parcimonieux et obtenus avec meilleurs temps de calcul.

Suite à ces travaux, nous envisageons des études suivantes dans les prochains jours:

Pour MSVM: Les algorithmes basés sur DCA donnent des bons résultats sur des données linéairement séparables. Dans un travail futur, la mise au point d'un algorithme DC dans le cas où les données sont non linéairement séparables est nécessaire. De manière classique, une fonction noyau sera utilisée afin de séparer les données dans un espace de grande dimension, et un DCA sera proposé pour résoudre efficacement ce problème.

De plus, la sélection de groupes de variables parcimonieux (Sparse Group Feature Selection) est un problème intéressant. Le modèle de "Group Lasso" ne permet pas d'obtenir une représentation parcimonieuse dans les groupes. Par conséquent, nous souhaitons explorer le modèle de "Sparse Group Lasso" qui produit parcimonie tant au niveau de groupe de variables et au niveau de l'individuels dans des groupes. Basé sur la norme l_0, nous souhaitons développer des DCA pour ce problème.

Pour S3VM: Nous souhaitons utiliser nos algorithmes dans les applications concrètent, en particulier en apprentissage des données textuelles.