AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

THÈSE

en vue de l'obtention du titre de

DOCTEUR DE L'UNIVERSITÉ DE LORRAINE

(arrêté ministériel du 25 mai 2016)

Spécialité Informatique

présentée par

Sara SAMIR

Titre de la thèse :

Approches coopératives pour certaines classes de problèmes d'optimisation non convexe : Algorithmes parallèles / distribués et applications

—

Cooperative approaches for some classes of nonconvex optimization problems: parallel / distributed algorithms and applications

soutenue le 06 05 2020

Composition du Jury :

Rapporteurs	Patrick SIARRY	*Professeur, Université Paris-Est Créteil*
	Van Dat CUNG	*Professeur, Institut Polytechnique de Grenoble*
Examinateurs	Ammar OULAMARA	*Professeur, Université de Lorraine*
	Mohammed YAGOUNI	*Maître de Conférences, USTHB*
	Adnan YASSINE	*Professeur, Université Le Havre Normandie*
Directrice de thèse	Hoai An LE THI	*Professeur, Université de Lorraine*

Thèse préparée au sein du laboratoire d'Informatique Théorique et Appliquée (LITA) et du département informatique & applications, LGIPM, Université de Lorraine, Metz, France

Je dédie cette thèse à mes chers parents.
À mon père aimé qui m'a inculqué une âme passionnée par la science et à ma mère, cette femme formidable qui me couvre toujours de sa tendresse.

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de ma formation doctorale et qui m'ont aidée tout au long de mon parcours.

Je voudrais tout d'abord adresser toute ma reconnaissance à Madame LE Thi Hoai An, Professeur à l'Université de Lorraine et ma Directrice de thèse, pour la confiance qu'elle m'a accordée en acceptant de diriger mes travaux doctoraux. Je tiens vraiment à la remercier pour son aide inestimable, ses conseils précieux, son soutien permanent, et pour sa disponibilité pour répondre à mes questions.

Mes remerciements vont également à Monsieur Patrick SIARRY, Professeur à l'Université de Paris-Est Créteil ainsi que Monsieur Van Dat CUNG, Professeur à l'Institut Polytechnique de Grenoble pour l'honneur qu'ils m'ont fait d'être rapporteurs de ma thèse et de m'avoir accordé de leur temps précieux pour lire et évaluer mes travaux.

Je tiens aussi à remercier Monsieur Ammar OULAMARA, Monsieur Mohammed YAGOUNI qui n'a pas cessé de m'encourager depuis ma Licence, et Monsieur Adnan YASSINE d'avoir accepté d'être membre du jury.

Je remercie vivement toutes les personnes du laboratoire LGIPM, en particulier tous les membres du Département d'Informatique et Application (EX. LITA).

Je tiens à témoigner toute ma reconnaissance aux personnes suivantes, pour leur aide dans la réalisation de cette thèse :

Mes chers parents, mes sœurs, mes frères ainsi que mes nièces et mes neveux.

Tous mes amis qui m'ont apporté leur soutien moral et intellectuel tout au long de ma démarche.

Contents

Résumé **6**

Introduction générale **8**

1 Methodologies **15**

1.1 Cooperative schemes 16

1.1.1 Parallel and distributed programming 16

1.1.2 Generic scheme of a cooperative approach 21

1.2 Component Algorithms 22

1.2.1 DC programming and DCA 24

1.2.2 Metaheuristics 33

2 Cooperative methods for solving MBLP **41**

2.1 Introduction 42

2.2 DCA and VNS for solving MBLP 43

2.2.1 DCA for solving MBLP 44

2.2.2 VNS for solving MBLP 45

2.3 VNS-DCA: a cooperative approach for solving MBLP 45

2.4 Applications 46

2.4.1 CFLP 48

2.4.2 A constructive heuristic for VNS 49

2.4.3 Adapting DC programming and DCA to CFLP 49

2.4.4 Adapting VNS to CFLP 51

2.4.5 Adapting the cooperative approach to CFLP 51

2.5 Numerical results 52

2.6 Conclusion 56

3 Cooperative methods for BQP **57**
3.1 Introduction . . . 58
3.2 DCA-like, GA and MBO for solving BQP . . . 59
3.2.1 A DCA-like scheme for BQP . . . 59
3.2.2 GA and MBO schemes for BQP . . . 61
3.3 DCA$_l$-Meta: a cooperative approach . . . 62
3.3.1 Different steps of DCA$_l$-Meta . . . 62
3.4 Application of DCA$_l$-Meta to the QAP . . . 65
3.4.1 Elaborating the solution method to QAP . . . 65
3.4.2 Stopping conditions . . . 65
3.5 Numerical results . . . 66
3.5.1 Comparative results . . . 66
3.5.2 Study of the behaviors of the component algorithms . . . 67
3.6 Conclusion . . . 75

4 Cooperative methods for clustering **76**
4.1 Introduction . . . 77
4.2 Application of the component algorithms to the MSSC . . . 79
4.3 Cooperative schemes . . . 80
4.4 Numerical results . . . 82
4.4.1 Comparison . . . 82
4.4.2 Study of the cooperations . . . 86
4.5 Conclusion . . . 93

Conclusion and perspectives **94**

Abbreviations and notations

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

BQP	Binary Quadratic Program
CFLP	Capacitated Facility Location Problem
DC	Difference of Convex functions
DCA	DC Algorithm
GA	Genetic Algorithm
MBLP	Mixed Binary Linear Program
MBO	Migrating Bird Optimization
MSSC	Minimum-Sum-of-Squares Clustering problem
QAP	Quadratic Assignment Problem
TS	Tabu Search
VNS	Variable Neighborhood Search

$\mathbb{R}$	the set of real numbers
$\mathbb{R}^n$	the set of real column vectors of size n
$\overline{\mathbb{R}}$	the set of extended real numbers, $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$
$\|\cdot\|$	the Euclidean norm (or ℓ_2-norm), $\|x\| = (\sum_{i=1}^{n} \|x_i\|^2)^{1/2}$, $x \in \mathbb{R}^n$
$\langle \cdot, \cdot \rangle$	the scalar product, $\langle x, y \rangle = \sum_{i=1}^{n} x_i.y_i$, $x, y \in \mathbb{R}^n$
$\chi_C(\cdot)$	the indicator function of a set C, $\chi_C(x) = 0$ if $x \in C$, $+\infty$ otherwise
co$\{C\}$	the convex hull of a set of points C
dom f	the effective domain of a function f
$\nabla f(x)$	the gradient of a function f at x
$\partial f(x)$	the subdifferential of a function f at x

Résumé

Dans cette thèse, nous nous intéressons au développement des approches coopératives pour la résolution de certaines classes de problèmes d'optimisation non convexe qui jouent un rôle très important de par leurs applications dans de nombreux domaines. Il s'agit de combiner plusieurs algorithmes connus sous les noms des algorithmes composants (participants). La combinaison est basée principalement sur la programmation DC (*Difference of Convex Functions*) et DCA (*DC Algorithm*) avec des métaheuristiques. Pour la conception des logiciels nous utilisons les paradigmes de la programmation parallèle et distribuée. Chaque processus s'occupe d'un algorithme et communique avec les autres en appelant les fonctions de la bibliothèque MPI (*Message Passing Interface*) qui est un protocole de communication en programmation parallèle et distribuée.

Outre l'introduction et la conclusion, la thèse est composée de quatre chapitres. Le chapitre 1 concerne les outils théoriques et algorithmiques comme servant de base méthodologique aux chapitres suivants. Le chapitre 2 s'articule autour les problèmes linéaires à variables mixtes binaires. Pour la résolution de ces problèmes, nous proposons une approche coopérative entre DCA et VNS (*Variable Neighborhood Search*). Puisque le schéma est constitué de deux algorithmes, nous optons pour la communication point à point entre les processus. Nous adaptons notre schéma pour résoudre le problème de localisation de l'installation avec des contraintes de capacités.

Dans le chapitre 3, nous étudions la programmation quadratique à variables binaires. Nous développons une coopération entre DCA-Like (une nouvelle version de DCA) et deux autres métaheuristiques : GA (*Genetic Algorithm*) et MBO (*Migrating Birds Optimization*). Pour la communication entre les processus, nous utilisons la communication collective. Plus précisément une fonction qui permet la diffusion simultanée l'information d'un processus à tous les autres. Cette approche est adaptée et appliquée au problème d'affectation quadratique.

Dans le chapitre 4, nous résolvons les problèmes de "*clustering*" via la minimisation de la somme des carrés par deux approches coopératives. La première consiste à combiner le DCA avec VNS et TS (*Tabu Search*). Quant à la deuxième, elle utilise la MBO avec les trois derniers algorithmes précités. Dans ces deux approches, nous utilisons une fonction de communication qui permet au processus d'accéder aux mémoires des autres et y enregistrer son information sans un temps d'attente.

Mots-clés : Optimisation non convexe, approches coopératives, programmation parallèle et distribuée, programmation DC et DCA, métaheuristiques.

Abstract

In this thesis, we are interested in developing new cooperative approaches for solving some classes of nonconvex problems which play a very important role to model real-world problems. To design the schemes of our approaches, we combine several algorithms which we call the component (participant) algorithms. The combination is mainly based on DC (Difference of Convex Functions) and DCA (DC Algorithm) with metaheuristics. To develop our solution methods, we use the paradigm of parallel and distributed programming. Therefore, each process deals with an algorithm and communicates with the others by calling the functions of the MPI (Message Passing Interface) library which is a communication protocol in parallel and distributed programming.

Besides the introduction and conclusion, this thesis is composed of four chapters. Chapter 1 concerns the theoretical and algorithmic tools serving as a methodological basis for the following chapters. Chapter 2 is about the mixed binary linear programs. To solve these problems, we propose a cooperative approach between DCA and VNS (Variable Neighborhood Search). Since the scheme is constituted by two algorithms, we use the point to point communication between the processes. As an application, we adapt our scheme to solve the capacitated facility location problem.

Concerning chapter 3, we study the class of binary quadratic problems. Regarding the solution methods, we develop a cooperation between DCA-like which is a new version of DCA and two other metaheuristics: GA (Genetic Algorithm) and MBO (Migrating Birds Optimization). The exchange of information between the processes is expressed by using collective communication's function. More precisely, we call a function which allows broadcasting information of a process to all the others at the same time. This cooperative approach is adapted to the quadratic assignment problem.

In chapter 4, we solve the MSSC (Minimum-Sum-of-Squares Clustering) using two cooperative approaches. The first combines DCA, VNS, and TS (Tabu Search). As for the second, it combines the MBO with the other three algorithms cited before. In these two approaches, we use a function of communication that allows a process to access the memories of the others and save the information there without blocking the work of the receiving processes.

Keywords: nonconvex optimization, cooperative approaches, parallel and distributed programming, DC programming and DCA, metaheuristics.

Introduction générale

Cadre général et motivations

L'optimisation est une branche des mathématiques qui a comme premier objectif la modélisation des problèmes dans divers domaines, sous forme de problèmes mathématiques. Ces derniers sont exprimés par une fonction objectif à maximiser ou à minimiser sous un ensemble des contraintes à respecter. Le deuxième objectif consiste à résoudre ces problèmes en utilisant les outils des mathématiques et de l'informatique. Plus précisément, il s'agit de concevoir des algorithmes efficaces et puissants afin de trouver la solution optimale ou sinon la meilleure solution d'un problème mathématique en un minimum de temps. En Mathématiques Appliquées-Informatique, il est sans conteste que l'optimisation non convexe (différentiable /non différentiable) et l'optimisation globale connaissent, au cours de ces deux dernières décennies, des développements spectaculaires partout dans le monde sur les plans théorique et pratique. L'optimisation non convexe joue un rôle très important pour la modélisation des problèmes du monde réel du fait que la plupart d'entre eux sont de nature non convexe.

Dans cette thèse, nous nous intéressons principalement aux deux classes de problèmes. D'abord, nous étudions les problèmes d'optimisation combinatoire qui consistent à trouver l'élément (ou les éléments) qui maximise ou minimise une fonction objectif. Un problème d'optimisation combinatoire peut être décrit par la formulation mathématique suivante:

$$\min / \max\{f(x) : x \in S\}, \tag{1}$$

sachant que S est un ensemble fini.
Quant à la deuxième classe, elle comprend le problème du *clustering*. Le *clustering* est une méthode fondamentale de l'apprentissage non supervisé qui consiste à partitionner un ensemble de données en *clusters* (groupes) de telle façon que les données de chaque *cluster* aient des propriétés similaires.
L'absence de convexité, dans ces classes de problèmes, crée une source de difficultés de toutes sortes, en particulier la distinction entre les minima locaux et globaux, la non-existence de la caractérisation vérifiable des solutions optimales, etc. La recherche d'une solution globale d'un programme non convexe est une quête du Graal pour des optimiseurs. Par ailleurs, les chercheurs et praticiens sont de plus en plus motivés et concernés par les techniques de résolution des programmes non convexes de très

grandes dimensions dans divers domaines, comme par exemple en transport, logistique, systèmes de communication, apprentissage et fouille de données. Les ordinateurs, ces outils indispensables pour mettre en œuvre toutes techniques de résolution, n'ont pas cessé de progresser depuis leur invention. En effet, des ordinateurs multiprocesseurs et multicœurs sont apparus, ce qui a donné naissance aux paradigmes de la programmation parallèle et distribuée. Cette dernière consiste à concevoir un programme informatique en faisant appel à plusieurs processus ou *threads* (tout dépend de la bibliothèque utilisée) qui s'exécutent en parallèle. Chaque processus (*thread*) traite une partie du programme ou aussi le programme entier d'une façon indépendante des autres processus utilisés. L'avantage de la programmation parallèle et distribuée est l'optimisation du temps de calcul, l'augmentation de la chance de l'obtention d'une solution efficace, ainsi que l'amélioration de la qualité de cette dernière. L'optimisation du temps de calcul est le résultat du partage des tâches entre les processus qui travaillent en parallèle. Quant au deuxième avantage, il est basé sur le fait d'adopter plusieurs algorithmes, chacun est exécuté sur un processus différent. En effet, quel que soit le modèle parallèle utilisé, le nombre d'algorithmes participants doit être au moins égal à celui des processus appelés. Lors de la résolution d'un problème mathématique, si un algorithme n'est pas efficace pour une instance du problème, on a un autre algorithme qui peut être meilleur que le premier ou encore s'entraide avec celui-ci pour l'amélioration de la qualité de la solution. L'utilisation de plusieurs algorithmes sur plusieurs processus en parallèle d'une façon distribuée est à l'origine de l'avènement des approches parallèles, coopératives et collaboratives. Alors la question qui se pose maintenant, *la coopération entre des algorithmes, de quoi s'agit-il*?

Les approches coopératives qui font l'objet principal de cette thèse, consistent à adopter plusieurs algorithmes appelés les algorithmes composants pour résoudre un problème donné. Ces derniers se lancent en même temps et s'exécutent en parallèle. Ils coopèrent entre eux en échangeant des informations (la solution, la valeur de la fonction objectif, le critère d'arrêt, etc.) et/ou en partageant les tâches (c'est-à-dire chaque algorithme s'occupe d'une partie du problème). Donc, l'idée d'une approche coopérative consiste à exploiter et combiner les points forts ainsi que les points faibles des algorithmes composants pour concevoir des approches puissantes et efficaces. Cette technique, en échangeant des solutions, permet aux algorithmes composants de changer de comportement lors de l'exploration de l'espace de recherche, ce qui va surement mener vers une amélioration de la qualité de la solution. L'implémentation de ces approches est effectuée en utilisant les techniques de la programmation parallèle et distribuée.

En ce qui concerne les méthodes de résolution des problèmes d'optimisation non convexe, abordées dans ce manuscrit, nous étudions une autre technique de combinaison des algorithmes. Nos méthodes s'appuient sur les approches coopératives basées sur la programmation DC (Difference of Convex Functions) et DCA (DC Algorithm) ainsi que les métaheuristiques. Nous effectuons plusieurs expériences numériques qui montrent l'efficacité et la performance des approches coopératives, en les comparant avec les autres méthodes existantes dans la littérature. Cette démarche est motivée d'une part par la puissance de DCA qui est une approche innovante d'optimisation non convexe, d'autre part, par le fait que les métaheuristiques constituent un recours intéressant pour certaines classes de problèmes et plus spécialement dans le domaine de l'optimisation combinatoire. L'amélioration et l'optimisation de la puissance de

calcul ainsi que la rapidité qui s'obtiennent en utilisant la programmation parallèle et distribuée, sont d'autres raisons qui nous ont motivés pour adopter cette technique dans nos travaux.
La programmation DC et DCA sont considérés comme l'épine dorsale de la programmation non convexe et l'optimisation globale compte tenu des applications réussies de cette théorie dans de nombreux domaines des sciences appliquées (la liste des références est dans Le Thi and Pham Dinh (2018)). En particulier, l'optimisation combinatoire (voir, par exemple, Le Thi et al. (2009, 2015); Le Thi (2019a); Le Thi et al. (2014e,d); Le Thi and Pham Dinh (2008); Chu (2017); Le Thi and Pham Dinh (2001, 2018); Le Thi et al. (2014f, 2007d); Pham Dinh et al. (2008, 2010); Quang Thuan and Le Thi (2011)) ainsi que le problème de *clustering* dont on peut citer les références suivantes, Le Hoai et al. (2013a, 2006, 2013b); Le Thi et al. (2007a, 2014a, 2008, 2007b, 2014c); Le Thi and Pham Dinh (2009); Thuy and Le Thi (2015); Thuy et al. (2013b). En programmation DC et DCA, on considère le problème décrit comme suit Pham Dinh and Le Thi (1997a):

$$\min \{f(x) = g(x) - h(x) : x \in \mathbb{R}^n\}, \tag{2}$$

où les fonctions g et h sont convexes définies sur $\mathbb{R}^n$ et à valeurs dans $\mathbb{R} \cup \{+\infty\}$, semicontinues inférieurement et propres. La fonction f est appelée fonction DC avec les composantes DC g et h, et $g-h$ est une décomposition DC de f. DCA est basé sur la dualité DC et des conditions d'optimalité locale. La construction de DCA implique les composantes DC g et h, et non la fonction DC f en elle-même. Or chaque fonction DC admet une infinité de décompositions DC qui influencent considérablement sur la qualité (la rapidité, l'efficacité, la globalité de la solution obtenue, etc.) 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.
Le choix de la programmation DC et DCA dans cette thèse est motivé par les arguments suivants (Pham Dinh and Le Thi (2014)) :

- L'efficacité de la programmation DC et DCA a été prouvée par les applications réussies de cette théorie pour la résolution de nombreux problèmes d'optimisation non convexe dans divers domaines (une liste de références est donnée dans Le Thi (2005); Le Thi and Pham Dinh (2018)). En particulier, les problèmes d'optimisation combinatoire et le problème du *clustering*.
- DCA est une philosophie plutôt qu'un algorithme. Pour chaque problème, nous pouvons concevoir une famille d'algorithmes basés sur DCA. La fexibilité de DCA sur le choix de la 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.

Les métaheuristiques sont des algorithmes génériques proposés pour la résolution des problèmes d'optimisation. Elles sont généralement inspirées des phénomènes de la nature. Plus précisément, il existe des métaheuristiques développées par des analogies

avec la physique comme le recuit simulé Merendino and Celebi (2013); Nwana et al. (2005); Selim and Alsultan (1991). D'autres sont inspirées de la biologie comme les algorithmes génétiques Maulik and Bandyopadhyay (2000); Misevicius and Staneviciene (2018); Potvin (1996), ou encore par l'éthologie comme les oiseaux migrateurs Duman and Elikucuk (2013); Duman et al. (2011, 2012), la colonie de fourmis Dorigo and Di Caro (1999); Gambardella et al. (1999); Shelokar et al. (2004), le comportement des éléphants Jaiprakash and Nanda (2019), etc. Les approches métaheuristiques sont caractérisées par deux principes très importants : l'intensification et la diversification. L'intensification consiste à exploiter les régions prometteuses dans l'espace de la recherche afin d'améliorer la qualité de la solution et faire tendre cette dernière vers un optimum local dans la région parcourue. Quant à la diversification, il s'agit d'orienter la recherche pour explorer de nouvelles régions dans l'espace de recherche afin de trouver une solution différente qui ne soit pas forcément un optimum local dans la région où elle est présente.

Dans nos travaux, nous utilisons d'abord la recherche à voisinage variable ou bien VNS (*Variable Neighborhood Search*) qui se base sur une structure de voisinage et exploite l'idée du changement systématique du voisinage. La VNS a été appliquée, avec succès, à plusieurs problèmes d'optimisation combinatoire (voir, Bercachi (2010); Fleszar and Hindi (2004); Hansen and Mladenović (2001); Lazić (2010); Mladenović and Hansen (1997); Salehipour et al. (2011)). Ensuite, nous appliquons les algorithmes génétiques et la MBO (*Migrating Birds Optimization*). Les algorithmes génétiques sont des algorithmes évolutifs inspirés du modèle biologique de l'évolution naturelle des espèces. Ils consistent à générer une population d'individus dont chacun possède des caractéristiques différentes et il représente une solution au problème en question. Ceux-ci s'appuient sur des opérateurs (la sélection, le croisement, la mutation, etc.) ainsi que des probabilités. En optimisation, on peut citer plusieurs articles qui montrent les résultats excellents obtenus par l'application des algorithmes génétiques, par exemple, Julstrom (2005); Maulik and Bandyopadhyay (2000); Merz and Freisleben (1999); Misevicius and Staneviciene (2018); Potvin (1996); Smith (2002). En ce qui concerne la MBO, elle a été inspirée de la formation en V et le comportement des oiseaux migrateurs. Cette métaheuristique a prouvé son efficacité pour la résolution des problèmes d'optimisation (voir, par exemple, Duman and Elikucuk (2013); Duman et al. (2011, 2012); Soto et al. (2016); Tongur and Ülker (2014)). La dernière métaheuristique que nous étudions est nommée TS (*Tabu Search*). C'est une méthode de recherche locale qui consiste à interdire des déplacements en utilisant une liste tabou dans la mémoire, et à permettre d'autres déplacements vers des solutions qui n'améliorent pas la qualité dans le but de s'extraire d'un minimum local. Le choix de cette méthode est motivé par son application dans divers domaines tels que le transport et l'ordonnancement (voir, Al-Sultan (1995); Glover (1986); Glover and Laguna (1998)).

L'utilisation des techniques de la programmation parallèle et distribuée est justifiée par le fait que cette dernière permet d'améliorer la puissance de calcul. Le traitement parallèle et le partage des tâches conduisent à une réduction du temps d'exécution. Le parallélisme consiste à effectuer un calcul en lançant plusieurs processus pour résoudre un problème donné. Le calcul s'effectue en partageant les tâches sur les processeurs.

Plus précisément, chacun traite un sous problème voire tout le problème en communiquant avec les autres pour l'échange des informations.

Nos contributions

Cette thèse a pour objectif de concevoir de nouvelles approches pour la résolution des problèmes d'optimisation non convexe cités plus haut. Les approches que nous proposons, consistent à exploiter la puissance de DCA et la diversité des métaheuristiques pour développer une façon différente inspirée du *Brainstorming*, de combinaison de ces algorithmes. C'est-à-dire opter plutôt pour des schémas orientés vers la coopération entre différents algorithmes. Pour la mise en œuvre de nos approches, nous adaptons les paradigmes de la programmation parallèle et distribuée. Cette manière de faire nous amène divers schémas collaboratifs et coopératifs similaires au *Metastorming* Yagouni and Le Thi (2014) qui se base sur des métaheuristiques. Dans cette thèse, nous résolvons trois classes différentes de problèmes en développant des approches coopératives distinctes et spécifiques pour chaque classe. La communication entre les algorithmes composants est exprimée par des fonctions de la bibliothèque MPI (*Message Passing Interface*) qui est un protocole de communication adapté en programmation parallèle et distribuée. Les fonctions de MPI que nous utilisons diffèrent d'une approche à une autre. Selon les fonctions que nous appelons dans nos travaux, nous pouvons classer les modèles de communication que nous adaptons en trois types. Le premier représente la communication point à point en deux sens (*point-to-point two-sided communication*) et est appliqué sur la première classe de problème. Le second modèle est utilisé lors de la résolution du deuxième problème et est connu par la communication collective à sens unique (*collective one-sided communication*). Tandis que le troisième est nommé par la communication point à point à sens unique (*point-to-point one-sided communication*). Dans ce qui suit, nous donnons quelques détails sur les principales réalisations mentionnées dans ce manuscrit.

La première classe de problèmes que nous étudions concerne les problèmes linéaires à variables mixtes binaires (*the mixed binary linear problems* (MBLP) en anglais)). Nous utilisons une formulation DC pour ces problèmes en nous servant de la fonction de pénalité exacte Le Thi and Pham Dinh (2001). Cette dernière peut causer l'obtention d'une solution non réalisable (non binaire). Pour éviter ce cas, nous optons pour une mise à jour de la variable de pénalité dans le schéma de DCA. D'autre part, nous adaptons à ce problème, la VNS pour laquelle nous choisissons soigneusement une structure de voisinage ainsi qu'une procédure de recherche locale. Nous proposons une approche coopérative à base de DCA et de VNS qui sont appelés les algorithmes composants, en utilisant le modèle du *master-slave* de la programmation parallèle et distribuée. Le concept général de cette approche consiste à lancer le DCA et VNS en parallèle, simultanément, et chacun sur un processus différent. Chaque processus exécute un certain nombre d'itérations de son algorithme puis envoie son résultat aux autres processus via la fonction MPI_Send de la bibliothèque MPI. Quant à la réception des messages, elle se fait par le biais de la fonction MPI_Recv, donc une communication point à point à sens unique est adaptée dans cette approche. Le processus master effectue une évaluation et envoie la meilleure solution trouvée (c'est-à-dire celle qui

donne la meilleure valeur de la fonction objectif) pour qu'elle soit utilisée comme un point de départ pour le prochain cycle. L'approche proposée dans ce chapitre ainsi que les algorithmes composants sont adaptés au problème de localisation de l'installation avec contraintes de capacité (CFLP, une abréviation de *the Capacitated Facility Location Problem*, en anglais). Les expériences numériques sur des données réelles du CFLP, prouvent l'efficacité de cette approche coopérative en la comparant avec quelques méthodes standard.

Dans le chapitre 3, nous traitons un problème d'optimisation combinatoire plus général que le premier. Il s'agit des problèmes quadratiques à variables binaires (en anglais, *the binary quadratic problems* (BQP). Nous proposons une nouvelle coopération fondée sur une version récente de DCA qui est nommée DCA-Like, et deux métaheuristiques évolutionnaires. La première est MBO tandis que la deuxième est un algorithme génétique. Dans la décomposition DC que nous utilisons, la deuxième composante peut être non convexe, d'où l'adoption de DCA-Like qui a été proposé à cette fin. Le schéma coopératif utilisé pour cette classe de problèmes lance les algorithmes composants en même temps et permet d'échanger les informations entre eux. Pour cela, nous optons pour une communication collective. Plus précisément, nous appelons la fonction MPI_Bcast qui sert à diffuser un message d'un processus à tous les autres parallèlement. À titre d'application, nous élaborons notre approche coopérative sur le problème d'affectation quadratique (QAP : *Quadratic Assignement Problem* en anglais). Les résultats numériques comparatifs avec les algorithmes participants (DCA-Like, GA, et MBO) montrent que notre coopération est la meilleure. En étudiant le comportement de chaque algorithme participant lors de la coopération, nous pouvons clairement constater que le principe de coopération est exploité efficacement.

Le dernier chapitre s'articule autour du problème du *clustering*. Nous optons pour le problème de *clustering* avec la minimisation de la somme des carrés communément appelé le MSSC (l'acronyme de : *Minimum-Sum-of-Squares Clustering*). Nous étudions deux approches coopératives qui se basent sur le DCA et les métaheuristiques. En effet, la première approche est une coopération de DCA avec deux métaheuristiques trajectoires : la recherche tabou (TS : *Tabu search*), et la VNS. Quant à la deuxième, elle utilise la métaheuristique évolutionnaire, MBO, en plus des trois algorithmes cités précédemment. Pour les deux approches on introduit deux modèles équivalents du MSSC. Plus précisément, un modèle continu adapté pour le DCA afin d'éviter l'utilisation de la fonction de pénalisation, ainsi qu'un modèle discret adapté pour les métaheuristiques. Pour la communication entre les processus, nous trouvons que la communication point à point à un sens unique est le meilleur choix. Par conséquent, nous choisissons la fonction MPI_Put de la bibliothèque MPI, qui consiste à envoyer l'information, accéder à la mémoire du processus réceptif et l'enregistrer sans bloquer le travail de ce dernier. L'information partagée est constituée de la fonction objectif, la solution, ainsi que du critère d'arrêt. Les résultats des tests numériques réalisés sur des *benchmarks* du MSSC, après avoir effectué une comparaison entre les approches coopératives et les algorithmes composants (DCA, TS, VNS, et MBO), montrent que l'idée de la coopération fonctionne bien et nos approches mènent à des résultats satisfaisants.

Organisation de la thèse

Dans ce manuscrit, nous répartissons nos travaux en quatre chapitres.

- Le premier chapitre est constitué de tous les outils théoriques et algorithmiques nécessaires pour la suite. En effet, nous présentons les approches coopératives et les paradigmes de la programmation parallèle et distribuée. Nous fournissons également dans ce chapitre une brève introduction aux algorithmes participants : la programmation DC et DCA ainsi que les métaheuristiques (VNS, GA, MBO, et TS), utilisés pour la conception des schémas coopératifs proposés.
- Le deuxième chapitre s'articule autour des problèmes linéaires à variables mixtes binaires. Pour la résolution de ces problèmes, nous utilisons une approche coopérative dont les algorithmes composants sont le DCA et la recherche à voisinage variable (VNS). La communication point à point, à deux sens est adoptée dans cette approche.
- Le chapitre trois concerne la programmation quadratique à variables binaires. Nous proposons une autre approche coopérative pour résoudre cette classe de problèmes. Les algorithmes composants sont les algorithmes génétiques, l'optimisation des oiseaux migrateurs (MBO), et le DCA-Like, pour développer l'approche de résolution. La diffusion des informations entre les algorithmes composants est effectuée en faisant appel à un modèle de communication collective à sens unique.
- Le chapitre quatre aborde le problème du *clustering*. Plus précisément, le *clustering* via la minimisation de la somme des carrés. Nous faisons appel à deux modèles mathématiques de MSSC utilisés dans la même approche de résolution. Nous proposons deux approches coopératives. Chacune utilise deux modèles mathématiques équivalents du MSSC. Quant aux algorithmes composants, nous avons opté pour la recherche à voisinage variable, la recherche tabou, et le DCA pour la première tandis que la deuxième utilise la MBO avec les trois derniers algorithmes précédents. Lors de l'implémentation de nos approches, nous utilisons une fonction de la bibliothèque MPI qui permet de faire une communication point à point à sens unique.

Chapter 1

Methodologies

Abstract: *Nonconvex optimization plays a very important role in modeling real-world problems because most of them are nonconvex. To solve these problems many algorithms have been used. In this thesis, we propose new methods called cooperative approaches. The latters are based on many algorithms that run in parallel and simultaneously using the paradigm of parallel and distributed programming. In this chapter, we present the groundwork of this thesis. We introduce the theoretical and algorithmic tools necessary for the following chapters. More precisely, we give a detailed description of the cooperative schemes, parallel and distributed programming, the theory of DC programming and DCA, as well as an overview of metaheuristics.*

1.1 Cooperative schemes

This section is devoted to a general presentation of the cooperative approaches and the tools that we need for these approaches. The concept here consists in using different algorithms that run at the same time for solving a problem. These algorithms are called component algorithms. They cooperate with each other: the cooperation is reflected by using several algorithms working simultaneously in parallel and exchanging information between each other. The implementation of a cooperative approach is based on the paradigms of parallel and distributed programming.

1.1.1 Parallel and distributed programming

Over the years, it was only possible to develop sequential software for which just one processor is used when solving a problem. However, during the implementation of any solution method by using classic sequential programming, the computing time associated with the search space exploration can be very important. To improve the computing power as well as its speed, researchers developed new algorithms basing on parallel and distributed programming techniques. For many years, parallel and distributed programming, this technique that dominates computers, has undergone remarkable progress. The parallelism consists in performing a calculation by cooperating several processors for solving different problems. In parallel programs, each processor treats a part of the whole program independently from the others. The content of this section is extracted from Charot (1993); Delage-Santacreu (2008); Flynn (2011); M Sasikumar (2014); Snir et al. (1996); Cung et al. (2002); Schryen (2020).

Classification of parallel architectures

The works of John Von Neumann during the Second World War are at the origin of the architecture model of most computers. The Von Neumann's machines (see, Fig.1.1.) are mainly constituted by four entities which are:

- **Central memory**
  The storage space of data and the instructions processed by the computer.

- **Input and output devices**
  They are the pieces of hardware that allow communicating with a computer. An input device is used to bring information into a system, for instance: Keyboard, Mouse, Microphone, etc. Whereas an output device sends data out of the system, some of them are monitor and speaker.

- **Central Processing Unit (CPU)**
  It encompasses the arithmetic logic unit (ALU) and the control unit(CU).

The first one performs a set of arithmetic operators (∗, +, etc.) and logical operations (and, or, xor, etc.). However, the second unit manages the responses of Central memory, Input and output devices, as well as the arithmetic logic unit, to the program's instructions.

- **Bus**
  They are interconnection wires to ensure communication between the previous entities.

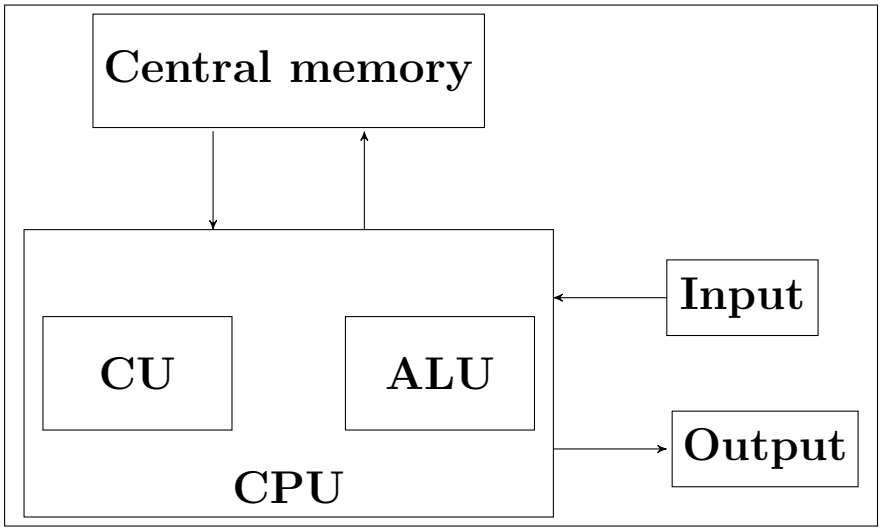


Figure 1.1 – Von Neumann's machine.

In the parallel context, according to the memory, the number of instructions, and data stream, several classifications have been proposed. We give two of them: the first is the most famous and called Flynn's taxonomy and the second is based on the structure of computers.

Flynn's taxonomy

Flynn's taxonomy (see, Fig.1.1.) is a classification of forms of parallel computer architectures. This classification is not based on the structure of computers but rather on the way the computer connects the instructions stream to the data stream.

— **SISD** (Single Instruction Single Data), this is the same principle of the Von Neumann machines, that is to say, treating a single flow of instructions on a single data stream.
— **SIMD** (Single Instruction Multiple Data), in this class, we have a single flow of instructions that must be operated on several data streams.
— **MISD** (Multiple Instruction Single Data), the processors receive different instructions and operate on the same stream of data. They execute the same instruction at each unit of time.

— **MIMD** (Multiple Instruction Multiple Data), the idea here is that each processor treats its instructions on its data streams independently of other processors.

	Single instruction	multiple instructions
Single data	SISD	MISD
multiple data	SIMD	MIMD

Table 1.1 – Flynn's taxonomy.

Classification of memories of parallel machines

Following the memory type, we can place the parallel computers into two main classes: shared memory machines and distributed memory machines.

- **Shared memory machines** (see, Fig.1.2.)

  It involves using multiple processors with one common memory shared between all these processors. The processors are connected to the memory through an interconnection network (bus). The communication between the processors is carried out by reading and writing in the memory. The problem for this architecture is the access of the processors to the memory because if a processor wants to access the memory it must go through the bus and this last may be occupied by another processor which creates a waiting time. To avoid this lost time, the researchers have opted for a second class that will be presented in what follows. For this type of machine, the parallel program is designed using the openMP (Multithreading) packages.

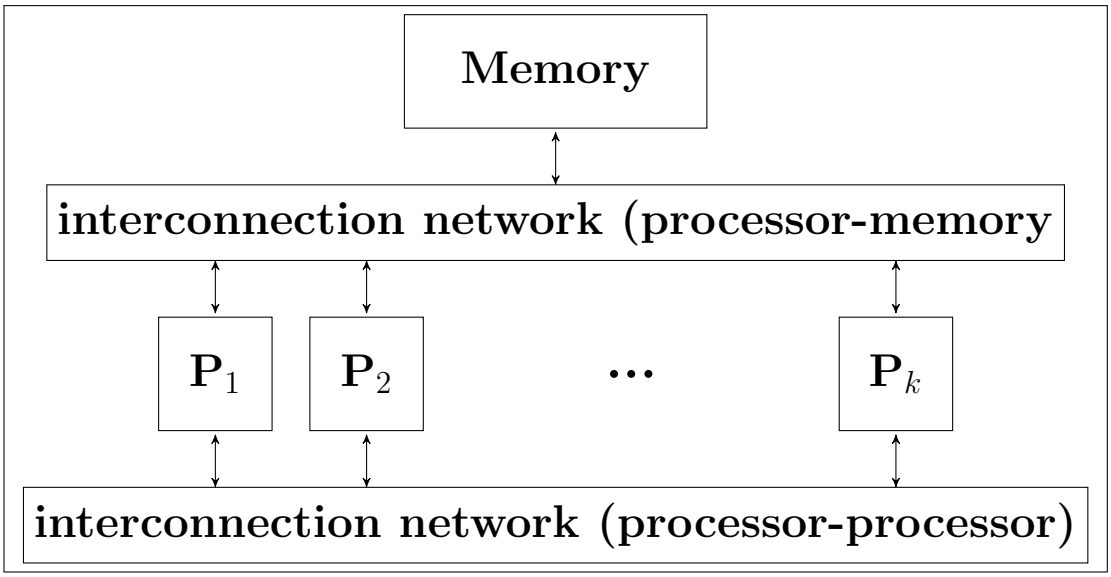


Figure 1.2 – Parallel structure with shared memory.

- **Distributed memory machines** (see, Fig.1.3.)

  It is an extension of the shared memory machine in which each processor has its memory. The processor is connected to its memory through the interconnection network. These last represent the means of connection between the processors. In this class, a processor P1 can not directly access the memory M2. Therefore, it sends information to P2 which will save or read the data in M2. This architecture can also be represented by several computers linked by an interconnection network.
  Here processors can communicate with each other via different protocols. We can distinguish two well-known message passing libraries.
  — Parallel virtual machine (PVM).
  — Message passing interface (MPI).

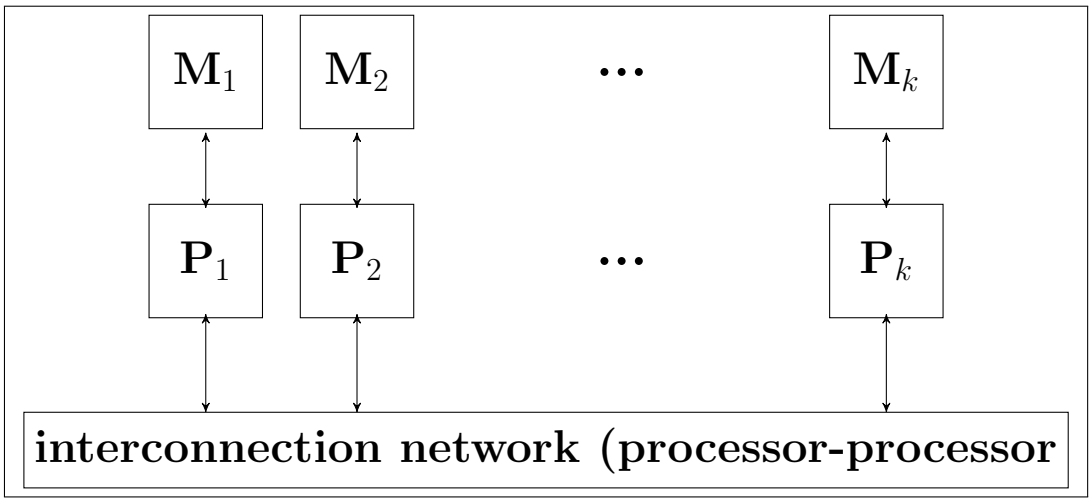


Figure 1.3 – Parallel structure with distributed memory.

Message Passing Interface (MPI)

MPI (Message Passing Interface) is a set of standardized functions belonging to a library, available on distributed memory architectures with many languages such as C/C++ and Fortran LASRI (2016). When we develop a program using MPI, the program is duplicated on many processes. Each process has its memory and runs a copy of the program. Here in each process, the variables become local variables. The process has no acces to the memories of the other processes, however it can send them information, taking in consideration that the receptive processes will be notified by a message. The exchange of data between the process is realized by calling the functions of the library MPI and that is what we call the message passing.
The use of the MPI library requires mainly the following steps.

1. Calling the MPI module (e.g., mpi.h in C++).
2. Initializing the MPI environment (via MPI_Init).

3. Calling the default communicator MPI_COMM_WORLD (for the rank and the size).
4. Terminating MPI execution environment (via MPI_Finalize).

Communications

A communicator in the MPI environment groups all the processes when the program gets started. We can distinguish many types of communications.

1. **Point_to_Point communications**
   Two processes in the same communicator are going to communicate. One of them (the sender) sends a message and the other one receives it. The exchange of information can be done by saving the value of the sent variable using the functions MPI_Send and MPI_Recv, or by overwriting (replacing) the value of the variable via the function MPI_Sendrecv_replace.

2. **Collective communications**
   All the processes in a communicator are going to communicate together. This means that a process can send information to all the other processes at the same time, e.g.:
   - MPI_Bcast allows the sender to broadcast the same data package to all the other processes.
   - MPI_Scatter allows a process to scatter data to processes in the same communicator. MPI_Scatter divides an array into a number of pieces equal to the number of processes and sends each process (including the sender) a different chunk of the array.
   - MPI_Gather, the role of this function is to involve a designated root process taking elements from all the processes and gathering them in this process. The result of the gathering is only known by the process which called the function.
   - MPI_Allgather is a generalization of the previous one. That is to say, each process gathers all the elements from all the other processes.

3. **Blocking communications vs. non-blocking communications**
   We can distinguish two types of sending and receiving a message which are synchronous and asynchronous. The first one blocks a process until the end of the operation. The second one is non-blocking which allows the process to passe to the next operations without waiting time. When we use the blocking functions, the system does not give back control (blocks) to a process to continue its tasks until it has completed the collective task (e.g., MPI_Send, MPI_Sendrec, etc.). In contrast, non-blocking functions optimize the time of communication. Which means, the control is given back to the processes before the communication operation has been finished, for example, we have the functions: MPI_Isend and

MPI_Irecv.
In practice, there are many other important functions:

- MPI_Barrier blocks until all processes in the communicator have reached the barrier.
- MPI_Test is used to test whether the communication operation is completed or not.
- MPI_Wait forces the communication operation to complete.
- etc.

4. **One-sided communications vs. two-sided communications**
   The first one allows the process sender to access the memory of the receiver in order to save and/or take data, for example, MPI_Put, MPI_Get, MPI_Accumulate, etc. Concerning the two-sided communication, the memory is private to each process. Here, when we call a function like MPI_Send and MPI_Recv, the data in the memory of the sender is copied to be sent to the receiver.

1.1.2 Generic scheme of a cooperative approach

The idea of the cooperative approaches is inspired by the brainstorming in real life. The brainstorming is a technique that has been suggested by Faickney (1948) and then by many other authors, like, Kumbhar (2018) and Clark (1958). The main strategy of this technique consists in grouping several participant members to exchange their different ideas in order to solve an advertising problem. The work of these members is organized by an animator. In brainstorming, the underlying assumption is to well take into consideration all the suggested ideas, regardless of how outlandish or important, their aspects are, hoping that a bad suggestion can give birth to a valuable new one. The aim here is to give rise to a new and efficient solution.

The brainstorming has been applied to combinatorial optimization problems (COPs) using different metaheuristic algorithms. Yagouni and Le Thi (2014) have proposed a collaborative approach called the MetaStorming, for the COPs using the same principle as in the brainstorming. The MetaStorming has been tested on benchmarks set for the Travelling Salesman Problem.

The advantage of a cooperative approach is to exploit and combine strengths and weaknesses of component algorithms as in the real brainstorming to produce a source of power. This technique, by exchanging solutions (cooperation), allows also to change the behavior of the component methods while exploring the research space which will surely lead to an improvement of the solution.

Different Steps of a cooperative Approach

The parallel initialization. In this phase, all the processes read data and initialize

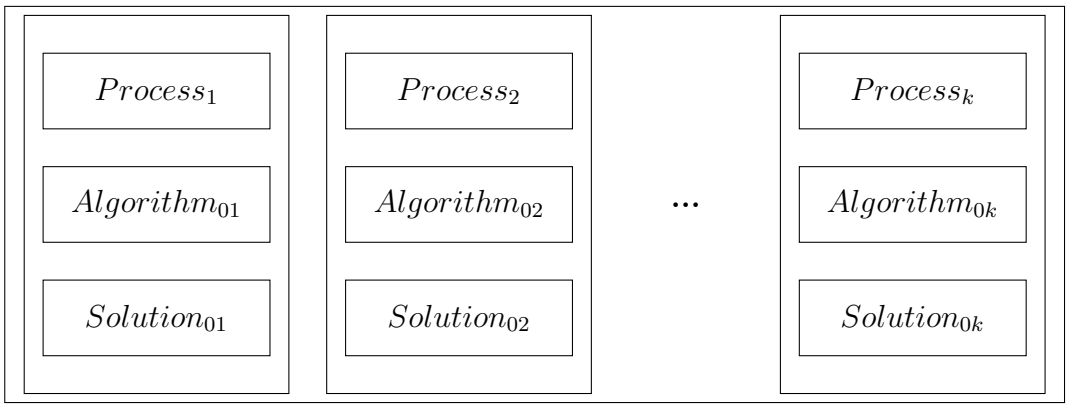


Figure 1.4 – Initialization phase using k processes. $Algorithm_{0j}$ is used by process j to generate an initial solution ($Solution_{0j}$) for the component algorithm j.

the algorithms basing mainly on the randomized functions or any other method. We can either use the same function to generate an initial solution, or a different function for each process.

Parallel and distributed running with cooperation between the component algorithms.

This step is considered as the main part of our approach since it consists of a parallel running and distributed information. Each process j runs m_j iterations of the component algorithm j, simultaneously in parallel. A process j after finishing m_j iterations moves to the cooperation step by distributing information to the other processes. Depending on the adopted communication functions, the processes will either directly exchange the information or wait until all other processes accomplish running their iterations to share it. Then, according to the distributed information, the approach stops or restarts new iterations. In case the stopping condition is not satisfied, a parallel evaluation is performed to determine the best-found solution which will be used as a starting point by the component algorithms in the next iteration.

Remark 1.1. ▶ *The shared information is composed of the objective function, the solution, and/or the stopping condition.*
▶ *The stopping condition can be: when the stopping criteria of all processes coincide, when there is no improvement in the solution, etc.*
▶ *In order to save time, the algorithms exchange first, just the value of the objective function. After the evaluation, the best one distributes its solution. Hence, we avoid sharing useless information.*

1.2 Component Algorithms

In the previous section, we gave the general scheme of cooperative schemes which are composed of a set of component algorithms. In this dissertation, we based on DCA and metaheuristics to design our cooperative schemes. Therefore, this section presents a

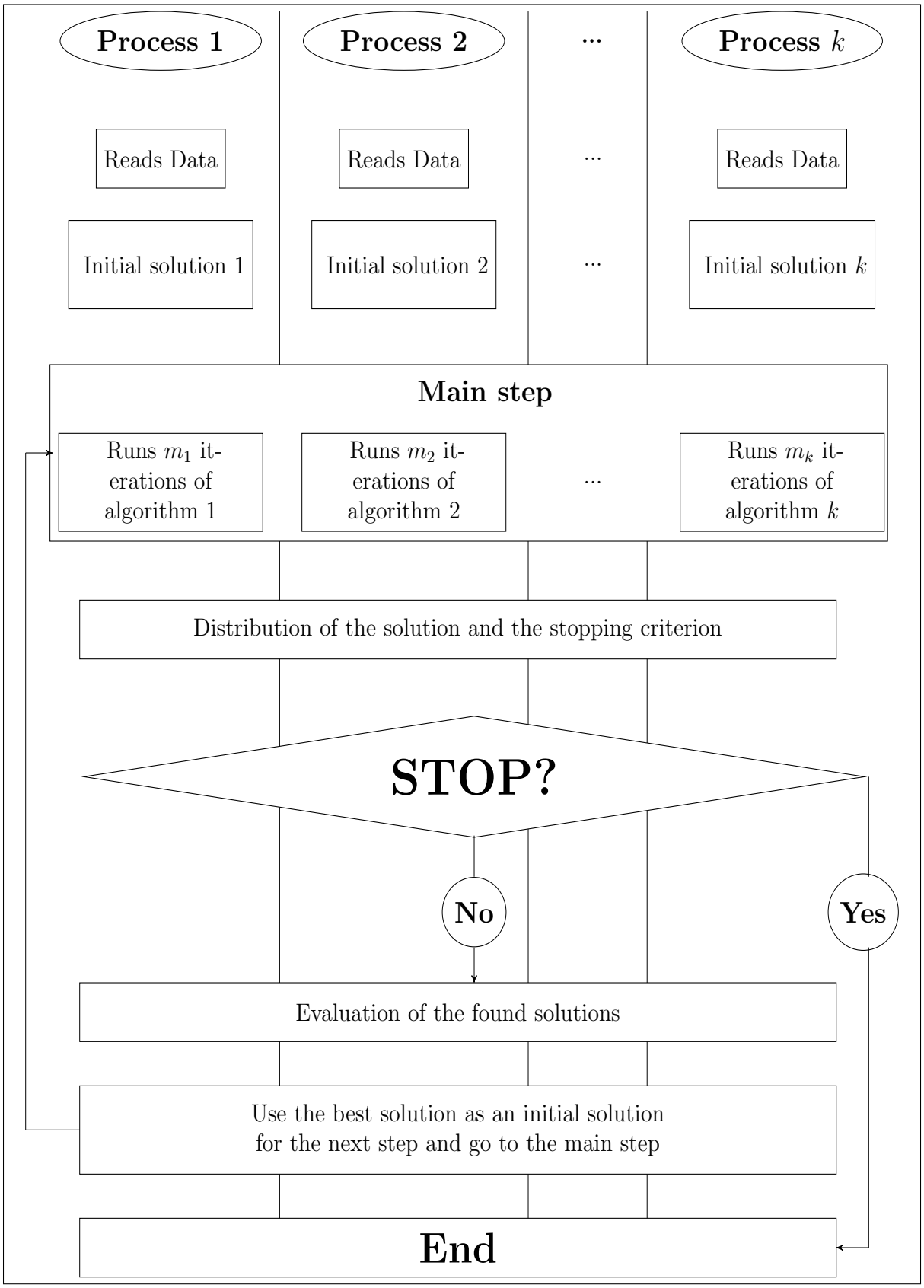


Figure 1.5 – General scheme of a cooperative approach.

brief introduction to DC programming and DCA, an overall overview of metaheuristic approaches, as well as the used ones in the proposed cooperations.

1.2.1 DC programming and DCA

The theory of DC programming and DCA which is known to be the backbone of nonconvex optimization has been introduced firstby Pham Dinh Tao in 1985 (see, Pham Dinh and El Bernoussi (1986)). This theory has been extensively developed from both theoretical and computational aspects since 1994 by Le Thi Hoai An and Pham Dinh Tao.

This section is devoted to the DC programming and DCA. However, before talking about this theory, we recall some basic properties of convex analysis. The materials of this section are extracted from Boyd and Vandenberghe (2004); Le Thi et al. (2014b); Le Thi and Pham Dinh (2005, 2018); Pham Dinh and Le Thi (1997b, 2014); Rockafellar (1970)

Throughout this section, X denotes the Euclidean space $\mathbb{R}^n$ and $\mathbb{R} = \{\overline{\mathbb{R}} = \mathbb{R} \cup \pm\infty\}$ is the set of extended real numbers. Let f be a function whose values are in $\overline{\mathbb{R}}$ and whose domain is C a subset of X.

Fundamental convex analysis

Definition 1.1. *A subset C of X is said to be convex if and only if*

$$\forall x_1, x_2 \in C, \forall \lambda \in [0,1] : \lambda x_1 + (1-\lambda)x_2 \in C. \tag{1.1}$$

Definition 1.2. *A convex hull of a set C is the smallest convex set containing C. We can also say that it is the intersection of all the convex subsets of $\mathbb{R}$ in which C is included. We denote $co(C)$*

$$co(C) = \left\{ x = \sum_{i=1}^{m} \lambda_i x^i : x^i \in C; \lambda_i \geq 0; \forall i = 1, ..., m, \sum_{i=1}^{m} \lambda_i = 1 \right\}. \tag{1.2}$$

Definition 1.3. *The effective domain of a function f, denoted by $domf$ is the set*

$$domf = \{x | f(x) < +\infty\}. \tag{1.3}$$

Definition 1.4. *f is a proper function if*

$$\exists y \in C \ s.t \ f(y) < +\infty \ and \ \forall x \in C \ f(x) > -\infty. \tag{1.4}$$

Definition 1.5. *f is lower semi-continuous at a point $x \in X$ if*

$$f(x) \leq \lim_{y \to x} f(y) \tag{1.5}$$

Definition 1.6. *We suppose that C is a convex set.*

i) ***Convex***
f is said to be a convex function if

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y) : \forall x, y \in C, \lambda \in [0,1], x \neq y. \quad (1.6)$$

ii) ***Strictly convex***
If f satisfies the following condition, we call it a strictly convex.

$$f(\lambda x + (1-\lambda)y) < \lambda f(x) + (1-\lambda)f(y) : \forall x, y \in C, \lambda \in [0,1], x \neq y. \quad (1.7)$$

iii) ***ρ-convex***
Let ρ be a nonnegative number. f is ρ-convex if

$$f(\lambda x+(1-\lambda)y) < \lambda f(x)+(1-\lambda)f(y)-\lambda(1-\lambda)\frac{\rho}{2}\|x-y\|^2 : \forall x, y \in C, \lambda \in [0,1], x \neq y. \quad (1.8)$$

iv) f is said to be ***concave*** *(respectively* ***strictly concave****) if $-f$ is strictly* ***convex*** *(respectively strictly* ***convex****).*

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

Definition 1.7. *Let x_0 be a point in $domf$*

i) ***Subgradient***
We call a vector y a subgradient of a convex function f at a point x_0 if

$$f(x) \geq f(x_0) + \langle y, x - x_0 \rangle, \forall x \in C. \quad (1.9)$$

ii) ***ε-subgradient***
A vector y is called a ε-subgradient of a convex function f at a point x_0 for $\varepsilon > 0$ if

$$f(x) \geq f(x_0 - \varepsilon) + \langle y, x - x_0 \rangle, \forall x \in C. \quad (1.10)$$

iii) ***Subdifferential***
$\partial f(x_0)$ denotes the subdifferential of a convex function f at a point x_0. $\partial f(x_0)$ presents the set of all the subgradients of f at x_0.

$$\partial f(x_0) = \{y | f(x) \geq f(x_0) + \langle y, x - x_0 \rangle; \forall x \in C\}. \quad (1.11)$$

If $\partial f(x_0)$ is not empty, f is said to be subdifferentiable at x_0.

iv) ***ε-subdifferential***

The set of all the ε-subgradients of a convex function f at a point x_0 is called ε-subdifferential of f at x_0 and is denoted $\partial_\varepsilon f(x_0)$.

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

1. *$\partial_\varepsilon f(x)$ is a closed convex set, for any $x \in X$ and $\varepsilon \geq 0$.*
2. *$ri(domf) \subset dom\partial f \subset \partial domf$. where $ri(domf)$ stands for the relative interior of $domf$.*

3. *If f is differentiable at $x \in domf$, then $\partial f(x) = \{f(x)\}$.*
4. *$x_0 \in argmin\{f(x) : x \in X\}$ if and only if $0 \in \partial f(x_0)$.*

Definition 1.8. *Let C be a nonempty convex subset of X. $\chi(f)$ denotes the indicator function of C, where*

$$\chi_C(x) = \begin{cases} 0 \ if \ x \in C, \\ +\infty \ otherwise. \end{cases} \tag{1.12}$$

Definition 1.9. *The function $f^* : X \longrightarrow \overline{\mathbb{R}}$ which is called the conjugate of a function f, can be defined by*

$$f^*(y) = sup\{\langle x, y\rangle - f(x) : x \in X\}. \tag{1.13}$$

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

1. *$f^* \in \Gamma_0(X)$ and $f^{**} = f$.*
2. *$f(x) + f^*(y) \geq \langle x, y\rangle$, for any $x, y \in X$.*
3. *$f(x) + f^*(y) = \langle x, y\rangle \Leftrightarrow y \in \partial f(x) \Leftrightarrow x \in \partial f^*(y)$.*

Definition 1.10. *A hyperplane H in $\mathbb{R}^n$ is a set defined by*

$$H = \{x | a^T x = b\} \tag{1.14}$$

$a \in \mathbb{R}^n$, a is non-zero, and $b \in \mathbb{R}$.

A half space is a hyperplane dividing $\mathbb{R}^n$, into two half spaces.

$$\begin{cases} H_1 = \{x | a^T x \geq b\}. \\ H_2 = \{x | a^T x \leq b\}. \end{cases} \tag{1.15}$$

H_1 and H_2 are two half spaces.

Definition 1.11. *A polyhedral convex set denoted P is the intersection of finitely many closed half-spaces of $\mathbb{R}^n$. P has the form*

$$P = \{x \in \mathbb{R}^n | a_i^T x \leq b_i : a_i \in \mathbb{R}^n, b_i \in \mathbb{R}, \forall i = 1, ..., m\}. \tag{1.16}$$

Definition 1.12. *Let a be a real number*

$$epif = \{(x, a) | x \in C, \ f(x) \leq a\} \tag{1.17}$$

The set $epif$ is called the epigraph of the function f. If $epif$ is convex as a subset of $\mathbb{R}^{n+1}$, then f is said to be convex on C.

Definition 1.13. *A function $f \in \Gamma_0(X)$ is defined to be polyhedral if*

$$f(x) = sup\{\langle a_i, x\rangle - \alpha_i : \forall i = 1, ..., k\} + \chi_C(x), \tag{1.18}$$

where $a_i \in \mathbb{R}^n$, $\alpha_i \in \mathbb{R}$ $\forall i = 1, ..., k$ and C is a nonempty polyhedral set.

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

1. *f is subdifferentiable at x, and $\partial f(x)$ is a polyhedral convex set. In particular, if f is defined by (1.18) with $C = X$ then*

$$\partial f(x) = co\{a_i : i \in I(x)\}, \tag{1.19}$$

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

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

$$dom f^* = co\{a_i : \forall i = 1, ..., k\}.$$

$$f^*(y) = \inf\left\{\Sigma_{i=1}^k \lambda_i \alpha_i | \Sigma_{i=1}^k \lambda_i a_i = y, \Sigma_{i=1}^k \lambda_i = 1, \lambda_i \geq 0, \forall i = 1, ..., k\right\}. \tag{1.20}$$

In particular,

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

Definition 1.14. ***Difference of convex (DC) functions***

A DC function is a function that can be expressed in the form

$$f(x) = g(x) - h(x), x \in X \tag{1.21}$$

where g and h belong to $\Gamma_0(X)$.

$$\begin{cases} g \text{ and } h \text{ are called the DC component.} \\ g - h \text{ is the DC decomposition.} \end{cases} \tag{1.22}$$

If g and h are in addition finite on all of X then one says that $f = g - h$ is finite DC function on X.

Remark 1.2.

1. *Give a DC function f having a DC decomposition $f = g - h$. Then for every $\theta \in \Gamma_0(X)$ finite on the whole X, $f = (g + \theta) - (h + \theta)$ is another DC decomposition of f. Therefore, a DC function f has infinitely many DC decompositions.*
2. *The set of DC functions (resp. finite DC functions) on X is denoted by $\mathbb{DC}(X)$ (resp. $\mathbb{DC}_f(X)$).*

DC optimization

Most of the optimization problems are nonconvex. The majority of these last can be formulated as the following forms Pham Dinh and Le Thi (1997a):

(A) $\sup\{f(x) : x \in C\}$, where f and C are convex.

(B) $\inf\{g(x) - h(x) : x \in \mathbb{R}^n\}$, where g and h are convex.

(C) $\inf\{g(x) - h(x) : x \in C; f_1(x) - f_2(x) \leq 0\}$, where g, h, f_1, f_2 and C are convex.

We call Problem (A) a convex maximization program. It is a special case of a DC program. Problem (B) is a standard DC program, and Problem (C) a general DC program.

The constrained DC program can be transformed into a standard DC program by using the indicator function $\chi_c(x)$ in the first DC component, i.e.,

$$\begin{array}{l} g(x) = \chi_c(x). \\ h(x) = f(x). \end{array} \tag{1.23}$$

It can easily be noted that Problem (C) is the most general and it is more difficult than the constrained and standard DC programs, due to the nonconvexity of the constraints. To reformulate (C) as standard DC program two approaches have been proposed in Le Thi et al. (2014b). The first one consists in introducing the penalty techniques in DC programming. The second is based on the idea of linearizing the concave functions in DC constraints to build convex inner approximations of the feasible set.

Standard DC optimization

In this section, we point out the main results in standard DC programs which are found by Le Thi Hoai An and Pham Dinh Tao and they are presented in their works such as Pham Dinh and Le Thi (1997a); Le Thi and Pham Dinh (2005); Pham Dinh and Le Thi (2014); Le Thi and Pham Dinh (2018).

Standard DC program

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

DC Programming and DCA address the problem of minimizing a function f which is a difference of convex functions on the whole space X. Generally speaking, a so-called standard DC program takes the form

$$(P) \qquad \alpha = \min\{f(x) = g(x) - h(x) : x \in X\}, \tag{1.24}$$

where $g, h \in \Gamma_0(\mathbb{R}^n)$.

In DC programming, (P) is called a primal DC program and it is associated with its dual which is also a DC program with the same optimal value. We denote the dual program by (D) and we define it by

$$(D) \qquad \alpha^* = \min\{f(y) = h^*(y) - g^*(y) : y \in Y\}. \tag{1.25}$$

There is a perfect symmetry between (P) and its dual (D): the dual of (D) is exactly (P).

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

$$domg \subset domh \text{ and } domh^* \subset domg^*. \tag{1.26}$$

Polyhedral DC program

Polyhedral DC program occurs when in a DC program (P) at least one of DC components g and h is polyhedral convex. In this case, P can be written as

$$(\tilde{P}) \qquad \alpha = \min\left\{f(x) = g(x) - \tilde{h}(x) : x \in X\right\}, \tag{1.27}$$

where the second DC component is supposed to be a polyhedral convex function.

Optimality conditions for standard DC optimization

A point x^* is said to be a local minimizer of $g - h$ if $x^* \in domg \bigcap domh$ (so, $(g-h)(x^*)$ is finite) and there is a neighborhood U of x^* such that

$$g(x) - h(x) \geq g(x^*) - h(x^*), \qquad \forall x \in U. \tag{1.28}$$

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

$$\partial g(x^*) \bigcap \partial h(x^*)\emptyset. \tag{1.29}$$

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

$$\mathbb{P}_l = \{x^* \in X : \partial h(x^*) \subset \partial g(x^*)\}, \tag{1.30}$$

$$\mathbb{D}_l = \{y^* \in X : \partial g(y^*) \subset \partial h(y^*)\}. \tag{1.31}$$

Below, we present some fundamental results on DC programming Pham Dinh and Le Thi (1997b).

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

$$\partial_\epsilon h(x^*) \subset \partial_\epsilon g(x^*); \forall \epsilon > 0. \tag{1.32}$$

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

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

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

$$g(x) - h(x) \geq g(x^*) - h(x^*), \qquad \forall x \in U \cap domg. \tag{1.33}$$

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

$$y^* \in int(domg^*) \quad and \quad \partial g^*(y^*) \subset U, \tag{1.34}$$

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

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

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

c) The necessary local optimality condition $\partial h^*(x^*) \subset \partial g^*(x^*)$ is also sufficient for many important classes programs, for example Le Thi and Pham Dinh (2005), if h is polyhedral convex, or when f is locally convex at x^*, i.e. there exists a convex neighborhood U of x^* such that f is finite and convex on U. We know that a polyhedral convex function is almost everywhere differentiable, that is to say, it is differentiable everywhere except on a set of measure zero. Thus, if h is a polyhedral convex function, then a critical point of $g - h$ is almost always a local solution to (P).

d) If f is actually convex on X, we call (P) a "false" DC program. In addition, if $ri(domg) \cap ri(domh) \neq \emptyset$, and $x^* \in domg$ such that g is continuous at x^*, then $0 \in \partial f(x^*) \Leftrightarrow \partial h(x^*) \subset \partial g(x^*)$ Le Thi and Pham Dinh (2005). Thus, in this case, the local optimality is also sufficient for the global optimality. Consequently, if in addition h is differentiable, a critical point is also a global solution.

DCA

DCA is an iterative approach proposed for solving DC problems. Therefore, it requires a DC representation to the objective function. In addition to that, this approach uses the DC components, instead of directly adopting the function f. This algorithm can be described as follows.

Algorithm 1.1 DCA

Initialization: Set an initial solution x^0, $k = 0$, $\epsilon_1 > 0$, $\epsilon_2 > 0$.
1. Compute $y^k \in \partial h(x^k)$.
2. Compute $x^{k+1} \in argmin\{g(x) - \langle x, y^k \rangle : x \in \mathbb{R}^n\}$
3. If$(\|x^{k+1} - x^k\| \leq \epsilon_1(\|x^k\| + 1)$ or $|f(x^{k+1}) - f(x^k)| \leq \epsilon_2(|f(x^k)| + 1))$, then **stop**.
4. Otherwise, $k \leftarrow k + 1$; go to 1.

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

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

ii) Their corresponding limits x^∞ and y^∞ satisfy the local optimality condition $(x^\infty, y^\infty)2\mathbb{P}_l \times \mathbb{D}_l$ or are critical points of $(g-h)(x^k)$ and $(h^*-g^*)(y^k)$, respectively.

From a given initial point $x^0 \in domg$, the DCA generates these sequences by the scheme

$$y^k \in \partial h(x^k) = argmin\{h^*(y) - \langle y, x^k\rangle : y \in X\}, \tag{1.35}$$

$$x^{k+1} \in \partial g^*(y^k) = argmin\{g(x) - \langle x, y^k\rangle : x \in X\}. \tag{1.36}$$

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

$$h_k(x) = h(x^k) + \langle x - x^k, y^k\rangle, \tag{1.37}$$

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

$$(P_k) \qquad \alpha_k = \min\{f_k(x) = g(x) - h_k(x) : x \in X\}, \tag{1.38}$$

that is equivalent to (1.36). It is easy to see that f_k is a majorant of f which is exact at x_k i.e. $f_k(x^k) = f(x^k)$. Similarly, by replacing g^* with its affine minorant

$$g_k^*(y) = g^*(y^{k-1}) + \langle y - y^{k-1}, x^k\rangle \tag{1.39}$$

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

$$(\mathbb{D}_k) \qquad \inf\{h^*(y) - g_k^*(y) : y \in X\} \tag{1.40}$$

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

Well definiteness of DCA

DCA is well defined if one can construct two sequences $\{x^k\}$ and $\{y^k\}$ as above from an arbitrary initial point x^0. The following Lemma is the necessary and sufficient condition for this property

Lemma 1.1. *(Pham Dinh and Le Thi (1997b)). The sequences $\{x^k\}$ and $\{y^k\}$ in DCA are well defined if and only if*

$$dom\partial g \subset dom\partial h \quad and \quad dom\partial h^* \subset dom\partial g^*. \tag{1.41}$$

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

Convergence properties of DCA

Complete convergence of DCA is given in the following results (Pham Dinh and Le Thi (1997b)).

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

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

DCA terminates at the k^{th} iteration if either of the above equalities holds.

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

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

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

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

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

Remark 1.4. *a) Finding $\{y^k\}$, $\{x^{k+1}\}$ based on the scheme (1.35) and (1.36) amounts to solving the problems (D_k) and (P_k). Thus, DCA works by reducing a DC program to a sequence of convex programs which can be solved efficiently.*

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

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

d) DCA's distinctive feature relies upon the fact that DCA deals with the convex DC components g and h but not with the DC function f itself. Moreover, a DC function f has infinitely many DC decompositions which have crucial implications for the qualities (e.g. convergence speed, robustness, efficiency,

globality of computed solutions) of DCA. For a given DC program, the choice of optimal DC decompositions is still open. Of course, this depends strongly on the very specific structure of the problem being considered.

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

Since its first appearance, DCA has not ceased developing. Recently, many optimization versions based on the powerful tools of nonconvex programming DC programming and DCA, have been proposed for increasing the speed of convergence of DCA. For example:

- DCA with successive DC decomposition Le Thi and Pham Dinh (2018);
- approximate DCA Le Thi and Pham Dinh (2018);
- the accelerated DCA Phan et al. (2018); Le Thi et al. (2018);
- DCA-like and its accelerated version Le Thi et al. (2018);
- online DCA Ho et al. (2016); Le Thi and Ho (2020);
- stochastic DCA Le Thi et al. (2017, a,b,c);
- collaborative DCA Le Thi (2019b);

1.2.2 Metaheuristics

In this section, we report an overview of the metaheuristic approaches and we briefly give details about some metaheuristics that we use in our study. These contents are extracted from Al-Sultan (1995); Alba (2005); Belacel et al. (2004); Blum and Roli (2008); Duman and Elikucuk (2013); Duman et al. (2012); Franco-Sepúlveda et al. (2019); Glover (1986); Glover and Laguna (1998); Handl and Knowles (2007); Hansen and Mladenović (2001); Mladenović and Hansen (1997); Osman and Laporte (1996); Rochat and Taillard (1995); Smith (2002); Tongur and Ülker (2014); Voß (2001).

In optimization, due to the complexity of many problems, it is a challenge to find an optimal solution to these problems in polynomial time. For this purpose, many algorithms have been proliferated and different programming techniques have been used. There exists a lot of methods that ensure the globality. However, the application of these methods often performs very poorly (it implies an exponential time) which led researchers to promote getting a good quality solution in a reasonable computational cost, rather than guaranteeing the globality and consuming a large amount of time. Here, the time factor dominates that of quality. Hence researchers oriented towards the approximate methods which include, but are not limited to heuristics and metaheuristics. In the main, these approaches achieve good quality solutions within an acceptable time and offer no guarantee of obtaining global solutions.
The heuristics consist in solving an optimization problem in polynomial time. However, the limit of these approaches is that they provide a solution basing on local search. Accordingly, the quality of the solution is not as efficient as the one obtained by other classes of algorithms. Therefore, they are used to build a starting solution that will be improved using another algorithm.

Metaheuristics are optimization algorithms that are generally inspired by the behavior of nature. They are formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information to find efficiently near-optimal solutions", in Osman and Laporte (1996). They are useful tools for solving NP-Hard optimization problems, easy to implement, less expensive in terms of time and perform good solutions.

A metaheuristic is mainly based on two strategies the *diversification* and the *intensification*. The first one drives the search to explore new regions in the solution space, regardless of the quality of its solution. Diversification leads the metaheuristic to avoid trapping in a local minimum. The second focuses more intently on the promising regions of the space of research.

Classification of metaheuristics

In the literature, metaheuristic algorithms can be classified into different categories by taking into consideration several criteria (usage of memory, nature-inspired, single solution or a population, etc.). This section briefly summarizes the most important categories:

1. **Nature-inspired metaheuristics vs. non-nature inspired metaheuristics.**

   Nature-inspired metaheuristics are a simulation of natural phenomena adapted to optimization problems e.g., ant colony, genetic algorithm, elephant herding algorithm, etc. Tabu search and variable neighborhood search are non-nature inspired metaheuristics.

2. **Memory-based vs. memory-less methods**

   The first category concerns the methods using memory which is considered as the fundamental element of a powerful metaheuristic. The memory-less methods carry out a Markov process, as the information they use to determine the next action is the current state of the search process.

3. **Trajectory methods vs. population-based search**

   The trajectory metaheuristics also called point-to-point methods, generates at each iteration a single solution (simulated annealing, variable neighborhood search). As for the second class, it invokes a set of solutions and describes their evolution, e.g., scatter search, genetic algorithms, etc.

In the literature, many metaheuristics have been proposed, e.g. :

- Genetic algorithms.
- Variable neighborhood search.
- Tabu search.
- Simulated annealing.
- Ant colony.
- Scatter search.
- Particle swarm optimization.
- Elephant herding algorithm
- migrating birds optimization
- Bee metaheuristics (artificial bee colony, bee colony optimization, etc.).
- Cuckoo Search.

Metaheuristics used in this thesis

This section is devoted to a brief presentation of the main metaheuristic approaches that we use in our work.

1. **Genetic algorithm**
   The genetic algorithms (see, Fig.1.6.) named also Darwinian approaches ascribed to Charles Darwin who revolutionized biology with his work are adaptative stochastic optimization algorithms. The genetic algorithms proposed by Holland in 1975 are evolutionary algorithms, inspired by the biological model of the natural evolution of species. To apply a genetic algorithm (GA), we keep the same context as in the natural evolution of species. Therefore, a solution in optimization is equivalent to a chromosome which is the genetic representation of the solution to our problem and is composed of genes. These last contain all the attributes (information) of the solution. The term "a population" refers to a set of solutions. We start by the coding principle in GA, we give the required parameters, the different operators, and the pseudo-code as well. The following scheme in Fig.1.6. is an organigram of the genetic algorithm.

   Coding
   Coding is a schematization of the solution. It contains all the characteristics of a feasible solution so that it differs from one problem to another. In optimization, it is recommended to choose the best coding way because efficient coding allows obtaining a good quality solution and contributes to saving time. We can distinguish numerous coding techniques: binary, real, Gray, etc.
   If we take the Resource-Constrained Project Scheduling Problem (RCPSP), the genes can contain the following attributes: the task identifier, processing time, start date, end date, predecessor tasks, precedence type, resource requirements, and quantity of each resource. The case of the traveling salesman problem (TSP) is more simple. A gene can be represented by an identifier of the city.

   The fitness function (adaptation or evaluation function)
   This parameter measures the quality of the solution. Hence, the fitness function is expressed by the objective function.

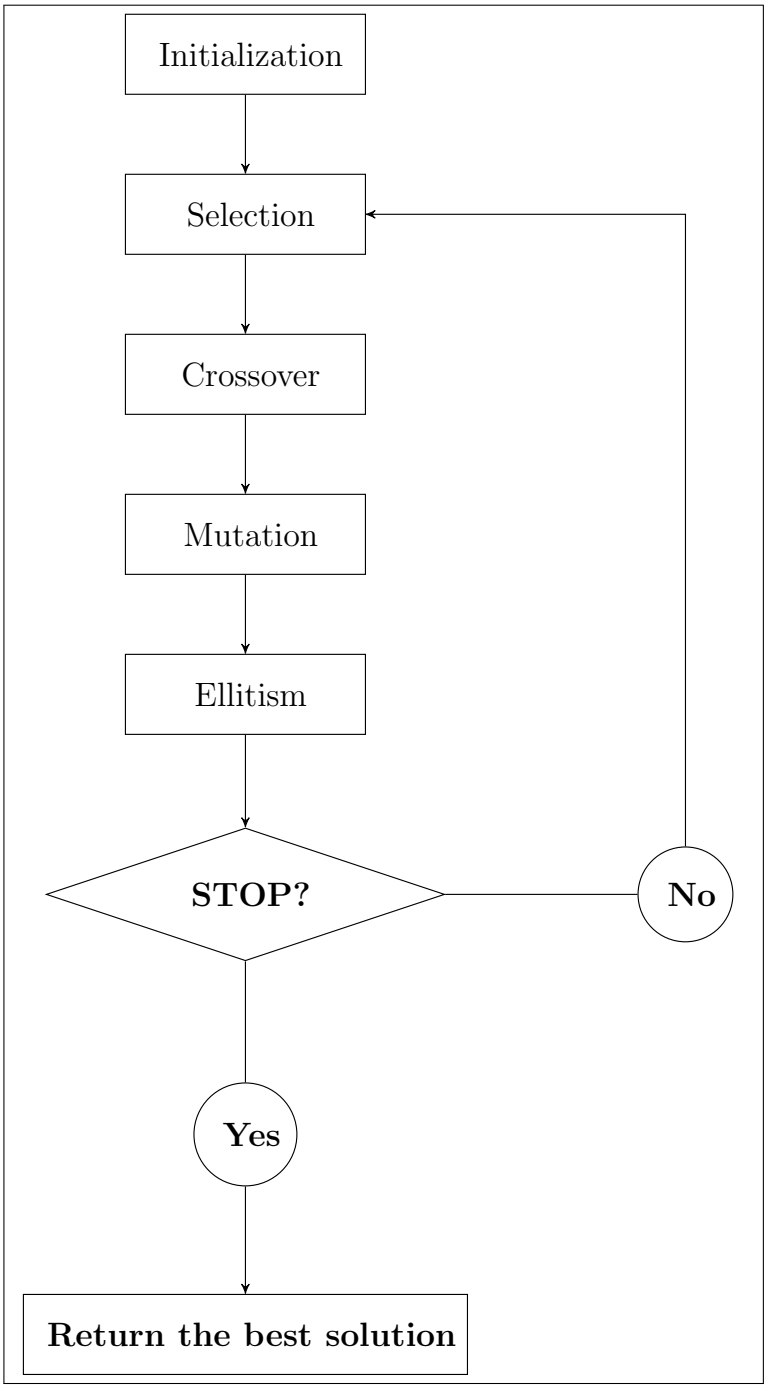


Figure 1.6 – The organigram of a genetic algorithm.

To start the procedure of GA, a population of solutions must be generated and evaluated. It is preferred to choose a reasonable size of the initial population because if it is big the algorithm takes a large time interval.

(i) **Selection operator**
A subset of the population, called parents, is selected to be mated and to produce new offsprings. Generally speaking, many selection procedures can be considered: the tournament selection (binary or between more than two individuals), another is the roulette wheel selection, stochastic universal sampling, rank selection, random selection, etc.

(ii) **Crossover operator**
As in nature, the crossover is applied to the parents to create a new individuals called children. A child takes its characteristics from both of its parents. The resulting children are often infeasible, which led the researchers to apply different techniques to overcome this problem. In our study, we opted for replacing the repeated genes by the missing genes using the crossover probability.

(iii) **Mutation operator**
It consists in randomly modifying one gene or more of a chromosome. The mutation operator aims to reveal new properties that did not exist in the population. Each individual has a chance to mutate, that is what we call the probability of mutation.

(iv) **Ellitism operator**
This operator allows keeping the individuals which have the best fitness functions for the next generation.

2. **Migrating birds optimization**

Contemplating nature allows scientists to create new ideas. From the V formation flight (see, Fig.1.7.) of the migrating birds and existing studies about it, a new metaheuristic algorithm has been proposed to solve the combinatorial optimization problem in Duman et al. (2012). This metaheuristic is called migrating birds optimization (MBO). It consists in exploiting the power saving in the V shape flight to minimize (or maximize) an objective function of a mathematical problem. https://sunprairieumc.org/2016/10/24/the-sense-of-a-goose/

The MBO is based on a population of birds and uses the neighboring search technique like other metaheuristics. A bird in a combinatorial optimization problem represents a solution. To show similarities with the real V formation flight, one bird is considered as a leader and the others constitute the right and left lines. After generating the initial population and choosing the neighborhood structure, the MBO starts running by treating from the leader and progressing on along the lines till tails. Before discussing the procedure of MBO, basing on the study investigated by Duman et al. (2012), we summarize the analogy between real-life V shape and parameters of MBO in table 1.2.



Figure 1.7 – The V formulation flight of the migrating birds. Source: https://sunprairieumc.org/2016/10/24/the-sense-of-a-goose.

Parameters of MBO	Real V formulation
n: Size of population	Number of birds
k: Number of neighborhood	Speed of bird
x: The shared neighbors $(2x < k)$	Wing distance
m: Number of tours	Maximum Number of flutters

Table 1.2 – An analogy between MBO and real V shape of migrating birds.

We try to improve the leader by its k neighbor solutions and then it shares the $2x$ better neighbors with the right and left lines. Then it generates $(k - x)$ neighbors to bird on the left side and tries to improve it. After that, its x best neighbors will be shared with the next bird in the same line till the tail. It proceeds in the same manner with birds of the right line. Notably, the energy spent by the leader is more than that consumed by other birds. More precisely, for the leader k neighbors are generated and $2x$ of them are shared with two lines. However, for the other birds' $k - x$ neighbors need to be generated and just x solutions among them will be shared and with only one line. We keep the same leader for a certain number of tours (until it gets tired) then we replace it with the first bird in the left (alternately, right) line and it will be placed in the tail of this line. Before stopping the MBO, all birds must lead the flock for at least one time. In addition to the precedent condition, the total number of iteration, the running time or if there is no improvement to the best solution after several iterations, can be used as a stop criterion. Now we are in the position to give the pseudo-code of MBO (see, Algorithm 1.2.).

Algorithm 1.2 MBO algorithm

Initialization : n, m, k, x, $nbrTour$.
1. Generate new population and choose the leader.
repeat
2. **Do**
3. Generate k neighbors for the leader and improve it.
4. Share $2x$ best neighbors of the leader with the left and right lines.
5. **For all** birds in the left and the right lines, do
6. Generate $k - x$ neighbors.
7. Try to improve it by using the shared and generated neighbors.
8. Share the x best neighbors with the next bird.
9. **End_For.**
10. $nbrTour = nbrTour + 1$.
11. **While** $(nbrTour < m)$.
12. Change the leader.

until Stopping criterion.

3. **Variable neighborhood search**
The variable neighborhood search has been presented first by Mladenović in 1995 in a conference and then in 1997, by Mladenović and Hansen (see, Hansen and Mladenović (2001); Mladenović and Hansen (1997)) for solving combinatorial optimization problems. VNS is a metaheuristic that adopts the strategy of the systematic change of neighborhood using the techniques of the local search. It is Characterized by two ways of exploring the set of feasible solutions. VNS performs a local search and intensifies a promising area until no improvement is possible for the current solution. After that, the VNS, to diversify the search, it proceeds by moving to a new area by changing the neighborhood and/or calling randomize functions. The VNS algorithm can be described as follows in Algorithm 1.3.

Algorithm 1.3 VNS

Initialization : Initial solution x, neighborhood structures $N_{k(k=1,\ldots,k_{max})}$, k:=1, stopping criterion.
1. $x \leftarrow$ Local-search(x).
2. while(k $\leq k_{max}$) do
3. $x^{'}$ is a randomly generated solution in the kth neighborhood of x ($N_k(x)$).
4. $x^{''} \leftarrow$ Local-search($x^{'}$).
5. If ($x^{''}$ is better than x) then $x := x^{''}$; k:=1.
6. Otherwise, k:=k+1.
7. End.

4. **Tabu search**

The methodology of tabu search (see, Algorithm 1.4.) has been proposed first by Glover in 1986 (see, Glover (1986)), to solve combinatorial optimization problems. It starts with an initial point, a neighborhood technique, a tabu list, and aspiration criteria. First, it generates a neighbor solution from the current one. Then it puts the new solution in the list of the forbidden moves. If the list is full then it deletes the oldest solution. After that, we use the aspiration criteria to allow moving to a banned solution. The strategy of TS memorizes and forbids some solutions to avoid getting trapped in a local minimum.

Algorithm 1.4 Tabu Search

Initialization : Initial solution x, neighborhood technique N, L: the tabu list, k: the size of L, stop criterion.
1. while(stop criterion not satisfied) do
2. $x^{'} = N(x)$.
3. $L = L + \{x^{'}\}$.
4. If ($f(x^{'}) < f(x)$) then $x := x^{'}$;
5. End.

Chapter 2

Cooperative methods for solving MBLP

Abstract: *This chapter addresses the Mixed Binary Linear Programming problems (MBLPs). We propose a cooperative approach using two component algorithms. The first is DCA, an efficient deterministic algorithm in the nonconvex programming framework, and the second is a variable neighborhood search metaheuristic, a well-known metaheuristic method. The DCA and VNS are executed in parallel. At the end of each cycle, the best-found solution is exchanged between these algorithms via functions belonging to the two-sided point-to-point communication of the MPI library. The next cycle starts with the previous best-found solution as an initial solution. We exploit our approach on the Capacitated Facility Location Problem (CFLP) and test its performance on a set of benchmarks of this problem. The numerical experiments show the efficiency of our approach.*

1. The material of this chapter is developed from the following work:
[1]. Samir S., Le Thi H.A. (2019) A Collaborative Approach Based on DCA and VNS for Solving Mixed Binary Linear Programs. In: Nguyen N.T., Gaol F. L., Hong T.-P., Trawiński B. (eds) Intelligent Information and Database Systems. ACIIDS 2019. Lecture Notes in Computer Science, vol 11431, pp. 510-519, Springer, Cham.

2.1 Introduction

Mixed Binary Linear Programs (MBLPs) are NP-hard optimization problems which take the form:

$$(MBLP)\begin{cases} \min Z = c^T x + d^T y, \\ Ax + By \leq b, \\ x \in \{0,1\}^n, \quad y \in \mathbb{R}^p_+, \end{cases} \tag{2.1}$$

where $c \in \mathbb{R}^n$, $d \in \mathbb{R}^p$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m*n}$ et $B \in \mathbb{R}^{m*p}$.
Let C be the set of feasible solutions of MBLP and let Ω be the corresponding linear relaxation set, say

$$\Omega = \{(x,y) \in \mathbb{R}^n * \mathbb{R}^p | Ax + By \leq b, x \in [0,1]^n, y \in \mathbb{R}^p_+\}. \tag{2.2}$$

Related works

MBLP plays a central role in combinatorial optimization and is a common model of several applications including scheduling, transportation, knapsack, routing, assignment, etc.

This problem has been studied by many searchers for a long time (see, Cabot and Hurter (1968); Crowder et al. (1983); Glover (1968); Qi and Sen (2017), etc.). One of the old exact numerical approaches is the direct search algorithms proposed in Lemke and Spielberg (1967). Another direction has been followed to exploit the fact that the optimal solution of an MBLP may be found at an extreme point of the LP feasible region. Thus, methods based on cutting plane and search processes have been studied in Balas et al. (1993), Marchand et al. (2002), etc. The automatic reformulation has also been used for MBLP in Van Roy and Wolsey (1987). Other searchers opted for algorithms based on branch-and-bound techniques. For example, Semih Atakan and Suvrajeet Sen developed a branch-and-bound algorithm based on progressive hedging Atakan and Sen (2018). We can find also many works combining cutting plane with branch-and-bound. Martin Alexander, in Martin (2001), has discussed a survey about the basic features of a branch-and-cut algorithm for solving mixed integer linear programs including MBLP. In Liu et al. (2018), they have proposed a branch-and-cut algorithm with application to the two-echelon capacitated vehicle routing problem with grouping constraints, etc. However, large scale MBLPs are not amenable to be solved using exact methods, which has prompted the development of heuristic and metaheuristic approaches that aim to find a good feasible solution. From the existing metaheuristics in the literature, we can cite, tabu search proposed by Lokketangen and Glover (1998), variable neighborhood pump heuristic by Hanafi et al. (2010), variable neighborhood search with local branching by Hansen et al. (2006), variable neighborhood decomposition search by Lazić et al. (2010), etc. There exist also some approaches based on the hybridization of branch-and-bound and metaheuristics, for example, the work of Nwana et al. (2005) in which a cooperative parallel combination between simulated annealing and branch and bound was proposed. Another technique of resolution is the local deterministic approaches such as DCA which achieve a local solution (which could be a global solution if the algorithm starts from a good initial

point), we can refer the reader to Le Thi et al. (2007d); Quang Thuan and Le Thi (2011). The mixed integer programming has been solved by hybridization between DCA and VNS by Chu (2017) in which the VNS solution has been used as an initial point for DCA.

Contribution and motivation

The main contribution of our work relies on a cooperative approach based on the deterministic algorithm DCA and variable neighborhood search metaheuristic, using paradigms of parallel and distributed programming, for solving MBLP. Our work is inspired by the idea of the Metastorming approach introduced in Yagouni and Le Thi (2014) (a collaboration between metaheuristics). The particularity of our approach is the cooperation between a metaheuristic and a deterministic algorithm, which has never been proposed before. In the present approach, the cooperation is expressed by exchanging information between component algorithms using the two-sided point-to-point communication of MPI Library. More precisely, this operation is made by calling the functions MPI_Send and MPI_Recv. The choice of component algorithms is motivated by the successful application of VNS in combinatorial optimization and the power of DCA in nonconvex programming. Being a continuous approach, DCA works on continuous domains but it achieves a very good integer solution if an integer solution is found during the algorithm. Therefore, a cooperation between VNS and DCA could be an efficient method for solving MBLP.

DC programming and DCA have been successfully applied to large-scale non-convex programs in various areas. A seminal review on thirty years of development of DC programming and DCA can be found in Le Thi and Pham Dinh (2018). To treat MBLP as a DC program, we will use exact penalty techniques (see, Le Thi et al. (1999, 2011)). In the literature, several forms of VNS have been applied to combinatorial optimization problems (see, e.g., Bercachi (2010); Fleszar and Hindi (2004); Hansen and Mladenović (2001); Lazić (2010); Mladenović and Hansen (1997); Salehipour et al. (2011)).

As an application, we consider the capacitated facility location problem (CFLP) to evaluate the performance of our approach. We show how to adapt the component algorithms and the cooperative scheme for this problem.

The rest of the chapter is organized as follows. Section 2 gives an overall description of the component algorithms including VNS, DCA, and their application for solving MBLP. Section 3 is devoted to the cooperative approach VNS-DCA and its application to the MBLP. Section 4 is an application of DCA, VNS, and the cooperative approach to the capacitated facility location problem. The numerical results are given and discussed in Section 5 while the conclusion is mentioned in Section 6.

2.2 DCA and VNS for solving MBLP

To facilitate the reader's understanding, let us introduce briefly how to apply DC programming and DCA and VNS to MBLP, before showing how to adapt the cooperative

approach VNS-DCA to this class of problems.

2.2.1 DCA for solving MBLP

DCA was first investigated for MBLP in Le Thi and Pham Dinh (2001) and then for several applications of MBLP (e.g. Le Thi and Pham Dinh (2018)). We present here below one version of DCA for MBLP Le Thi and Pham Dinh (2001).

DC reformulation and DCA for solving MBLP

To treat (2.1) as a DC program, we will reformulate it as a concave quadratic program with continuous variables by using the exact penalty function. We consider p the penalty function defined by

$$p(x) = \Sigma_{i=1}^{m} x_i(1 - x_i). \tag{2.3}$$

The problem (2.1) is equivalent to

$$\min \left\{ \; c^T x + d^T y : (x, y) \in \Omega; p(x) \leq 0 \; \right\}. \tag{2.4}$$

Thus, we can define MBLP_t the penalized program of (2.1) by

$$\min \left\{ \; F(x, y) = c^T x + d^T y + tp(x) : (x, y) \in \Omega \; \right\}. \tag{2.5}$$

Where $t \in \mathbb{R}$ and $t > 0$.

Theorem 2.1. *Le Thi et al. (1999) Let $\mathbb{K}$ be a non-empty bounded polyhedral convex set on $\mathbb{R}^n$. Let f be a finite concave function on $\mathbb{K}$ and p a finite non negative concave function on K. Then there exists $t_0 \geq 0$ such that for $t \geq t_0$, the following problems have the same optimal value and the same solution set:*

i) $\alpha(t) = inf\{f(x) + tp(x) : x \in \mathbb{K}\}$.

ii) $\alpha = inf\{f(x) : x \in \mathbb{K}, p(x) \leq 0\}$.

Furthermore, if the vertex set of $\mathbb{K}$, denoted with $V(\mathbb{K})$ is contained in $\{x \in \mathbb{K}, p(x) \leq 0\}$, then $t_0 = 0$, otherwise $t_0 = \min\{\frac{f(x)-\alpha(0)}{\xi} : x \in \mathbb{K}, p(x) \leq 0\}$, where $\xi := \min\{p(x) : x \in V(\mathbb{K}), p(x) > 0\} > 0$.

DC Decomposition for MBLP_t

According to theorem 2.1, there exists $t_0 \geq 0$ such that (2.1) and (2.5) have the same optimal solution for $t \geq t_0$.

We can suggest the following DC decomposition for MBLP_t:

$$\begin{cases} g(x, y) = & \chi_K(x, y), \\ h(x, y) = & -c^T x - d^T y - tp(x). \end{cases} \tag{2.6}$$

The first sequence $(w_x^k, w_y^k) \in \partial(h(x^k, y^k))$ is calculated as

$$\begin{cases} w_{x_i}^k = -c_i - t(1 - 2x_i^k), & \forall i \in \{1, ..., n\}. \\ w_{y_j}^k = -d_j, & \forall j \in \{1, ..., p\}. \end{cases} \tag{2.7}$$

The second sequence is

$$(x^{k+1}, y^{k+1}) \in argmin\,\{-\langle\,(x,y),(w_x^k, w_y^k)\,\rangle\colon (x,y) \in K\}\,. \tag{2.8}$$

The DCA scheme for solving MBLP can be described as follows

Algorithm 2.1 DCA for solving MBLP

Initialization : (x^0, y^0), $k = 0$, $\epsilon > 0$, $\epsilon_2 > 0$, $t > 0$, $\theta > 0$.
1. Compute $(w_x^k, w_y^k) \in \partial(h(x^k, y^k))$ via (2.7).
2. Compute (x^{k+1}, y^{k+1}) by solving (2.8).
3. If $(|f(x^{k+1}, y^{k+1}) - f(x^k, y^k)| \leq \epsilon(|f(x^k, y^k)| + 1)$, then return (x^k, y^k).
4. Otherwise,
4.1. If$(p(x^{k+1} \geq \epsilon_2)$, then $t \leftarrow t + \theta$.
4.2. $k \leftarrow k + 1$.
4.3. go to 1.

Convergence properties of DCA for MBLP can be found in Le Thi and Pham Dinh (2001). For instance, it is worthwhile to note that, if at iteration r of DCA, x^r is integer, then x^k is integer for all $k \geq r$. This property is very interesting and useful for the cooperative VNS-DCA scheme.

2.2.2 VNS for solving MBLP

VNS is a standard metaheuristic method for MBLP. Two main points to be determined in VNS are local search methods and neighborhood structures. Several local search methods can be taken on for MBLP such as k-opt, another metaheuristic, complementing a value of a binary variable, etc. As for neighborhood structures, suppose that we have a solution (x, y), we can use the idea of generating a new solution $(x^{'}, y^{'})$, such that:

$$\Sigma_i|x_i - x_i^{'}| + \Sigma_j|y_j - y_j^{'}| \geq MinVal. \tag{2.9}$$

Or,

$$\Sigma_i|x_i - x_i^{'}| \geq MinVal. \tag{2.10}$$

2.3 VNS-DCA: a cooperative approach for solving MBLP

VNS-DCA is the brainstorming of VNS and DCA. The cooperative scheme is inspired by the Metastorming approach Yagouni and Le Thi (2014) in which only metaheuristics have been used. In the literature, there exist many paradigms of parallel programs. For

our approach, we opted for the Client-Server model. As for exchanging and distributing information between processes, we use the two-sided point-to-point communication of the MPI library.
We are now in a position to describe VNS-DCA. An illustrative scheme of the design of VNS-DCA is found in Fig. 2.1).

• **Step 0:** all the processes read data and then the server generates an initial solution. After that, it distributes the solution (the value of the objective function and the variables) to the clients. The exchange of the solution is executed using two-sided point-to-point communication. More precisely, we opted for the following two functions: MPI_Send and MPI_Recv.

• **Step 1:** the clients receive the solution and start running, in parallel.

• **Step 2:**
∗**Client 1** performs some iterations of DCA until it gets a binary solution. Then, it sends the objective function to the server.

∗**Client 2** performs one iteration of VNS. Then, it sends the objective function to the server.

• **Step 3:** the server receives the information. It determines the best client (the process having the best objective value) and the worst one (the process having the worst objective value) and distributes this information. If the stopping condition is satisfied, it gives the stop order to the two clients.

• **Step 4:** the clients receive information (best client and worst one).

•**Step 5:** if the stopping condition is not satisfied, the client which gets the best result will distribute the solution to the server and to the other client. These two last processes will receive the solution. Then, the clients restart running using the best current solution, from Step 2.

•**Step 6:** if the stopping condition is satisfied, the best client will send its solution to the server and all the processes will stop running.

2.4 Applications

The class of MBLP includes many combinatorial optimization problems, e.g., traveling salesman problem, knapsack problem, assignment problem, etc. To evaluate the efficiency of our approach, VNS-DCA is tested on a set of benchmarks of the capacitated facility location problem. In this section, we introduce the chosen problem and discuss the implementation of DCA, VNS as well as VNS-DCA to it.

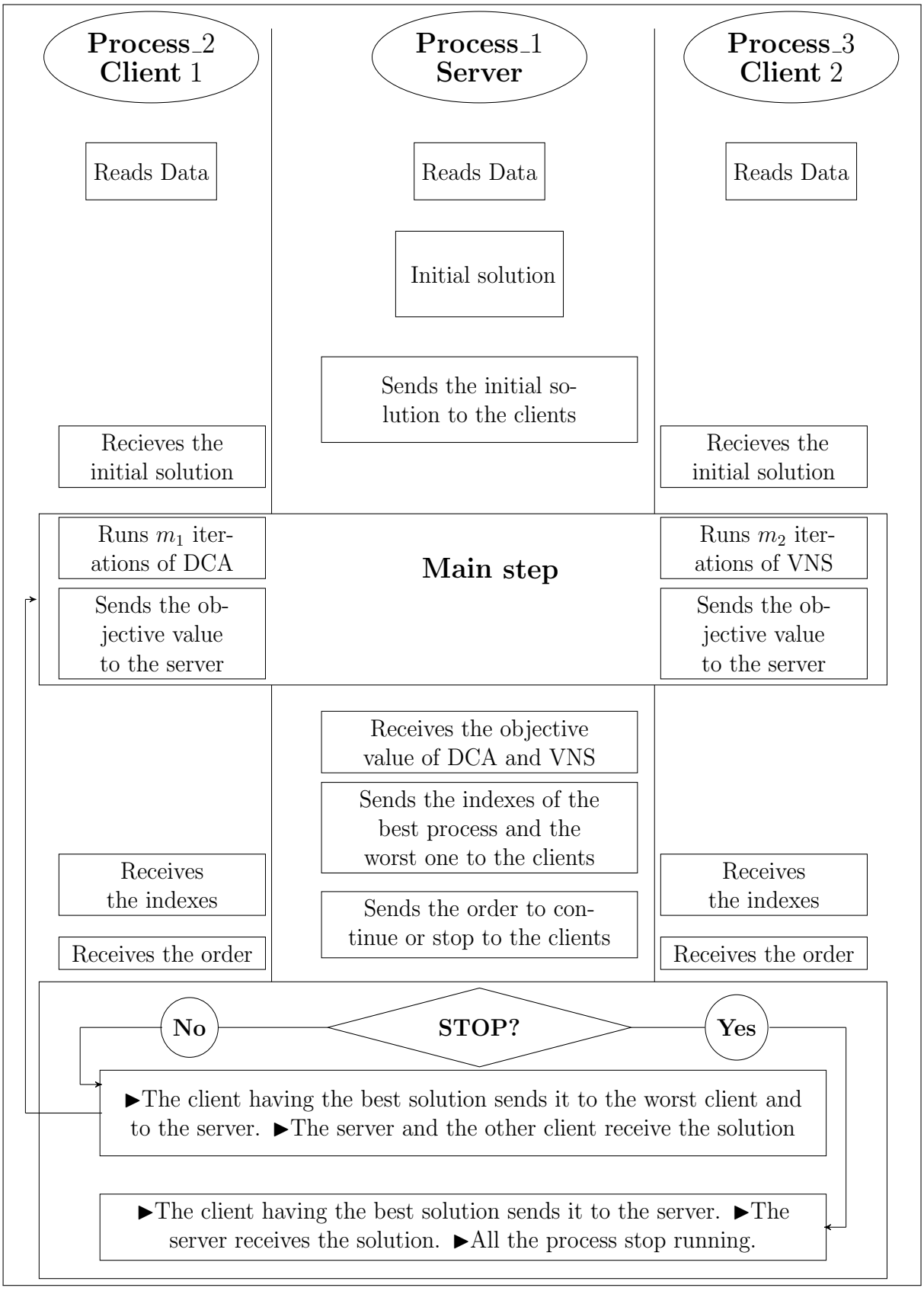


Figure 2.1 – VNS-DCA : cooperative scheme based on VNS and DCA.

2.4.1 CFLP

The Capacitated facility location problem K. Ahuja et al. (2003); Holmberg et al. (1999), which is an NP-hard combinatorial optimization problem, is composed of a set of m facilities, and a second one is constituted by n customers. Each facility has a location and a limited capacity. However, a customer has only a demand. In the CFLP, we distinguish two types of costs. The opening cost of the facility as well as the connection cost between facilities and customers. The objective of the CFLP is assigning customers to facilities with respect to capacity constraints, in such a way that the total cost is the minimum. Before talking about the mathematical model of CFLP we will give notations and decision variables.

m: the number of facilities.
n: the number of customers.

$$x_{ij} = \begin{cases} 1 & if\ customer\ j\ is\ assigned\ to\ facility\ i, \\ 0 & otherwise. \end{cases}$$

$$y_i := \begin{cases} 1 & if\ facility\ i\ is\ opened, \\ 0 & otherwise. \end{cases}$$

c_{ij}: the assignment cost of customer j to facility i.

f_i: the opening cost of facility i.

d_{ij}: the demand for customer j from facility i.

q_i: the amount of product available in facility i (the capacity of facility i).

$\forall i := 1, ..., m$, $\forall j = 1, ..., n$.

The CFLP can be formulated as follows.

$$(CFLP) \begin{cases} \min Z = \Sigma_{i=1}^{m}\Sigma_{j=1}^{n} c_{ij}x_{ij} + \Sigma_{i=1}^{m} f_i y_i, & \\ \Sigma_{i=1}^{m} x_{ij} = 1, & \forall j = 1, ..., n, \\ \Sigma_{j=1}^{n} d_{ij}x_{ij} \leq q_i y_i, & \forall i = 1, ..., m, \\ x_{ij} \leq y_i, & \forall i = 1, ..., m, \forall j = 1, ..., n, \\ x_{ij}, y_i \in \{0, 1\}, & \forall i = 1, ..., m, \forall j = 1, ..., n. \end{cases} \tag{2.11}$$

2.4.2 A constructive heuristic for VNS

To start VNS, an initial solution is required. Thus, in this section, we propose a constructive heuristic for CFLP. We denote with:

I= $\{1, 2, ..., m\}$, the set of facilities.
J= $\{1, 2, ..., n\}$, the set of customers.
In the beginning, we agree that:

$$x_{ij} := 0 \ and \ y_i := 0 \ \forall i \in I, \ \forall j \in J \tag{2.12}$$

This heuristic is inspired from the nearest neighborhood search, the difference remains in the starting step. Therefore, the idea is to choose the facility (i) and the customer (j) which requires the minimum value of connection cost matrix. We assign the customer (j) to the facility (i) and we continue with the last one. For facility (i), we assign customers until it will be saturated, starting with the one who has the smallest connection cost. We give the pseudo-code of the heuristic in Algorithm 2.2.

Algorithm 2.2 H0: a constructive heuristic for solving the CFLP

1. While$(J \neq \phi)$, do
2. Choose $(i, j) \in I \times J$ having the minimum connection cost.
3. Assign j to i, update J and the capacity of the facility i.

repeat

4. Assign customers in J to facility i, starting with the customers having the smallest connection cost and satisfying the capacity constraint.
5. Update J and the capacity of the facility i.

until Facility i is saturated.

2.4.3 Adapting DC programming and DCA to CFLP

We denote with S_t the set of feasible solutions of the relaxed model of CFLP.

$$S_t = \left\{ \begin{array}{c} (x, y) \in R^{m*n} \times R^n | \Sigma_{i=1}^m x_{ij} = 1, \Sigma_{j=1}^n d_{ij} x_{ij} \leq q_i y_i, x_{ij} \leq y_i, \\ \\ x_{ij}, y_i \in [0, 1], \forall i = 1, ..., m, \forall j = 1, ..., n \end{array} \right\}. \tag{2.13}$$

To treat CFLP by DCA we transform it into a continuous concave problem using the exact penalty as in (2.2.1). In our case, we use the following penalty function,

$$p(x, y) = \frac{1}{2} \Sigma_{i=1}^m \Sigma_{j=1}^n x_{ij}(1 - x_{ij}) + \frac{1}{2} \Sigma_{i=1}^m y_i(1 - y_i). \tag{2.14}$$

The resulting penalized program of CFLP, denoted CFLP_t can be expressed by the following formulation

$$CFLP_t = \min\left\{F_t(x,y) = \Sigma_{i=1}^{m}\Sigma_{j=1}^{n} c_{ij}x_{ij} + \Sigma_{i=1}^{m} f_i y_i + tp(x,y) | (x,y) \in S_t\right\}. \quad (2.15)$$

In accordance with (2.1), there exists $t_0 \geq 0$ such that for $t \geq t_0$, CFLP and CFLP_t are equivalent, they have the same feasible solutions as well as the optimal ones. We apply DCA to the continuous version CFLP_t.

For the DC decomposition of CFLP, we propose the following DC components.

$$\begin{cases} g(x,y) = & \chi_K(x,y), \\ \\ h(x,y) = & -\Sigma_{i=1}^{m}\Sigma_{j=1}^{n} c_{ij}x_{ij} - \Sigma_{i=1}^{m} f_i y_i - tp(x,y). \end{cases}$$

To apply DCA on the CFLP, we have to compute two sequences (w_x^k, w_y^k) and $(h(x^k, y^k))$. First, $(w_x^k, w_y^k) \in \partial(h(x^k, y^k))$ can be computed as follows

$$\begin{cases} w_x^k = & -\Sigma_{i=1}^{m}\Sigma_{j=1}^{n} c_{ij} - \frac{t}{2}\Sigma_{i=1}^{m}\Sigma_{j=1}^{n}(1 - 2x_{ij}), \\ \\ w_y^k = & -\Sigma_{i=1}^{m} f_i - \frac{t}{2}\Sigma_{i=1}(1 - 2y_i), \end{cases} \quad (2.16)$$

and (x^{k+1}, y^{k+1}) by solving

$$\min\left\{-\langle (x,y), (w_x^k, w_y^k)\rangle : (x,y) \in K\right\}. \quad (2.17)$$

The hardest point while applying DCA to any problem with a binary variable is getting a null penalty function (p). To get binary variables, we will use the technique of updating the value of the penalty parameter in a case p is not null or greater than a certain value of ϵ. Therefore, we describe the DCA for solving CFLP_t as follows (see, Algorithm 2.3.)

Algorithm 2.3 DCA1: DCA for solving CFLP

Input: (x^0, y^0), $k = 0$, $\epsilon_1 > 0$, $\epsilon_2 > 0$, t, δ;

1.Compute (w_x^k, w_y^k) via (2.16) ;
2. Compute (x^{k+1}, y^{k+1}) via (2.17);
3. If $(p(x^{k+1}, y^{k+1}) \geq \epsilon_2)$ then, $t := t + \delta$;
4. If $(|F_t(x^{k+1}, y^{k+1}) - F_t(x^k, y^k)| \leq \epsilon_1(|F_t(x^k, y^k)| + 1))$, then
4.1 Return (x^{k+1}, y^{k+1});
4.2 **STOP**;
5. Otherwise, $k := k + 1$; go to 1.

2.4.4 Adapting VNS to CFLP

To solve the CFLP by VNS, we adopt the same steps of the VNS algorithm in the chapter one. The difference here remains in the local search procedure and the neighborhood structures. Therefore, we use a new local search method which is defined below in Algorithm 2.4. As for the neighborhood structures, they can be the same as the ones defined by the equations 2.9 and 2.10. Another idea which can be used is to find all possible solutions when we re-assign k customers and we change the value of k customers for each neighborhood, etc.

Local search algorithm

The idea of the local search here is to repeat exchanging two customers assigned to two different facilities in such a way that this exchange will improve the solution until no improvement is possible. The choice of customers and facilities is random. An initial solution is needed to start this local search.

Algorithm 2.4 Local-Search

repeat

1. Choose randomly an opened facility i;
2. Choose randomly a customer j assigned to the facility i;
3. Choose randomly another opened facility k;
4. Find a customer l assigned to facility k, such that
   4.1 $c_{ij} + c_{kl} \geq c_{il} + c_{kj}$;
   4.2 $q_i + d_{ij} - d_{il} \geq 0$;
   4.3 $q_k + d_{kl} - d_{kj} \geq 0$;
5. Update
   5.1 $x_{ij} = 0$; $x_{il} = 1$;
   5.2 $q_i = q_i + d_{ij} - d_{il}$;
   5.3 $x_{kl} = 0$; $x_{kj} = 1$;
   5.4 $q_k = q_k + d_{kl} - d_{kj}$;

until no improvement is possible.

2.4.5 Adapting the cooperative approach to CFLP

Some changes are required to make VNS-DCA fit the CFLP. We have explained how to adapt the component algorithms (DCA and VNS) to CFLP. Now we are in a position to discuss DCA-VNS-CFLP, the version of DCA-VNS adapted to CFLP. In Fig. 2.2. the DCA-VNS-CFLP scheme is illustrated and the different steps can be summarized as follows

1. Reading data and initializing the different variables. In what follows we give the processes and the corresponding variables to each one of them:
   the server: ε_1, client 1: ε_2, t, and θ, client 2: K_{max} and $N_{k_{VNS}}$.

2. The server generates an initial solution CFLP by a constructive heuristic H0 (see, Algorithm 2.2.) and then improve it by applying VNS.
3. The server sends the solution to the clients using the function MPI_Send.
4. The clients receive the solution by calling the function MPI_Recv.
5. Client 1 Applies some iterations of DCA1 2.3 for CFLP until a binary solution is found and distributes its solution to the other processes.
6. Client 2 Applies one iteration VNS and distributes its solution to the other processes.
7. The server makes an evaluation and sends the order to stop or continue running.

2.5 Numerical results

The proposed approach VNS-DCA-CFLP was implemented using C++ and compiled within Microsoft Visual Studio 2017. Experiments were carried out on a Dell desktop computer with Intel Core(TM) i5-6600 CPU 3.30GHz and 8GB RAM under Windows 10. Linear programs were solved using the software CPLEX version 12.6. The parameters of CPLEX solver were set at their default. The instances of CFLP used to perform VNS-DCA-CFLP, are available at Benchmarks library [2]. In the tables 2.1. and 2.2., dataset consists of 20 instances having the same size ($n = 100$, $m = 100$).

For each instance, we run the algorithms 5 times and take the best solution within 5 runs. We compare VNS-DCA-CFLP with the results of CPLEX, DCA1, and VNS as indicated in tables 2.1. and 2.2. More specifically, table 2.1. shows the objective value and the gap obtained by VNS-DCA, DCA, VNS. Here the "Gap" of Algorithm (A) is defined by

$$Gap = \frac{|Objective\ value\ given\ by\ (A) - Optimal\ value\ (given\ by\ CPLEX)|}{Objective\ value\ given\ by\ (A)} * 100\%.$$

Table 2.2. provides the corresponding running time in seconds taken by each algorithm to get these solutions, and "Ratio" of the algorithm (A) is calculated as

$$Ratio(A) = \frac{CPU(CPLEX)}{CPU(A)}.$$

From the numerical experiments, we observe that

- The solution given by the cooperative approach VNS-DCA-CFLP is very close to the optimal solution.
- In all 20 instances, the gap is less than 2.7% and the average gap is 0.6%. In particular, the gap is less than 1% in 14/20 instances and 0% in two instances.

2. http://www.math.nsc.ru/AP/benchmarks/CFLP/cflp-eng.html

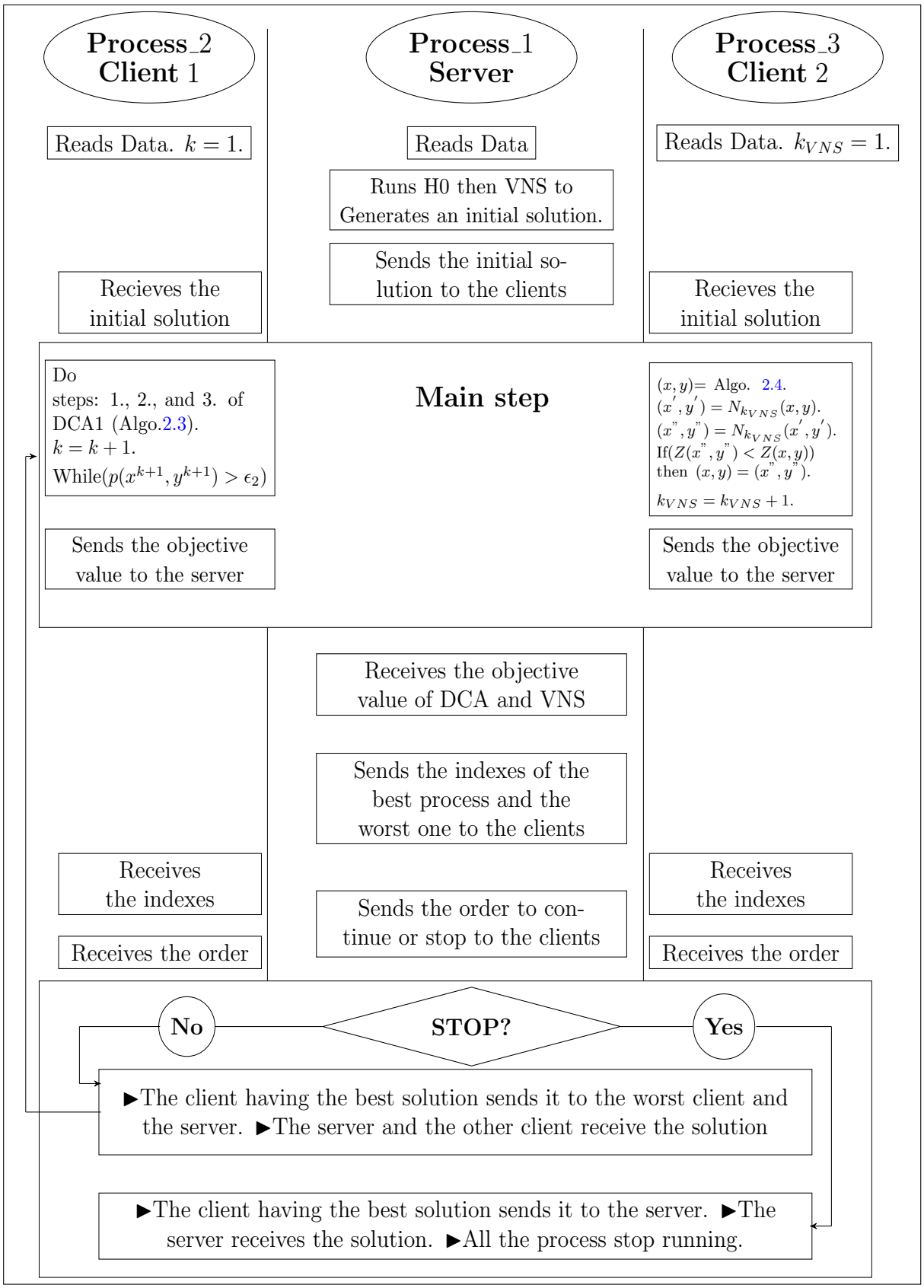


Figure 2.2 – VNS-DCA-CFLP: cooperative scheme based on VNS and DCA for solving CFLP (the best solution is the one having the best objective value).

Code	Objective value						
	VNS-DCA-CFLP	Gap%	DCA1	Gap%	VNS	Gap%	CPLEX
1	1880.6	0.04	1882	0.11	2555	26.42	1879.9
2	1950.5	0.00	1991.3	2.05	2485	21.51	1950.5
3	1997.6	0.39	2700.4	26.31	2457	19.01	1989.9
4	1987.6	0.68	2024.6	2.50	2685	26.48	1974
5	1868	0.42	1868	0.42	2687	30.77	1860.2
6	2054.8	0.83	2165.1	5.88	2608	21.86	2037.8
7	1979.1	0.18	2027	2.54	2693	26.64	1975.6
8	1923.4	1.34	2458.9	22.82	2653	28.47	1897.7
9	1945.8	1.55	1969.6	2.74	2485	22.91	1915.7
10	1990	0.01	1991.4	0.08	2638	24.57	1989.9
11	1963.5	1.27	2088.1	7.16	2393	18.99	1938.5
12	2019.7	2.69	2484.7	20.90	2499	21.36	1965.3
13	1920.7	0.11	2454.7	21.84	2518	23.81	1918.5
14	1995.9	0.01	2104.4	5.16	2669	25.22	1995.8
15	1887.6	0.42	1892.5	0.68	2678	29.81	1879.7
16	1960.5	0.35	2000.4	2.33	2591	24.60	1953.7
17	2029.7	0.78	2040.6	1.31	2803	28.15	2013.9
18	1993.8	0.98	2057.3	4.04	2647	25.42	1974.2
19	1915.3	0.00	1918.5	0.17	2626	27.06	1915.3
20	1939.3	0.01	2580.3	24.85	2761	29.76	1939.2
Average:		0.60		7.69		25.14	

Table 2.1 – Comparison of the objective values obtained by VNS-DCA-CFLP, DCA1, VNS, and CPLEX.

Code	CPU(s)						
	VNS-DCA-CFLP	Ratio	DCA1	Ratio	VNS	Ratio	CPLEX
1	40	1.5	4	14.5	102	0.57	58
2	21	1.3	4	7	102	0.27	28
3	32	2.2	0.61	113.11	107	0.64	69
4	17	1.2	4	5	41	0.49	20
5	26	0.8	9	2.22	76	0.26	20
6	26	3.8	2	49	129	0.76	98
7	28	122.8	7	491.14	67	51.31	3438
8	37	2.3	1	86	67	1.28	86
9	60	0.7	5	7.8	92	0.42	39
10	27	2.5	2	34	70	0.97	68
11	51	0.5	2	14	160	0.18	28
12	36	1.9	1	67	102	0.66	67
13	76	1	0.86	90.70	70	1.11	78
14	22	3.1	2	34.5	52	1.33	69
15	24	0.8	5	3.8	74	0.26	19
16	31	1.3	1	39	111	0.35	39
17	44	3.5	3	50.67	63	2.41	152
18	28	5.8	1	161	69	2.33	161
19	19	0.9	3	6	63	0.29	18
20	22	2.5	0.61	90.16	80	0.69	55
Average:	33.35	8.01		68.33		3.33	230.5

Table 2.2 – Comparison between CPU time of VNS-DCA-CFLP, DCA1, VNS, and CPLEX.

- As for the component algorithms, DCA1 is much better than VNS: the average gap is 7.69% versus 25.14%.

Regarding the running time,

- DCA1 is the fastest and VNS is the slowest.
- It can be seen that VNS-DCA-CFLP is faster than CPLEX in 15/20 instances. In particular, the seventh dataset, VNS-DCA-CFLP runs approximately 123 times faster than CPLEX
- In terms of the average time, the CPU time is 33.35 seconds for VNS-DCA-CFLP and 230.50 seconds for CPLEX.

The disadvantage of the cooperative approach,

- VNS is much slower than DCA1. Consequently, the cooperative approach consuming much more time than DCA1, as a waiting time is required for DCA1 in the cooperative scheme.

2.6 Conclusion

We have presented in this chapter a new cooperative approach named VNS-DCA based on DCA and VNS for solving MBLP. Our approach is developed using the techniques of parallel and distributed programming. The component algorithms of the approach cooperate by exchanging information between each other via functions of the two-sided point-to-point communication. To evaluate the efficiency of our scheme, the CFLP was considered and VNS-DCA was adapted to it, and VNS-DCA-CFLP is the cooperative approach for solving the CFLP. The numerical experiments confirm the efficiency of the proposed approach. Through extensive numerical analysis, it can be seen that the results obtained by the cooperation VNS-DCA-CFLP are satisfactory. The technique introduced in this chapter can be extended to cooperate with more methods for solving larger classes of problems.

Chapter 3

Cooperative methods for BQP

Abstract: *In this chapter, we present a new Cooperative scheme for solving the binary quadratic programming problem (BQP). This approach is based on a deterministic algorithm and two metaheuristics. Hence, we opted for the DCA-like, a new version of DCA, genetic algorithm and migrating birds optimization. These algorithms start in parallel and cooperate by exchanging solutions via specific functions belong to the one-sided collective communication of the MPI library, at the end of each cycle. The adopted function of communication allows broadcasting one message by a process to all the others at the same time. Experiment results on several instances of the quadratic assignment problem prove the efficiency of our approach and obviously illustrate the cooperation between the component algorithms.*

1. The material of this chapter is developed from the following work:
[1]. Samir Sara, Le Thi Hoai An, Yagouni Mohammed. DCA-Like, GA and MBO: A Novel Hybrid Approach for Binary Quadratic Programs. In: Le Thi H., Le H., Pham Dinh T. (eds) Optimization of Complex Systems: Theory, Models, Algorithms and Applications. WCGO 2019. Advances in Intelligent Systems and Computing,Springer, Cham, vol 991, pp. 299-309 (2020).

3.1 Introduction

The binary quadratic programs are NP-hard combinatorial optimization problems and frequently used in operational research. We consider the following mathematical formulation:

$$(BQP)\begin{cases} \min Z(x) = x^T Qx + c^T x, \\ Ax = b, \\ Bx \leq b', \\ x \in \{0,1\}^n, \end{cases} \tag{3.1}$$

where $Q \in \mathbb{R}^{nn}$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{mn}$, $B \in \mathbb{R}^{pn}$, $b \in \mathbb{R}^m$, $b' \in \mathbb{R}^p$.

Related works

Many real-life problems can be formulated as BQP including scheduling, facility location, assignment, and knapsack. In order to find a global and/or local solution for BQP, many approaches have been considered. Several exact algorithms have been proposed: e.g., branch-and-bound Pardalos and Rodgers (1990), cutting planes Helmberg and Rendl (1998) and branch-and-cutRostami et al. (2018), to solve the BQP. Due to the computational burden posed by exact methods, researchers have widely studied metaheuristics, among them: genetic algorithms (see, e.g.,Holland (1992); Julstrom (2005); Merz and Freisleben (1999); Misevicius and Staneviciene (2018)), scatter search Glover (1977); Van-Dat Cung et al. (1997), ant colony optimization Dorigo and Di Caro (1999); Gambardella et al. (1999), tabu search Glover (1986); Misevicius (2005), variable neighborhood search Mladenović and Hansen (1997); Zhang et al. (2005), particle swarm optimization Eberhart and Kennedy (1995); Mamaghani and Meybodi (2012), Cuckoo Search Dejam et al. (2012); Yang and Suash Deb (2009), migrating birds optimization Duman et al. (2012). Since BQP is a nonconvex problem, we can not talk about the proposed solution methods in the literature without citing the DC programming and DCA which constitute the backbone of nonconvex optimization. Many authors have effectively applied DC programming and DCA to BQP (see, e.g.,Le Thi and Pham Dinh (2001, 2018); Pham Dinh et al. (2008)).

Contribution and motivation

The main contribution of this study to the literature lies in a new cooperative approach using DCA-like (a new version of DCA) and two other evolutionary metaheuristics which are the genetic algorithm and migrating birds optimization. DCA$_l$-Meta consists in combining the component algorithms (DCA-like, GA and MBO) in a cooperative way using parallel and distributed programming. The idea of our approach is heartened by the work of in Yagouni and Le Thi (2014) in which the collaboration has been applied by using a set of metaheuristic algorithms. The power of DCA, the efficiency of metaheuristics and their successful applications in combinatorial optimization gave rise to our choice of the participating algorithms of our cooperation. DCA-like, GA and MBO exchange information between each other by calling the function MPI_Bcast of the known communication protocol MPI (Message Passing Interface). MPI_Bcast

wich allows a one-sided collective communication, consists in broadcasting a message, which means that it permits sending the same message to many processes in parallel and at the same time. The shared information is composed either by the objective value, the solution or the stopping criterion. In order to asses the efficiency of our cooperative approach, we test on several datasets of the QAP (Quadratic Assignment Problem).

DCA has proved its efficiency for solving combinatorial optimization problems (e.g. Le Thi et al. (2014c); Le Thi and Pham Dinh (2001); Le Thi et al. (2014f); Pham Dinh et al. (2010)). In this area, the application of DC programming and DCA requires a reformulation of the problem to an equivalent continuous optimization problem by adopting the exact penalty techniques (see, Le Thi and Pham Dinh (1997, 2001); Le Thi et al. (2011); Pham Dinh et al. (2010)).

The process of natural selection gave rise to the famous metaheuristic known as the genetic algorithm. In combinatorial optimization, we can find excellent results of GA (e.g. Julstrom (2005); Merz and Freisleben (1999); Misevicius and Staneviciene (2018); Potvin (1996)).

The idea of migrating birds optimization presented by in Duman et al. (2011), is inspired by the V formation flight of migrating birds. Since its appearance, the MBO has furnished many important results in combinatorial optimization (see, Duman and Elikucuk (2013); Duman et al. (2011, 2012); Soto et al. (2016); Tongur and Ülker (2014), etc.).

The remainder of the chapter is organized as follows; In section 2, we will discuss all participating algorithms including DCA-like, GA, MBO and their application for solving BQP. Section 3 is devoted to a global description of the cooperative approach DCA$_l$-Meta for BQP. Section 4 presents an application of our approach to the quadratic assignment problem. Numerical results are given and discussed in Section 5 and Section 6 concludes this chapter.

3.2 DCA-like, GA and MBO for solving BQP

The cooperative approach is constructed from three algorithms. Each one of them has special characteristics. Being population-based metaheuristics, GA and MBO have some common points. However, DCA-like is a deterministic algorithm, it is totally different. First of all, we give a DC reformulation of BQP. Then the elaboration of genetic algorithms and MBO to the studied problem BQP.

3.2.1 A DCA-like scheme for BQP

Many works have tackled DCA for solving BQP (see, e.g., Le Thi and Pham Dinh (2001); Pham Dinh et al. (2010)). In order to apply DC programming and DCA, we

reformulate BQP as a quadratic program with continuous variables using an exact penalty function (see, e.g., Le Thi and Pham Dinh (2001); Le Thi et al. (1999, 2011); Pham Dinh et al. (2010)).

We can choose the following penalty function

$$p(x) = \Sigma_{i=1}^{m} x_i(1 - x_i). \tag{3.2}$$

Let

$$\Omega = \{x \in [0,1]^n : Ax = b, Bx \leq b^{'}\}. \tag{3.3}$$

BQP is equivalent to the following formulaion

$$\min \left\{ \; Z(x) = x^T Q x + c^T x : x \in \Omega, p(x) \leq 0 \; \right\}. \tag{3.4}$$

Thus, we can define BQP_t the penalty program of BQP

$$\min \left\{ \; F(x) = Z(x) + tp(x) : x \in \Omega \; \right\}. \tag{3.5}$$

According to the work in Le Thi et al. (2011), if there exists a number $t_0 > 0$ such that for all $t > t_0$, the problems (3.1) and (3.5) are equivalent in the sense that they have the same set of optimal solutions and the same optimal value.
We can express the problem (3.5) as follows:

$$\min \left\{ \; F(x) = G_\rho(x) - H_\rho(x) : x \in \mathbb{R}^n \; \right\}, \tag{3.6}$$

with $\rho > 0$, the DC components G_ρ and H_ρ can be given by

$$G_\rho(x) = \frac{\rho}{2}\|x\|^2 + \chi_\Omega(x) \tag{3.7}$$

and

$$H_\rho(x) = \frac{\rho}{2}\|x\|^2 - Z(x) - tp(x), \tag{3.8}$$

We can see that G_ρ is convex since Ω is convex. As for H_ρ, it is convex with ρ being larger than the spectral radius of $\nabla^2 Z(x)$, denoted $\rho\left(\nabla^2 Z(x)\right)$. The gradient of Z can be computed by

$$\nabla Z(x) = \left(\frac{\partial Z(x)}{\partial x_{ij}} \right), \tag{3.9}$$

where

$$\frac{\partial Z(x)}{\partial x_{ij}} = \sum_{p,q} \left(a_{ip} d_{jq} + a_{pi} d_{qj} \right) x_{pq}. \tag{3.10}$$

The Hessian of Z is computed by

$$\nabla^2 Z(x) = \left(\frac{\partial^2 Z(x)}{\partial x_{ij} \partial x_{pq}} \right), \tag{3.11}$$

where

$$\frac{\partial^2 Z(x)}{\partial x_{ij}\partial x_{pq}} = a_{ip}d_{jq} + a_{pi}d_{qj}. \tag{3.12}$$

Since $\rho\left(\nabla^2 Z(x)\right) \leq \|\nabla^2 Z(x)\|$, we can choose $\rho = \|\nabla^2 Z(x)\|$, where

$$\|\nabla^2 Z(x)\| = \sqrt{\sum_{i,j,p,q} \left(a_{ip}d_{jq} + a_{pi}d_{qj}\right)^2}. \tag{3.13}$$

However, $\rho = \|\nabla^2 Z(x)\|$ is quite large which can affect the convergence of DCA. Hence, we can update ρ as in the following DCA-like algorithm Le Thi et al. (2018):

Algorithm 3.1 DCA-Like for solving BQP

Initialization: Choose x^0, $\zeta \geq 1$, $\rho^0 = \frac{\varrho}{\zeta}$, $\eta_1 > 1$, $\eta_2 > 1$, t, t_{max}, $\xi \geq 1$ and $k \leftarrow 0$.

repeat

1. Compute $\rho = \max\{\rho^0, \frac{\rho^k}{\eta_1}\}$.
2. Compute $y^k = \nabla H_\rho(x^k)$ by

$$y^k = \rho x^k - \nabla Z(x^k) - te + 2tx^k, \tag{3.14}$$

where e is the one vector, i.e., all elements are 1.

3. Compute $\tilde{\text{x}} \in argmin\{\frac{\varrho}{2}\|x\|^2 - \langle y^k, x\rangle : x \in \Omega\}$.
4. **While** $H_\rho(\tilde{\text{x}}) < H_{\rho_k}(x^k) + \langle \tilde{\text{x}} - x^k, y^k\rangle$ do

4.1 $\rho := \eta_2\rho$.

4.2 Update y^k and $\tilde{\text{x}}$ by STEPS 2. and 3.

End.

5. Set $k \leftarrow k + 1$.
6. Set $x^k :=\tilde{\text{x}}$.
7. Set $\rho^k := \rho$.
8. If$(t < t_{max})$, $t := t * \xi$.

until Stopping criterion.

3.2.2 GA and MBO schemes for BQP

- GA: First, we start by coding our solution. For this purpose, a solution is schematized by a table of n integers. Then, the initial population is randomly generated. For the selection operator, we use a large enough probability. The crossover is done in k points (here we use k equals to one). As for the mutation operator, we opt for a small probability and it can be effected by changing a randomly chosen value. Finally, we preserve the best chromosomes to the next generation.

- MBO: A bird is schematized in the same manner as a chromosome in the GA. The initial birds are randomly generated. A neighbor is generated by exchanging the values of two indexes in the table presenting a bird.

3.3 DCA$_l$-Meta: a cooperative approach

DCA$_l$-Meta is inspired by the Metastorming approach Yagouni and Le Thi (2014). It presents the brainstorming of metaheuristics which means that it combines several metaheuristics. Metastorming uses many processes. One of them is called the animator which organizes the work of the other processes. As for the role of these last, it consists in running the metaheuristics in parallel and simultaneously.

In DCA$_l$-Meta, the shared information is either the solution, the stopping conditions or the objective value. The participant members are the component algorithms. The unique difference between Metastorming and DCA$_l$-Meta resides in the animator which we don't use in our study. In order to organize the work of the component algorithms without needing an animator, they communicate directly with each other.

The idea of DCA$_l$-Meta is to make a storm between the deterministic algorithm DCA-like and the two other evolutionary metaheuristics (GA and MBO). DCA$_l$-Meta is based mainly on the cooperation principle. The component algorithms work in parallel to solve the problems. They also exchange and distribute information between each other. The cooperation between DCA-like, GA and MBO is achievable despite the fact that DCA-like is a continuous algorithm because this last can get a good zero-one solution. The implementation of a Metastorming approach or DCA$_l$-Meta needs a parallel and distributed programming model. In DCA$_l$-Meta, we use the Client-Server model, in which two clients run two metaheuristics and the server runs the DCA-like (no animator used). As for the communication between the server and the clients we opt for the the one-sided collective communication. Our approach starts by an initialization step, then it goes through two other phases. We first outline the detail and the different operation of DCA$_l$-Meta. After that, we show its design by an illustrative scheme in Fig. 3.1. Now we are in the position to outset the procedure of DCA$_l$-Meta.

3.3.1 Different steps of DCA$_l$-Meta

Step 1: parallel initialization.

In this phase, we initialize our algorithms basing mainly on the randomized generation. Indeed, the server creates a random initial point for DCA-like. Then, it computes the value of the variable

$$\rho = \|\nabla^2 Z(x)\|$$

following the formula (3.13). As for the clients running the metaheuristics, each one of them will randomly generate an initial population. In order to diversify, the initial population of GA is different from that generated for MBO. After that point, DCA$_l$-Meta gets into a cycle which is constituted by two phases.

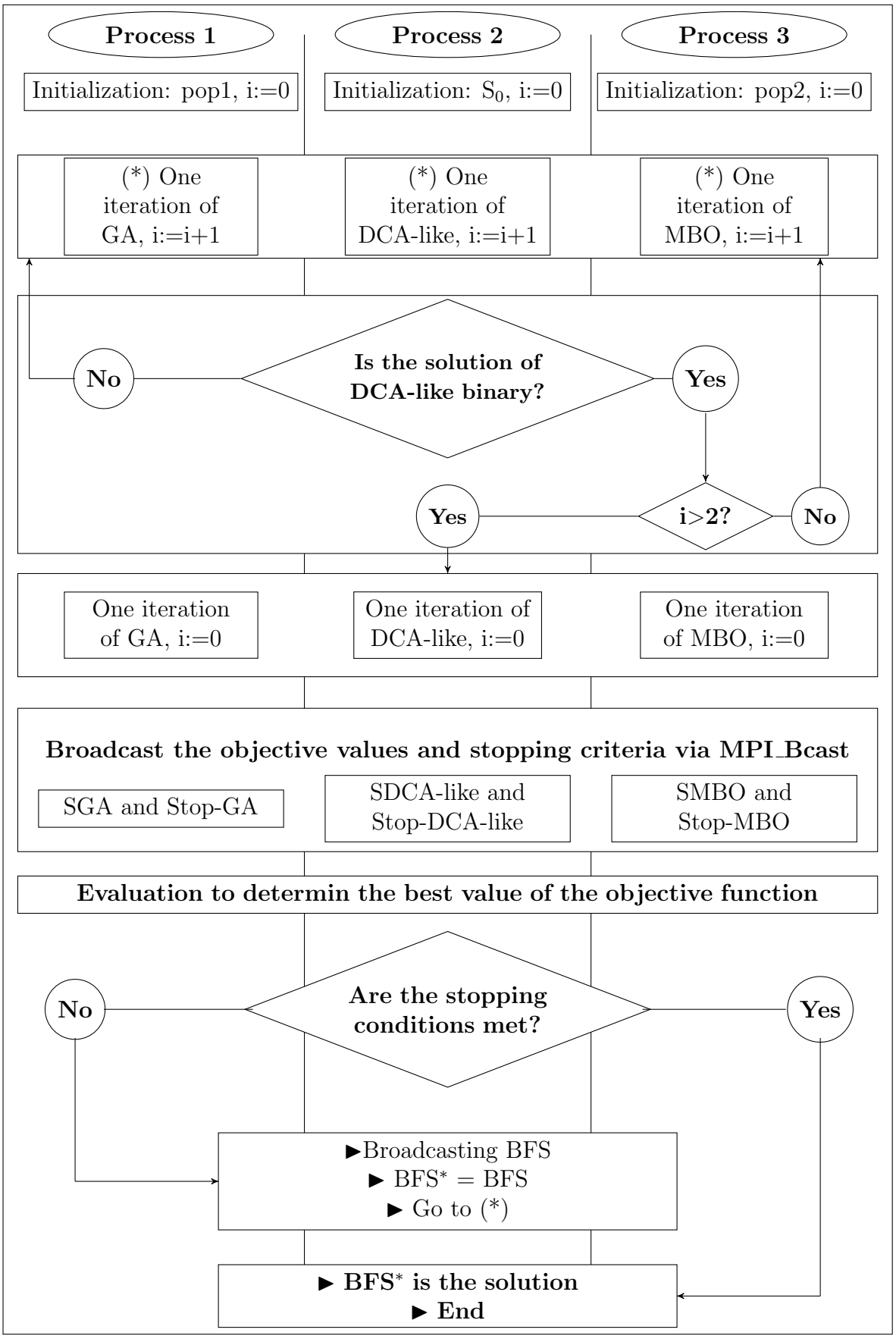


Figure 3.1 – DCA$_l$-Meta: cooperative scheme based on DCA-like, GA, and MBO. (BFS = Best Found solution).

Step 2: parallel and distributed running (the first phase of a cycle).

This step is considered as the main part of our approach since it is constituted by a parallel running and distributed information. All the component algorithms perform one iteration in parallel. Then, the server distributes via the function MPI_Bcast, the order to the clients either to run a new iteration or to passe to the next step. The decision of the server depends on the result obtained by DCA-like. More precisely, getting a zero-one solution allows the approach to initiate the succeeding phase and if it does not, the server broadcasts the order to start a new iteration. In the case that the clients accomplish their iterations before the server, they must wait until this last gives an order.

Remark 3.1.
- *The next phase starts when DCA-like finds a binary solution so as to make possible the exchange of the solution.*
- *A cycle is composed of two iterations at least.*
- *The communication between different processes is carried out via the message passing interface library.*

Step 3: parallel cooperation and evaluation (the second phase of a cycle).

All the component algorithms cooperate by exchanging between each other the objective values (SDCA-like, SGA, and SMBO) obtained by DCA-like, GA, and MBO. The stopping conditions (Stop-DCA-like, Stop-GA, and Stop-MBO) constitute part of the broadcasted information, as well. After that, a parallel evaluation is performed to determine the algorithm which finds the best solution. The end of DCA$_l$-Meta is determined by the condition that all the participating algorithms satisfy the stopping criterion. In this case, the final solution is BFS* (see, Fig. 3.1). If the stopping criterion is not satisfied, the best solution (BFS) will be distributed to be used by DCA-like as an initial solution,by GA a new chromosome, or by MBO as new bird. After that, DCA$_l$-Meta restarts the parallel and distributed running, which means a new cycle.

Remark 3.2. *In order to save time, the algorithms exchange first just the value of the objective function. After the evaluation, the best one distributes its solution. Hence, we avoid sharing useless information.*

The advantage of this approach is to exploit and combine the strengths and weaknesses of the component algorithms as in the real brainstorming to get a good result. This technique allows changing the behavior of the component algorithms while exploring the research space to browse more promising regions to get better solutions.

3.4 Application of DCA$_l$-Meta to the QAP

To assess the efficiency of DCA$_l$-Meta, we consider the quadratic assignment problem (QAP) to perform the tests. The QAP which is a very special binary quadratic problem, is composed of a set of n facilities and a set of n locations. The objective is to assign the facilities to the locations in such a way that the total cost is minimized. The assignment cost is a function of the distance between the locations and flow between the facilities, plus the costs associated with the corresponding flows of placing a facility at a certain location. The mathematical formulation of QAP is given by

$$(QAP)\begin{cases} \min F(x) = \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\Sigma_{p=1}^{n}\Sigma_{q=1}^{n} A_{ip}D_{jq}x_{ij}x_{pq}, \\ \Sigma_{i=1}^{n} x_{ij} = 1, j = 1, ..., n, \\ \Sigma_{j=1}^{n} x_{ij} = 1, i = 1, ..., n, \\ x \in \{0,1\}^{n\times n}, \end{cases} \tag{3.15}$$

where $A \in {\mathbb{R}_+}^{nn}$ and $D \in {\mathbb{R}_+}^{nn}$.

3.4.1 Elaborating the solution method to QAP

- **DCA-Like**
  We first replace $Z(x)$ in 3.1 by $F(x)$ in 3.15. Then, we apply the DCA-Like given in algorithms 4.1.

- **GA and MBO**
  We use the same steps as in section 3.2.2. The unique difference remains in how to schematize a solution (a chromosome and/or a bird). More precisely and following the mathematical model of the QAP, a solution is a table of two dimensions. The table is composed of n rows. Each one contains a location, and the facilities affected to it.

- **DCA$_l$-Meta**
  Here we follow the same steps in Fig. 3.1.

3.4.2 Stopping conditions

The stopping conditions that we use for each algorithm in DCA$_l$-Meta approach are expressed in the following equations (by taking $\epsilon = 10^{-4}$).

- DCA-like
$$|SDCA_{like}^{k} - F(BFS^{k-1})| \leq \epsilon. \tag{3.16}$$
- GA
$$|SGA^{k} - F(BFS^{k-1})| \leq \epsilon, \tag{3.17}$$
and
$$|SGA^{k} - SGA^{k-1}| \leq \epsilon, \tag{3.18}$$

- MBO

$$|SMBO^k - F(BFS^{k-1})| \leq \epsilon. \tag{3.19}$$

The second condition for MBO is that each bird must lead the population at least one time.

3.5 Numerical results

In this section, we describe the results produced by our cooperative approch on sixteen instances of the QAP, as well as a comparison between DCA$_l$-Meta and the component algorithms.

We used C++ to code our approach and we compiled it within Microsoft Visual Studio 2017. As for solving the convex quadratic sub-problem of DCA-like we chose the software CPLEX version 12.6. All experiments were carried out on a Dell desktop computer with Intel Core(TM) i5-6600 CPU 3.30GHz and 8GB RAM under Windows 10.

Our experiments are composed of two parts. In the first, we evaluate the performance of DCA$_l$-Meta by comparing it with DCA-like, GA, and MBO. In the second part, we study the detailed behaviors of the component algorithms in the cooperation.

3.5.1 Comparative results

This part of the numerical experiment is devoted to discussing the performance of the following algorithms: DCA-like, GA, MBO, and the cooperation DCA$_l$-Meta. The solutions obtained by the previous methods are compared to the best known solution (BKS) which is taken from (A Quadratic Assignment Problem Library [2]) in OR-Library [3]. The comparison is realized on three criteria: the objective value, the computation time (measured in seconds), the gap between each algorithm, as well as the best solution. The gap of an algorithm (A) is computed by the following formulation

$$Gap = \frac{|Objective\ value\ given\ by\ (A) - BKS|}{Objective\ value\ given\ by\ (A)} * 100\%. \tag{3.20}$$

We tested the four algorithms on several datasets of QAP taken from the Quadratic Assignment Problem Library. The summary of the comparative results of the four approaches is reported in table 3.2. and table 3.1. In each table, we first give information about the dataset (ID, size, and BKS). Then, the solution, the gap and the CPU of DCA$_l$-Meta, DCA-like, GA and MBO.
From the numerical experiences, we observe that:

2. http://anjos.mgi.polymtl.ca/qaplib//inst.html
3. http://people.brunel.ac.uk/~mastjjb/jeb/info.html

1. In terms of the objective function value and the gap: In general, DCA$_l$-Meta is comparable with MBO and both are better than DCA-like which is superior to GA. More precisely, DCA$_l$-Meta yields the best objective value in 13/16 cases, MBO in 11/16 cases, DCA-like 3/16 times, and GA one time. As for the gap:

   - $\text{Gap}_{DCA_l-Meta} \in [0, 1.79]\%$ (0.45% in the average).
   - $\text{Gap}_{MBO} \in [0, 9.71]\%$ (1.25% in the average).
   - $\text{Gap}_{DCA-like} \in [0, 27]\%$ (4.81% in the average).
   - $\text{Gap}_{GA} \in [0, 18.58]\%$ (2.98% in the average).

2. In terms of running time, GA and DCA-like are the fastest. MBO is faster than DCA$_l$-Meta. In the cooperation, the component algorithms exchange information with each other, which creates a waiting time. Regarding the quality of the solution found by the cooperation, we can say that this difference in time is very advantageous.

3. The cooperation between the component algorithms allows them to change the behavior and browse more promising regions to get better results.

3.5.2 Study of the behaviors of the component algorithms

In the tables from 3.3 to 3.18, we give the detailed behavior of the component algorithms while cooperating for solving sixteen instances of the QAP. More specifically, we are interested in the following aspects given in each table: the number of cycles, the number of iterations of each cycle, the cycle end time and the objective value obtained by DCA-like, GA and MBO. We also use the gap (computed by the formula (3.20)) to evaluate the quality of the solution.
Regarding the tables, it can be obviously seen that cooperation is harmoniously and perfectly realized which means that each component algorithm is important in this cooperation. For example, we take the problem in table 3.16. The MBO found the best solution in the two first cycles. In the third cycle, DCA-like improved the result by using this last one as an initial point. Until the 10^{th} cycle, the best solution has been found either by MBO or DCA-like. We observe that from the 11^{th} cycle the GA changes its behavior and becomes competitive with DCA-like and MBO. Hence, we can say that each of the component algorithms has an important role in this approach.

Dataset		Algorithm	Objective value	Gap%	CPU
Lipa40a	n = 40 BKS = 31538	DCA$_l$-Meta	**31876**	**1.06**	339.81
		DCA-like	31887	1.09	609.42
		GA	31942	1.26	**216.69**
		MBO	31887	1.09	609.42
Lipa50	n = 50 BKS = 62093	DCA$_l$-Meta	**62725**	**1.01**	1543.28
		DCA-like	62735	1.02	1449.88
		GA	62862	1.22	**1073.42**
		MBO	62735	1.02	1449.88
Lipa90	n = 90 BKS = 360630	DCA$_l$-Meta	362970	0.64	16027.00
		DCA-like	**362830**	**0.61**	9797.05
		GA	363276	0.73	**3533.52**
		MBO	362953	0.64	16618.70
Wil50	n = 50 BKS = 48816	DCA$_l$-Meta	48884	0.14	600.94
		DCA-like	**48824**	**0.02**	713.97
		GA	49144	0.67	**251.27**
		MBO	48846	0.06	2067.23
Sko100a	n = 100 BKS = 152002	DCA$_l$-Meta	153334	0.87	19537.80
		DCA-like	**152694**	**0.45**	39563.90
		GA	153700	1.10	**2154.76**
		MBO	154458	1.59	4982.69
Bur26a	n = 26 BKS = 5426670	DCA$_l$-Meta	**5426670**	**0.00**	1610.41
		DCA-like	5438650	0.22	**128.11**
		GA	5435200	0.16	**128.11**
		MBO	5431680	0.09	241.69
Bur26b	n = 26 BKS = 3817852	DCA$_l$-Meta	**3818730**	**0.02**	1074.46
		DCA-like	3826200	0.22	146.53
		GA	3826610	0.23	**67.82**
		MBO	**3818600**	**0.02**	123.21
Chr12a	n = 12 BKS = 9552	DCA$_l$-Meta	**9552**	**0.00**	112.43
		DCA-like	11418	16.34	17.65
		GA	10192	6.28	63.39
		MBO	**9552**	**0.00**	**12.55**

Table 3.1 – Comparison between DCA$_l$-Meta, DCA-Like, GA, and MBO (1).

Dataset		Algorithm	Objective value	Gap%	CPU
Chr20a	n = 20 BKS = 2192	DCA$_l$-Meta	**2232**	1.**79**	1310.45
		DCA-like	2624	16.46	**5.23**
		GA	2382	7.98	409.55
		MBO	2244	2.32	138.24
Chr25a	n = 25 BKS = 3796	DCA$_l$-Meta	**3796**	**0.00**	835.62
		DCA-like	5200	27	**62.66**
		GA	4662	18.58	2050.78
		MBO	4204	9.71	618.38
Rou20	n = 20 BKS = 725522	DCA$_l$-Meta	**726652**	**0.16**	206.69
		DCA-like	731436	0.81	**20.81**
		GA	737112	1.57	84.84
		MBO	734246	1.19	106.49
Nug12	n = 12 BKS = 578	DCA$_l$-Meta	**578**	**0.00**	167.31
		DCA-like	594	2.69	**8.65**
		GA	586	1.37	214.66
		MBO	**578**	**0.00**	12.10
Nug30	n = 30 BKS = 6124	DCA$_l$-Meta	**6128**	**0.07**	911.16
		DCA-like	6176	0.84	**131.47**
		GA	6212	1.42	718.38
		MBO	6148	0.39	367.87
Esc32d	n = 32 BKS = 200	DCA$_l$-Meta	**200**	**0.00**	368.00
		DCA-like	210	4.76	73.90
		GA	**200**	**0.00**	**59.84**
		MBO	**200**	**0.00**	135.66
Tai35b	n = 35 BKS = 283315445	DCA$_l$-Meta	**283315445**	**0.00**	3208.83
		DCA-like	**283315445**	**0.00**	295.11
		GA	287271000	1.38	**151.21**
		MBO	283335000	0.01	666.65
Ste36a	n = 36 BKS = 9526	DCA$_l$-Meta	**9658**	**1.37**	5926.28
		DCA-like	9960	4.36	398.52
		GA	9906	3.84	1368.50
		MBO	9702	1.81	**366.39**

Table 3.2 – Comparison between DCA$_l$-Meta, DCA-Like, GA, and MBO (2).

Chr12a					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	124	53.32	11782	**9552**	10192
2	138	112.43	**9552**	**9552**	**9552**
Gap%: 0.00%					

Table 3.3 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Chr12a).

Chr20a					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	224	181.48	2904	2688	**2566**
2	366	483.24	2566	**2318**	2566
3	360	791.86	2318	**2232**	2318
4	576	1310.45	**2232**	**2232**	**2232**
Gap%: 1.79%					

Table 3.4 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Chr20a).

Chr25a					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	119	133.25	5318	4902	**4756**
2	240	419.57	4756	**3884**	4756
3	165	623.00	3884	**3796**	3884
4	170	835.62	**3796**	**3796**	**3796**
Gap%: 0.00%					

Table 3.5 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Chr25a).

Nug12					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	248	130.22	762	**578**	582
2	80	167.31	**578**	**578**	**578**
Gap%: 0.00%					

Table 3.6 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Nug12).

Nug30					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	45	62.48	6660	6538	**6178**
2	323	489.45	**6128**	6154	6136
3	320	911.16	**6128**	**6128**	**6128**
		Gap%: 0.07%			

Table 3.7 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Nug30).

Esc32d					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	100	114.63	210	**204**	212
2	91	229.40	**200**	202	**200**
3	100	368.00	**200**	**200**	**200**
		Gap%: 0.00%			

Table 3.8 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Esc32d).

Bur26a					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	576	506.09	**5432000**	5432600	5432450
2	591	1088.13	5432000	**5426670**	5430460
3	585	1610.41	**5426670**	**5426670**	**5426670**
		Gap%: 0.00%			

Table 3.9 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Bur26a).

Bur26b					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	201	611.95	3829340	3827360	**3818730**
2	178	1074.46	**3818730**	**3818730**	**30818730**
		Gap%: 0.02%			

Table 3.10 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Bur26b).

Rou20					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	76	82.40	731070	**726652**	735306
2	113	206.69	**726652**	**726652**	**726652**
		Gap%: 0.16%			

Table 3.11 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Rou20).

Tai35b					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	28	287.38	306812000	295383000	**286160000**
2	48	896.13	285564000	284015000	**283889000**
3	57	1661.16	283824000	**283315445**	283817000
4	56	2425.38	**283315445**	**283315445**	**283315445**
5	57	3208.83	**283315445**	**283315445**	**283315445**
Gap%: 0.00%					

Table 3.12 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Tai35b).

Lipa40a					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	37	75.85	32210	32190	**31986**
2	51	151.34	**31927**	31963	**31927**
3	43	216.75	31927	31914	**31888**
4	39	276.31	31878	**31876**	**31876**
5	41	339.81	**31876**	**31876**	**31876**
Gap%: 1.06%					

Table 3.13 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Lipa40a).

Lipa50a					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	44	183.75	63003	63189	**62952**
2	50	368.80	62914	62921	**62852**
3	49	544.53	62809	62852	**62808**
4	48	771.52	**62762**	62808	62807
5	50	1023.55	**62737**	62754	62747
6	49	1279.51	62737	62737	**62725**
7	50	1543.28	**62725**	**62725**	**62725**
Gap%: 1.01%					

Table 3.14 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Lipa50a).

Wil50					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	44	172.58	49946	50792	**49740**
2	10	210.81	**49066**	49702	49604
3	6	231.17	**49052**	49066	49066
4	23	324.27	49052	49052	**49050**
5	21	407.38	**48886**	49050	49050
6	25	504.30	**48884**	48886	**48884**
7	25	600.94	**48884**	**48884**	**48884**
Gap%: 0.14%					

Table 3.15 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Wil50).

Lipa90a					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	2	215.29	367558	366374	**366216**
2	2	406.24	366216	366128	**365836**
3	22	1339.12	**363687**	365118	364323
4	8	1821.52	363687	363687	363687
5	13	2717.09	363687	363630	**363610**
6	12	3553.76	**363577**	363610	363582
7	12	4410.77	363577	363577	**363553**
8	2	4607.55	363553	363553	**363550**
9	20	5896.51	**363218**	363550	363498
10	14	6860.27	363218	363218	**363214**
11	15	7945.66	363193	**363154**	363162
12	10	8673.28	363154	363154	363154
13	12	9592.32	363154	363154	**363144**
14	11	10372.40	**363113**	**363113**	363123
15	2	10570.40	363113	**363094**	363113
16	17	11695.40	**363066**	363094	363067
17	2	11874.30	363066	363066	363066
18	15	12870.10	363066	363066	**363065**
19	2	13059.30	363065	**363049**	363065
20	20	14358.80	**363001**	363049	363012
21	11	15129.20	363001	363001	**362970**
22	13	16027.00	**362970**	**362970**	**362970**
Gap%: 0.64%					

Table 3.16 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Lipa90a).

Sko100					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	2	146.56	177760	172924	**171286**
2	3	488.18	**163432**	170830	168306
3	4	933.35	**158354**	163432	162338
4	6	1494.24	**155612**	158354	158112
5	8	2213.16	**154772**	155612	155572
6	7	2971.10	**154752**	154772	154756
7	4	3494.26	154736	154750	**154724**
8	7	4112.62	**154154**	154684	154724
9	7	4865.49	154154	154154	154154
10	6	5504.81	154154	154154	**154150**
11	5	5986.47	**153658**	154114	154118
12	2	6213.74	153658	153658	153658
13	2	6493.64	153658	153658	**153650**
14	2	6727.20	153650	153650	153650
15	3	7032.66	**153618**	153650	153650
18	12	8370.27	**153386**	153618	153606
24	2	9559.61	153386	153386	**153374**
39	2	10389.70	153374	153374	**153370**
49	2	12247.60	153370	153370	**153364**
53	2	13411.10	153364	153364	**153360**
67	2	16091.70	153360	153360	**153354**
68	2	16508.40	153354	153354	153354
74	2	17957.40	153354	153354	**153350**
75	2	18143.30	153350	153350	**153334**
82	1	19537.80	**153334**	**153334**	**153334**
Gap%: 0.87%					

Table 3.17 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: Sko100). (some steps are skipped because the the BFS was not improved).

ste36a					
Cycle	Iteration	CPU	DCA-like	GA	MBO
1	75	219.90	11442	10508	**10032**
2	246	855.31	**9896**	9900	9936
3	248	1542.43	9896	9844	**9828**
4	275	2332.19	9828	9828	9828
5	267	3102.86	9828	**9762**	9828
6	254	3852.41	9762	9762	9762
7	261	4525.87	9762	9762	9762
8	261	5245.73	9762	**9658**	9762
9	246	5926.28	**9658**	**9658**	**9658**
Gap%: 1.37 %					

Table 3.18 – Analysis of the behavior of component algorithms (approach: DCA$_l$Meta, dataset: ste36a).

3.6 Conclusion

We investigated a new cooperative approach for solving binary quadratic problems. It consists in cooperating DCA-like, GA, and MBO using parallel and distributed programming. The component algorithms exchange information by calling a function of the one-sided collective communication. As an application, we opted for the quadratic assignment problem. We have given a very brief outline of the application of DCA-like, GA, MBO, and DCA$_l$-Meta to the QAP. The numerical experiments on several datasets of QAP show that in terms of objective value the cooperative approach is the best, MBO is more efficient than DCA-like which is superior to GA. In terms of running time, GA and DCA-like are the fastest and MBO is faster than DCA$_l$-Meta. As for the cooperation principle, it is perfectly realized and all component algorithms were competitively cooperating.

Chapter 4

Cooperative methods for clustering

Abstract: *This chapter addresses the clustering problem. More precisely, we solve the Minimum-Sum-of-Squares Clustering problem using two cooperative approaches. We consider mathematical models of MSSC: a continuous bilevel programming model adapted by DCA and a discrete model for the metaheuristics. The first approach is a cooperation between DCA and two trajectory metaheuristics (TS and VNS). The second uses in addition to the previous algorithms, an evolutionary metaheuristic which is the MBO. The two cooperative schemes consist in exchanging, at each iteration, the solutions between the component algorithms and then starting the next iteration by the best one. The communication between the processes is effected via MPI_Put function (one-sided point-to-point communication) which allows a process to send a message, access the memory of the receiver process, and save the message without blocking the work of the receiver. A comparison between the two new approaches and the component algorithms (DCA, TS, VNS, and MBO) is investigated. Preliminary numerical simulations show that cooperative approaches work well for solving MSSC.*

4.1 Introduction

Clustering is a fundamental technique of unsupervised learning. The purpose of clustering is to divide a set of the dataset into clusters (groups) in such a way that the data of each cluster are similar. The clustering has great significance and different applications (e.g. image segmentation Chuang et al. (2006), object recognition Lowe (2001), information retrieval Jardine and van Rijsbergen (1971)), in various domains including as biology and pattern recognition. In recent years, the searchers had shown great attention to this field (see, e.g., Al-Sultan (1995); Belacel et al. (2004); Belghiti et al. (2004); Brusco (2006); Da Silva et al. (2020); Handl and Knowles (2007); Hartigan and Wong (1979); Hatamlou (2013); Jain (2010); Jaiprakash and Nanda (2019); Jensen (1969); Kanungo et al. (2002); Le Hoai et al. (2013a, 2006, 2012, 2013b); Le Thi (2019b); Le Thi et al. (2006, 2007a, 2014a, 2008, 2007b,c, 2014c); Le Thi and Pham Dinh (2009); Mahdavi and Abolhassani (2009); Maulik and Bandyopadhyay (2000); Merendino and Celebi (2013); Merle et al. (1999); Paterlini and Krink (2004); Selim and Alsultan (1991); Shelokar et al. (2004); Sherali and Desai (2005); Ta et al. (2012); Thuy and Le Thi (2015); Thuy et al. (2013b,a); Vo et al. (2016); Xia and Peng (2005)).

In the general term, an instance of clustering problem consists of $X = \{x_1, x_2, ..., x_m\}$ a set of m points in $\mathbb{R}^n$ to be subdivided into a k subsets of similar points called clusters. The objective is to assign the elements to the clusters so that a criterion function is minimized (or maximized). In the literature, many criteria and various concepts of measuring the similarities between the dataset elements have been studied, we can refer the reader to a general survey in Jain et al. (1999). The squared Euclidean distance from each element to the centroid of its cluster, or the minimum sum-of-squares (MSSC) for short, is the most popular and used one.

The MSSC can be mathematically formulated as a continuous bilevel programming model as follows

$$\min\{\Sigma_{i=1}^{m} \min_{l=1,...,k} \|x^l - a^i\|^2 : x^l \in \mathbb{R}^n, l = 1, ..., k\}, \tag{4.1}$$

where $\|.\|^2$ denotes the Euclidean distance.

On another hand, the MSSC can also be modeled as a mixed integer program. To describe this last, we start with the decision variables

$$y_{il} = \begin{cases} 1 & if\ the\ point\ i\ is\ assigned\ to\ the\ centroid\ l, \\ 0 & otherwise. \end{cases}$$

The discrete optimization problem of MSSC can be expressed as follows Al-Sultan (1995); Le Thi et al. (2014c)

$$\begin{cases} \min \Sigma_{i=1}^{m} \Sigma_{l=1}^{k} y_{il} \|x^l - a^i\|^2, \\ \Sigma_{l=1}^{k} y_{il} = 1, & \forall i = 1, ..., m, \\ x^l \in \mathbb{R}^n, \\ y_{il} \in \{0, 1\}, & \forall i = 1, ..., m, \forall l = 1, ..., k. \end{cases} \tag{4.2}$$

In this chapter, we consider the two models. Both problems (4.1) and (4.2) are non-convex which makes very hard to find an efficient solution for them.

Related works

Many searchers have studied the clustering problem and have proposed numerous algorithms. k-means is one of the most popular ones (see, Hartigan and Wong (1979); Jain (2010); Kanungo et al. (2002); Mahdavi and Abolhassani (2009)). We can find some other optimization approaches for the clustering problem published in several articles (e.g. Brusco (2006); Jensen (1969); Merle et al. (1999); Sherali and Desai (2005); Xia and Peng (2005)).
Another direction of research is focusing on the concept of the nature-inspired metaheuristic algorithms among them: genetic algorithm (GA) in Maulik and Bandyopadhyay (2000), Paterlini and Krink have studied a performance comparison between GA, particle swarm optimization and differential evolution in Paterlini and Krink (2004), an algorithm based on tabu search technique has been developed by in Al-Sultan (1995), simulated annealing Merendino and Celebi (2013); Selim and Alsultan (1991), elephant herding algorithm Jaiprakash and Nanda (2019), an ant colony approach for clustering Shelokar et al. (2004), variable neighborhood search Belacel et al. (2004), an evolutionary approach Handl and Knowles (2007), and a new heuristic optimization Hatamlou (2013). For an overall overview of the major metaheuristics applied to cluster data, we refer the interested reader to Das et al. (2009); José-García and Gómez-Flores (2016); Ramadas and Abraham (2019). DC programming and DCA have also been successfully applied to clustering problem. Many works basing on this theory have been investigated in this field among them Belghiti et al. (2004); Da Silva et al. (2020); Le Hoai et al. (2013a, 2006, 2012, 2013b); Le Thi (2019b); Le Thi et al. (2006, 2007a, 2014a, 2008, 2007b,c, 2014c); Le Thi and Pham Dinh (2009); Ta et al. (2012); Thuy and Le Thi (2015); Thuy et al. (2013b,a); Vo et al. (2016).

Contribution and motivation

We propose two cooperations between DCA and metaheuristic algorithms for the clustering problem. We consider for the models in the problem (4.1) and the problem (4.2). The concept of a cooperative scheme has been studied in combinatorial optimization by several authors. In Yagouni and Le Thi (2014), a so-called Metastorming approach have been used. Metastorming is a cooperation between metaheuristic algorithms. The same idea has been studied and extended to cooperate many DC

Algorithms Le Thi (2019b). Our approaches are based on DCA, tabu search, variable neighborhood search, and migrating birds optimization. We use the discrete model in metaheuristics, whereas the continuous bilevel model in DCA because this last is more convenient for DCA than the discrete model with a penalty function.The proposed approaches in this chapter, start by running the component algorithms in parallel with each one in a different process. Then, the processes communicate with each other by distributing information via the Message Passing Interface protocol. Therefore, they call the MPI_Put function which allows a process to send a message, access the memory of the receiver process, and save the message without blocking the work of the receiver. The choice of the component algorithms is motivated by the successful application of DCA to many smooth/nonsmooth large scale nonconvex programs. Concerning the metaheuristics, they are proven to be good algorithms due to their results in combinatorial optimization and clustering.

The outline of this chapter is as follows; Section 2 gives an application of the component algorithms to the MSSC. We discuss the concept of the cooperation as well as the schemes of the proposed approaches, in section 3. Then, the computational results are reported in section 4. Finally, section 5 is a conclusion to this chapter.

4.2 Application of the component algorithms to the MSSC

The application of TS, VNS, and MBO to the mixed integer model of the MSSC is equivalent to what we did in chapter 2 and chapter 3. As for DCA, the DC programming and DCA have been applied to the continuous bilevel model of the MSSC in Le Thi et al. (2007a). The following DC decomposition has been proposed to (4.1):

$$G(x) = \Sigma_{i=1}^{m}\Sigma_{l=1}^{k} G_{il}(x), G_{il}(x) = \frac{1}{2}\|x_l - a^i\|^2 \quad for\ i = 1, ..., m, l = 1, ..., k. \tag{4.3}$$

$$H(x) = \Sigma_{i=1}^{m} H_i(x), H_i(x) = \max_{j=1,...,k} \Sigma_{l=1, l\neq j}^{k} \frac{1}{2}\|x_l - a^i\|^2 \quad for\ i = 1, ..., m. \tag{4.4}$$

The first sequence y^k can be computed by

$$y^k = mx^k - B - \Sigma_{i=1}^{m} e_{j(i)}^{[k]}(x_{j(i)}^k - a^i). \tag{4.5}$$

x^{k+1}, the solution to the DC sub-problem is equivalent to

$$x^{k+1} = \frac{1}{m}(B + y^k). \tag{4.6}$$

Where $B \in \mathbb{R}^{kn}$, $B_l = \Sigma_{i=1}^{m} a^i$ for $l = 1, ..., k$.

$e_j^{[k]}$ being the canonical basis of $\mathbb{R}^k$.

Algorithm 4.1 DCA for solving MSSC (4.1)

Initialization: Choose $x^0 \in \mathbb{R}^{kn}$, $k = 0$.
repeat
1. Compute $y^k = \nabla H_\rho(x^k)$ by

$$y^k = mx^k - B - \Sigma_{i=1}^{m} e_{j(i)}^{[k]}(x_{j(i)}^k - a^i) \tag{4.7}$$

for some $j(i) \in K_i(x)$.
2. Compute

$$x^{k+1} = \frac{1}{m}(B + y^k). \tag{4.8}$$

3. Set $k = k + 1$.
until Stopping criterion is satisfied.

$K_i(x) = \{j = 1, ..., k : H_{ij}(x) = H_i(x)\}$.

$H_i = \max_{j=1,...,k} H_{ij}$ and $H_{ij} = \Sigma_{l=1,l\neq j}^{k} \frac{1}{2}\|x_l - a^i\|^2$ for $i = 1, ..., m$.

4.3 Cooperative schemes

We investigate two cooperative approaches. The principle is the same for both of them, which means that they use two mathematical models of MSSC, combine DCA with metaheuristics, and operate the same steps. Each approach uses the two equivalent models the mixed integer program (4.2) and the continuous bilevel one (4.1) of the MSSC. The problem (4.2) is adopted by the metaheuristics while the problem (4.1) is handled by the DCA. The difference between the two approaches remains in the used participating (component) algorithms. More precisely, Cop1 is based on DCA, TS, and VNS whereas Cop2 uses, additionaly, MBO with the previous algorithms. The exchange of data is effected via one-sided point-to-point communication. We first give an overall description of the cooperations' principle. Then, we present the schemes of Cop1 in Fig. 4.1. and Cop2 in Fig. 4.2.

The initialization

In this phase, each process generates an initial solution for its algorithms. In order to diversify the research, it is favored to use different methods. After that, all the processes exchange their solutions (the coordinate of the centroids) as well as the objective function, to be saved in the memory as the information about the result of the others. For this purpose, we call the function MPI_Put. The algorithms used to generate the initial solutions for Cop1 and Cop2 can be summarized in table 4.1.

The idea of a random function consists in randomly choosing k points among the given ones to be the centroids. To generate a random population, we run many random functions. To diversify the members of a population, we don't allow redundant

Process	The model of MSSC	Initial algorithm	Initial solution	Algorithms
1	Problem (4.1)	DCA-Kmean	S_{01}	DCA
2	Problem (4.2)	Random function	S_{02}	TS
3	Problem (4.2)	Random function	S_{03}	VNS
4	Problem (4.2)	Random population	Pop_{04}	MBO

Table 4.1 – The tools of the initialization phase.

solutions. As for DCA-Kmean, the procedure is about running some iterations of Kmean and DCA in an alternative way. DCA-Kmean has been investigated in Le Thi et al. (2007a).

Phase of running and communication
Each process runs one iteration of its algorithm. Then, it distributes its result which is composed of the coordinate of the centroids and the objective value, to the others using the function MPI-Put of MPI. The role of this function is to allow a process X to access the memory of a process Y to save information there without blocking the receiver process (Y). Afterward, each process compares the solutions saved in its memory and chooses the best one which means that the solution having the smallest objective value, for starting the next iteration. Coming up next, the step of testing whether a stopping condition is satisfied or not. Therefore, if all the processes satisfy a stopping condition, the cooperation ends. Otherwise, they repeat this "running and communication" phase.

The last phase
When all the processes finish running, one of them saves the final result, and the cooperative approach terminates.

Properties of Cop1 and Cop2

In what follows, we briefly outline the specific properties of each cooperative approach. More precisely, we give the component algorithms and the motivation of our choice.

1. **Cop1**
   Cop1 is based on DCA and two trajectory metaheuristics (TS and VNS). We opted for these three algorithms because each one of them uses a single solution which makes the exchange of information easier and faster than using population-based metaheuristics. The following scheme describes the work of each process over time.
2. **Cop2**
   This approach is an extension of Cop1. In Cop2 we used MBO in addition to DCA, TS, and VNS. In Fig. 4.2., after the evaluation step, in the case where the solution of MBO is not the best, the process running MBO will proceed as follows:

(a) Deletes the bad solution in the lines of MBO.

(b) Put the leader in the place of the deleted bird.

(c) The best solution to the iteration will replace the leader.

4.4 Numerical results

This section is devoted to numerical simulations. We used C++ to code our approach and we compiled it within Microsoft Visual Studio 2017. All experiments were carried out on a Dell desktop computer with Intel Core(TM) i5-6600 CPU 3.30GHz and 8GB RAM under Windows 10.
To assess the efficiency of Cop1 and Cop2 on the MSSC, we opted for several datasets of different sizes (see, table 4.2. for all the details). The used datasets are taken from different websites which are:

— https://archive.ics.uci.edu/ml/datasets.php
— https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
— http://featureselection.asu.edu/datasets.php
— http://cs.joensuu.fi/sipu/datasets/

This section contains two parts. The first deals with a comparison between Cop1, Cop2, and component algorithms. The second shows how cooperation is expressed.

4.4.1 Comparison

We provide a comparison between Cop1, Cop2, and component algorithms. The comparison is realized on three criteria: the objective value, the rand index (RI), and the computation time (measured in seconds). The rand index is a value that measures the similarities between two different clusterings and it has been named by William. Before giving the RI formulation, we introduce some notations. Let $P_1 = \{C_{11}, ..., C_{1k}\}$ and $P_2 = \{C_{21}, ..., C_{2k}\}$ be two different partitions of X into k classes.

- $a = \{(x_i, x_j) \in X | (x_i, x_j) \in C_{1l}, (x_i, x_j) \in C_{2t}\}$
- $b = \{(x_i, x_j) \in X | x_i \in C_{1i}, x_j \in C_{1j}, x_i \in C_{2i}, x_j \in C_{2j}\}$
- $c = \{(x_i, x_j) \in X | (x_i, x_j) \in C_{1l}, x_i \in C_{2i}, x_j \in C_{2j}\}$
- $b = \{(x_i, x_j) \in X | x_i \in C_{1i}, x_j \in C_{1j}, (x_i, x_j) \in C_{2t}\}$

The RI can be computed as follows:

$$\frac{|a| + |b|}{|a| + |b| + |c| + |d|} \tag{4.9}$$

From the numerical results reported in tables 4.4., 4.5., 4.6., 4.7., and 4.8., we observe:

The objective value

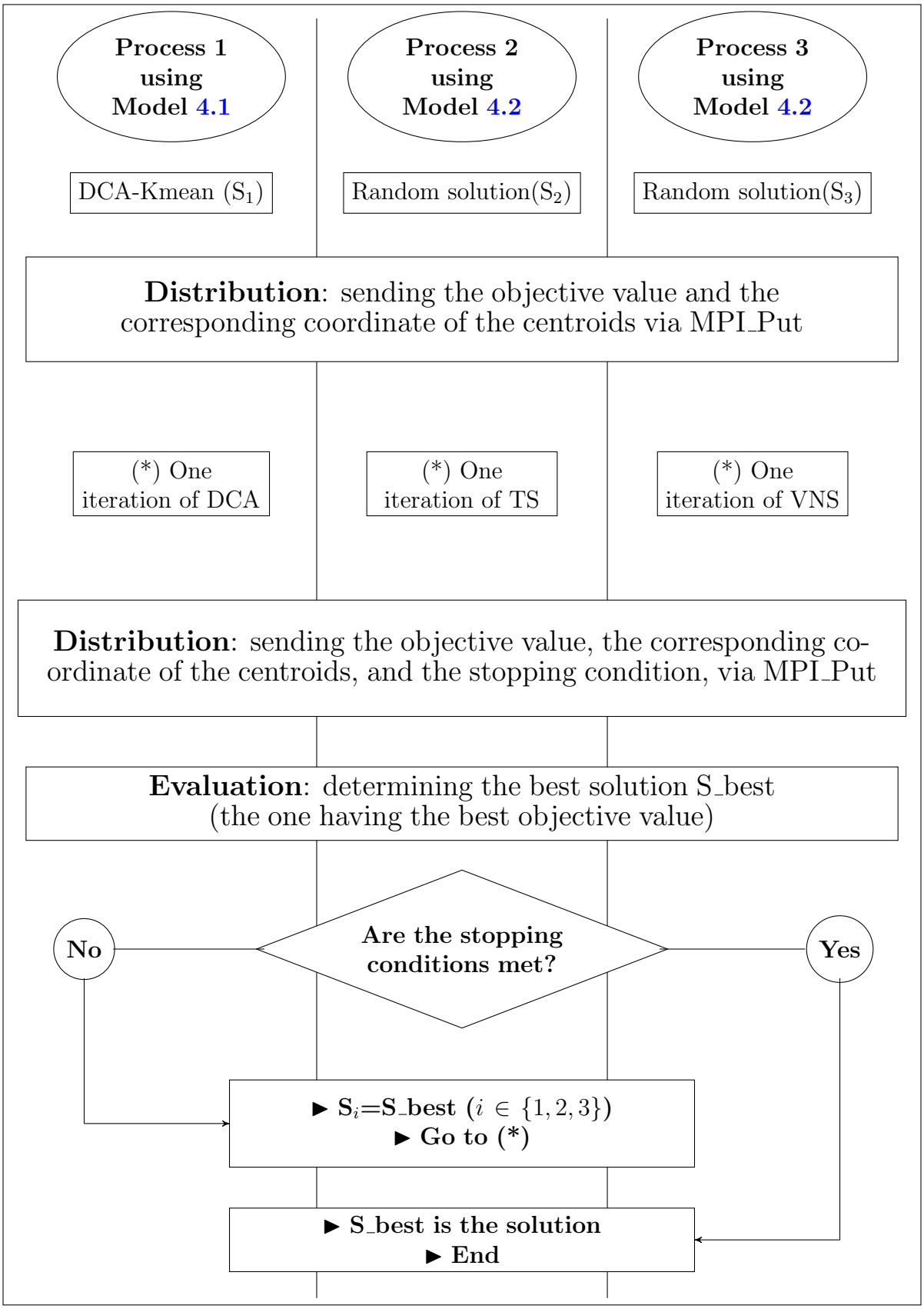


Figure 4.1 – Cop1: cooperative scheme based on TS, VNS, and DCA.

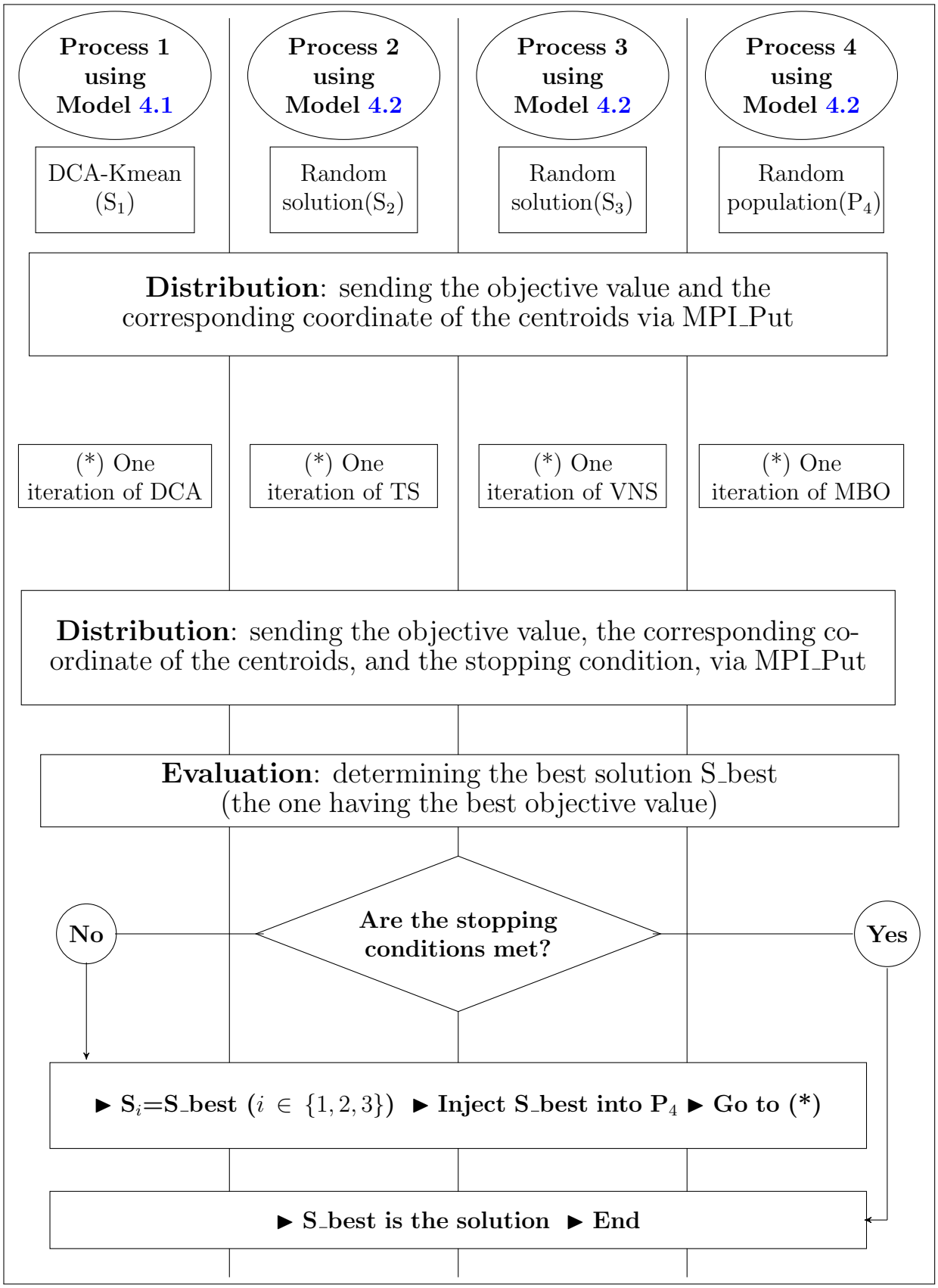


Figure 4.2 – Cop2: cooperative scheme based on TS, VNS, MBO, and DCA.

Number	Dataset	Points	Number of classes	Dimension
1	Zoo	101	7	16
2	Iris	150	3	4
3	Wine	178	3	13
4	Glass	214	7	9
5	Ecoli	336	8	7
6	WDBC	569	2	30
7	Balance_scale	625	3	4
8	DIM032	1024	16	32
9	DIM064	1024	16	64
10	DIM128	1024	16	128
11	DIM256	1024	16	256
12	DIM512	1024	16	512
13	DIM1024	1024	16	1024
14	Coil20	1440	20	1024
15	CMC	1473	3	9
16	Yeast	1484	10	8
17	Madelon	2600	2	500
18	Abalone	4177	28	8
19	waveform_noise_v1	5000	3	40
20	Isolet	7797	26	617
21	USPS	9298	10	256
22	RCV1	10000	4	2000
23	Epileptic_Seizure_Recognition	11500	5	178
24	Protein	17766	3	357
25	HTRU2	17898	2	8
26	Letter_recognition	20000	26	16
27	shuttle.scale	43500	7	9

Table 4.2 – Datasets used in our experiments.

Starting from the best one, the order is as follows: Cop2, Cop1, MBO, DCA, TS, then finally VNS. It can be seen that Cop2 is better than the other algorithms for twelve data instances, Cop1 and MBO for six datasets, DCA for three other ones, whereas TS is the best only one time and VNS is the worst always. Generally, there is a very small difference between Cop1 and Cop2. This means that adding MBO as a component algorithm makes a bit of improvement.

The rand index

The result of sorting the RI from the best to the worst shows that MBO comes at the first position, Cop2 the second, after that VNS, Cop1 and TS, then finally DCA. In terms of the average of RI, we note in table 4.3. that all algorithms get good results. More precisely, TS and MBO are the best. Then Cop1 and Cop2. After that, VNS. As for DCA, it comes at the end.

Algorithm	The average of RI
Cop1	0,76
DCA	0,73
TS	0,77
VNS	0,75
MBO	0,77
Cop2	0,76

Table 4.3 – Average of RI.

The CPU time

From the results in tables 4.4., 4.5., 4.6., 4.7., and 4.8., we observe that in term of CPU time, DCA is better than the other algorithms on all the datasets. The average times of Cop1, DCA, TS, VNS, MBO, and Cop2 are, respectively, 498.57, 44.38, 644.94, 381.01, 5443.99, and 1123.04 seconds.

4.4.2 Study of the cooperations

In tables 4.9., 4.10., and 4.11., we show for two datasets, how the cooperation has been realized between the component algorithms of Cop1 and Cop2. More specifically, we give the objective value of the initial solution (OVIS) and the objective value (OV) found by the component algorithms. It can be seen in the tables that, all the component algorithms participate to improve the quality which is the goal of our study.

1. **Cop1**:

Dataset	Algorithm	Objective value	Rand Index	CPU
Zoo	Cooperation 1	230.598	0.83	0.25
	DCA	214.099	0.83	0.02
	TS	311	0.89	0.58
	VNS	263	0.83	0.84
	MBO	187	**0.90**	31.07
	Cooperation 2	187.796	0.85	3.20
Iris	Cooperation 1	94.2633	**0.91**	0.21
	DCA	118.686	0.88	0.02
	TS	207.89	0.76	0.49
	VNS	207.28	0.84	0.41
	MBO	98.06	0.87	55.92
	Cooperation 2	92.0231	0.60	0.74
Wine	Cooperation 1	2.7145E+06	0.71	0.31
	DCA	2.58728E+06	0.72	0.05
	TS	2.68735E+06	0.72	0.60
	VNS	4.91481E+06	**0.75**	0.84
	MBO	2.76145E+06	0.73	41.47
	Cooperation 2	2.43413E+06	0.71	0.70
Glass	Cooperation 1	478.499	0.69	0.85
	DCA	527.413	0.69	0.12
	TS	494.941	0.69	0.94
	VNS	1028.94	0.67	1.10
	MBO	487.686	0.63	78.27
	Cooperation 2	450.709	**0.73**	4.84

Table 4.4 – Comparative results (1).

Dataset	Algorithm	Objective value	Rand Index	CPU
Ecoli	Cooperation 1	17.883	**0.83**	1.38
	DCA	20.1263	**0.83**	0.08
	TS	20.4137	0.79	1.18
	VNS	41.6087	0.72	0.86
	MBO	19.2892	0.79	44.58
	Cooperation 2	17.1079	0.80	8.16
WDBC	Cooperation 1	1.14335E+08	0.79	1.01
	DCA	9.02889E+07	0.69	0.13
	TS	1.2941E+08	**0.83**	3.13
	VNS	1.19511E+08	**0.83**	3.36
	MBO	8.16337E+07	0.77	200.70
	Cooperation 2	8.02063E+07	0.76	5.40
balance_scale	Cooperation 1	3697.13	**0.69**	0.81
	DCA	4021.33	0.54	0.10
	TS	4300	0.64	1.21
	VNS	3832	0.58	1.63
	MBO	3775	0.56	134.36
	Cooperation 2	3697.13	0.54	0.81
DIM032	Cooperation 1	29675300	0.97	7.29
	DCA	3.10935E+07	0.97	0.71
	TS	3.1407E+07	0.98	7.26
	VNS	1.24792E+08	0.93	14.72
	MBO	1.38894E+07	**0.99**	490.77
	Cooperation 2	2.08258E+07	0.97	71.93
DIM064	Cooperation 1	3.26576E+07	**0.98**	11.49
	DCA	8.40377E+07	0.91	2.59
	TS	6.34153E+07	0.97	25.12
	VNS	2.90236E+08	0.96	29.06
	MBO	5.12798E+07	0.97	566.14
	Cooperation 2	1.05872E+08	**0.98**	230.82
DIM128	Cooperation 1	1.33373E+08	0.96	35.74
	DCA	1.42596E+08	0.96	1.81
	TS	1.96677E+08	0.98	33.26
	VNS	6.4109E+08	0.95	76.26
	MBO	1.07958E+08	**0.99**	1504.77
	Cooperation 2	2.47359E+08	0.98	330.85

Table 4.5 – Comparative results (2).

Dataset	Algorithm	Objective value	Rand Index	CPU
DIM256	Cooperation 1	2.85507E+08	0.92	52.46
	DCA	3.18199E+08	**0.98**	3.52
	TS	4.66294E+08	**0.98**	359.77
	VNS	1.39332E+09	0.92	79.39
	MBO	3.3148E+08	0.92	2429.69
	Cooperation 2	3.30267E+08	**0.98**	182.37
DIM512	Cooperation 1	6.24188E+08	0.92	136.60
	DCA	8.9027E+08	0.84	10.16
	TS	7.11343E+08	**0.98**	280.08
	VNS	2.80916E+09	0.95	381.03
	MBO	3.51075E+08	0.99	5225.52
	Cooperation 2	6.08348E+08	0.92	851.20
DIM1024	Cooperation 1	1.29133E+09	0.92	165.58
	DCA	1.78735E+09	0.91	55.19
	TS	2.20708E+09	0.94	397.49
	VNS	5.77921E+09	0.93	815.00
	MBO	1.10799E+09	**0.99**	17966.60
	Cooperation 2	1.03324E+09	0.98	1671.98
Coil20	Cooperation 1	55186	0.93	186.30
	DCA	58437.1	0.89	47.31
	TS	64979.5	0.93	615.62
	VNS	161328	0.73	427.03
	MBO	57181.3	0.92	2513.45
	Cooperation 2	52925.1	**0.94**	2670.14
CMC	Cooperation1	37129.1	0.55	6.50
	DCA	35665	0.55	0.29
	TS	41499	**0.56**	5.34
	VNS	53275	0.51	6.37
	MBO	30039	**0.56**	122.46
	Cooperation 2	35784.6	0.5(	14.59
yeast	Cooperation 1	59.5699	0.74	15.74
	DCA	66.0902	0.74	1.254
	TS	64.5993	0.68	8.997
	VNS	65.0684	0.73	4.948
	MBO	56.6818	**0.75**	475.83
	Cooperation 2	59.5699	0.74	16.1565

Table 4.6 – Comparative results (3).

Dataset	Algorithm	Objective value	Rand Index	CPU
Madelon	Cooperation 1	1.18205E+09	0.50	108.56
	DCA	1.18228E+09	0.50	4.53
	TS	1.87982E+09	**0.52**	1718.19
	VNS	2.2074E+09	0.51	100.97
	MBO	1.89563E+09	0.50	3404.81
	Cooperation 2	1.18222E+09	0.50	117.17
abalone	Cooperation1	118.744	0.86	53.75
	DCA	130.214	0.86	17.82
	TS	154.406	0.86	30.94
	VNS	151.125	0.86	36.37
	MBO	98.7095	0.86	1667.22
	Cooperation 2	96.1557	0.86	195.80
waveform_noise_v1	Cooperation 1	8216.49	**1.00**	44.11
	DCA	25248.6	0.36	0.9512
	TS	8214.5	**1.00**	52.1712
	VNS	25759.2	0.95	55.47
	MBO	8513.04	**1.00**	4591.86
	Cooperation 2	8668.29	**1.00**	218.01
isolet	Cooperation 1	797554	0.86	5212.46
	DCA	783240	0.91	611.58
	TS	825501	0.93	1799.78
	VNS	1036380	0.92	1924.39
	MBO	793292	**0.94**	13159.50
	Cooperation 2	761537	0.93	10301.40
USPS	Cooperation 1	527724	0.59	1504.64
	DCA	488472	0.84	103.411
	TS	584929	0.85	1922.31
	VNS	555785	**0.86**	587.34
	MBO	541237	**0.86**	3338.22
	Cooperation 2	496036	0.84	3013.77
rcv1	Cooperation 1	5509.08	0.25	257.01
	DCA	5508.06	0.25	98.52
	TS	8783.63	0.28	2712.41
	VNS	9032.22	**0.31**	4138.91
	MBO	6364.15	0.27	12600.60
	Cooperation 2	5509.59	0.25	1612.51

Table 4.7 – Comparative results (4).

Dataset	Algorithm	Objective value	Rand Index	CPU
Epileptic_	Cooperation 1	5.36327e+010	**0.60**	1884.57
Seizure_	DCA	5.34616e+010	**0.60**	121.58
Recognition	TS	5.49156e+010	**0.60**	372.92
	VNS	5.59836e+010	0.51	286.96
	MBO	5.37663e+010	0.49	51479.70
	Cooperation 2	5.35539e+010	0.52	1777.94
Protein	Cooperation 1	140595	0.36	49.26
	DCA	140597	0.36	44.42
	TS	249236	0.44	6670.16
	VNS	365508	**0.54**	846.05
	MBO	180942	0.37	15704.50
	Cooperation 2	140590	0.36	727.21
HTRU2	Cooperation 1	1.25002E+08	0.66	74.07
	DCA	1.29263E+08	0.64	7.08
	TS	1.52912E+08	0.62	53.55
	VNS	1.94265E+08	0.61	49.64
	MBO	1.31281E+08	**0.72**	1462.63
	Cooperation 2	1.25326E+08	0.63	313.46
Letter_	Cooperation 1	879210	**0.92**	2478.45
Recognition	DCA	968083	**0.92**	37.79
	TS	998651	**0.92**	175.85
	VNS	1068010	0.91	218.199
	MBO	927662	**0.92**	6631.31
	Cooperation 2	900072	0.91	3559.83
shuttle.scale	Cooperation 1	1316.77	0.541	1171.93
	DCA	1393.16	0.53	27.20
	TS	1487.43	0.49	164.04
	VNS	2177.54	0.53	200.21
	MBO	1856.22	**0.56**	1065.86
	Cooperation 2	1231.58	**0.56**	2600.10

Table 4.8 – Comparative results (5).

At each iteration of the cooperation, the processes have the same solutions in their memories because the consumed time at each iteration of everyone is approximately the same as it is expressed in tables 4.9. and 4.10.

Iteration	Details	DCA	TS	VNS
0	OVIS	8973.77	25759.20	25759.2
1	OV	8973.77	25759.20	25759.2
2	OV	8973.77	8972.31	8973.77
3	OV	8913.24	8859.84	8972.31
4	OV	8779.58	8835.49	8859.84
5	OV	8779.58	8774.82	8776.37
6	OV	8771.79	8768.29	8771.05
7	OV	8750.10	8768.11	8768.29
8	OV	8750.10	8749.87	8750.10
9	OV	8747.16	8688.64	8748.01
10	OV	8660.03	8655.13	8678.99
11	OV	8602.65	8625.08	8649.62
12	OV	8602.65	8567.00	8602.42
13	OV	8547.92	8564.99	8565.72
14	OV	8547.92	8543.79	8547.92
15	OV	8526.67	8538.76	8542.69
16	OV	8526.67	8513.55	8501.55
17	OV	8500.81	8501.20	8501.55

Table 4.9 – Analysis of the behavior of component algorithms (approach: Cop1, Dataset: waveform_noise_v1) (part 1).

Iteration	Details	DCA	TS	VNS
18	OV	8500.81	8478.02	8500.81
19	OV	8475.59	8441.19	8478.02
20	OV	8375.95	8439.65	8441.19
21	OV	8375.95	8374.17	8350.08
22	OV	8349.28	8333.47	8299.98
23	OV	8285.34	8272.04	8266.01
24	OV	8244.55	8260.98	8262.25
25	OV	8244.55	8237.33	8244.55
26	OV	8216.49	8231.58	8234.62
27	OV	8216.49	8216.49	8216.49

Table 4.10 – Analysis of the behavior of component algorithms (approach: Cop1, Dataset: waveform_noise_v1) (part 2).

2. **Cop2**:
   For the second approach, at each iteration of the cooperation, the processes don't have the same solutions in their memories because the consumed time at each iteration of everyone is not the same (MBO is quite slow) as it can be seen in the table 4.11.

Iteration		DCA	TS	VNS	MBO
0	OVIS	100.304	332.72	225.59	96.19
1	OV	100.159	332.72	225.59	96.19
2	OV	92.0554	96.19	96.19	96.19
3	OV	92.0231	96.19	96.19	96.19
4	OV	92.0231	92.0554	92.0554	92.0231
5	OV	92.0231	92.0231	92.0231	92.0231

Table 4.11 – Analysis of the behavior of component algorithms (approach: Cop2, dataset: Iris).

4.5 Conclusion

We have developed two new and efficient cooperative approaches based on DCA and the following metaheuristics, VNS, TS, and MBO for solving the MSSC problem using two equivalent models of this last. A bilevel programming model adopted by DCA and a mixed integer program for the metaheuristics. The approaches consist in starting the component algorithms in parallel and working independently. At each iteration, they exchange their solutions via the function MPI_Put (one-sided point-to-point communication) to start the next iteration by the best one. The best solution is the one having the smallest objective value. Computational experiments on various benchmark datasets report that Cop1 and Cop2 are quite efficient, they provide a good objective value. As for the running time, DCA is the fastest. The comparative results turn out that the best trade-off between quality and rapidity is realized by DCA. In this study, the cooperation principle has been proved to be a good technique to improve the component algorithms' quality.

Conclusion and perspectives

The research reported in this thesis focuses on the development of new cooperative approaches for solving nonconvex optimization problems using the paradigm of parallel and distributed programming. The backbones of our approaches are DC (Difference of convex functions) programming and DCA (DC Algorithms) which are powerful tools in the nonconvex optimization area, as well as metaheuristic methods. In this dissertation, we studied three classes of problems. We proposed different cooperative approaches inspired by the brainstorming and the metastorming.

First, we have solved the mixed binary linear programs via a new cooperative approach based on DCA and VNS using point-to-point two-sided communication. Through extensive numerical analysis on a set of benchmarks of the capacitated facility location problem, the efficiency of our approach has been proven in terms of the quality and the speed by comparing it with CPLEX solver as well as the component algorithms (DCA and VNS). In the cooperative approach, both VNS and DCA participated to improve the solution which means that the solution of VNS has been used as an initial solution by DCA which has improved it, and the same thing for VNS. The disadvantage here is that VNS is much slower than DCA. Consequently, the cooperative approach consuming much more time than DCA, as a waiting time is required for DCA in the cooperative scheme.

Second, for handling the binary quadratic programming in the chapter three, we have proposed another new cooperative approach which is achieved by using migrating birds optimization, the genetic algorithm, and DCA-Like (a recent version of DCA) as component algorithms. The adopted manner of communication is the collective one-sided model. The computational experiences on several benchmark datasets of the quadratic assignment problem indicated the superiority of the investigated approach in comparison with the component algorithms. Moreover, the study of the behaviors of the component algorithms showed that cooperation has perfectly been realized. As for the speed, MBO is the slowest and DCA-like is the fastest. What is missing in this cooperation is that the running time for solving big size problems is too large.

As for the last part of this thesis, our study is focused on Minimum-Sum-of-Squares Clustering problem. We developed two cooperative approaches. The first is composed of DCA, VNS, and TS. Whereas the second is based on DCA, VNS, TS, and MBO. Here, the parameters used for VNS and MBO differ from those used for VNS in the chapter 2 and MBO in the chapter 3. We have considered two different mathematical

models of the MSSC in each approach. To be able to share solutions at each iteration without waiting time, we have relied on the point-to-point one-sided communication to exchange information between the component algorithms. In the first cooperation, each one of the component algorithms played an important role and participated to improve the quality of the solution. When we used MBO in the second cooperation, the quality of the solution is better, comparing with the first approach. Here, the solution of MBO has been used as an initial solution by the other component algorithms and has been improved by these last. As for the running time, the first approach was faster than the second. In the first approach, at each iteration, the component algorithms consumed approximatively the same computation time. In the second one, MBO was the slowest. The running times of VNS, TS, and DCA are comparable. The numerical results show that the cooperative approaches are quite efficient and they provide good solutions. In these approaches, the weakness remains in solving big size clustering datasets in a short time

In a general way, it turns out that when we use a cooperative approach, combining population-based algorithms improves more the quality of the solution and using algorithms based on a single solution can reduce the computation time. The communication functions play also a very important role in the speed of the approach. The two-sided communication implies a synchronization whereas one-sided communication reduces it. The synchronization means that the sender process needs to wait for the receiver to start receiving. The advantages of a cooperative approach remain in exploiting characteristics of each component algorithm, in our case, the flexibility and the power of DCA with the intensification and diversification principles of metaheuristics to improve the solution and use this last by all the component algorithms to improve it more. The main limit of the cooperative approaches is that they are expensive in terms of time especially when the communication functions require a waiting time. Moreover, using different component algorithms requires many parameters which are very difficult to choose.

In summary, we solved three classes of problems using various cooperative approaches. These last differ from each other because we want to make this thesis stronger, richer and composed of divers algorithms. The difference between the developed cooperations lies in many important points that we give in what follows:

- **The DCA schemes**: DCA with an update of the penalty parameter, DCA-like with an update of the penalty parameter, and DCA for a continuous problem.
- **The metaheuristics**: the suitable choice of parameters.
- **The component algorithms**: we used different component algorithms, the number is not the same at each cooperation, the nature of the algorithms differs (deterministic, trajectory metaheuristics, population-based metaheuristics).
- **The communication manner**: point-to-point, collective, one-sided, two-sided.
- **The adopted mathematical models**: in the chapter two and the chapter three, we used one model of one problem, as for the chapter four, we integrated two mathematical models for the same problem in every single approach.

The work conducted in this thesis drew our attention to a number of extremely thought-provoking paths.

Chapter one:
- introducing other component algorithms in the cooperative approach;
- using other penalty function in DCA;
- using non blocking communication functions.

Chapter three:
- solving BQP by the approaches used in the chapter four;
- using a cooperative approach based on single solution algrithms;
- change the communication functions and/or the penalty function;
- cooperating many versions of DCA (DCA, DCA-like accelerated DCA, etc) with other deterministic algorithms and/or metaheuristics.

Chapter four:
- use other communication function to optimize the running time;
- introduce other deterministic algorithms in the cooperative approach.

In general:
- apply the approaches in this thesis for solving more complicated classes of problems, for example, the Quadratically Constrained Quadratic Programming;
- exploiting tools of parallel and distributed programming to optimize the waiting time of the processes in a cooperative approach;
- developing new cooperations based on algorithms of various kinds (alternative, deterministic, approximative, exact, etc).

appendix

Sara SAMIR

Née le 21 Juillet, 1992 (Algérie)

Tél: +33665629635

E-mail: sara.samir@univ-lorraine.fr

Adresse personnelle: 24 Boulevard de Guyenne, 57070, Metz, France

Adresse professionnelle: Bureau UFR-MIM-AN1-33, Département Informatique & Application, LGIPM, Université de Lorraine, 3 rue Augustin Fresnel, BP 45112, 57073 Metz, France

Situation Actuelle

Depuis Octobre 2016	Doctorante au Département Informatique & Application, LGIPM, Université de Lorraine. Encadrée par Prof. Hoai An Le Thi Sujet de thèse : **Approches coopératives pour certaines classes de problèmes d'optimisation non convexe : Algorithmes parallèles / distribués et applications**

Experience Professionnelle

01/2016–06/2016	Stagiaire au Département de Système Informatique, SGSIA, Aéroport international d'Alger, Houari Boumédiène, Alger, Algérie. Mémoire: **Gestion de projet de la construction de la nouvelle aérogare ouest**.

Diplôme et Formation

2016 au present	Doctorante au Département Informatique & Application, LGIPM, Université de Lorraine, Metz, France.
2014–2016	Master en Recherche Opérationnelle - Modèles et Méthodes pour l'Ingénierie et la Recherche, Université des sciences et de la technologie Houari Boumédiène (USTHB), Alger, Algérie.
2011–2014	Licence en Recherche Opérationnelle,Université des sciences et de la technologie Houari Boumédiène (USTHB), Alger, Algérie.

Publications

Refereed papers in books / Refereed international conference papers

[1] Samir Sara, Le Thi Hoai An. A Collaborative Approach Based on DCA and VNS for Solving Mixed Binary Linear Programs. In: Nguyen N.T., Gaol F. L., Hong T.-P., Trawiǹski B. (eds) Intelligent Information and Database Systems. ACIIDS 2019. Lecture Notes in Computer Science, Springer, Cham, vol 11431, pp. 510-519, (2019).

[2] Samir Sara, Le Thi Hoai An, Yagouni Mohammed. DCA-Like, GA and MBO: A Novel Hybrid Approach for Binary Quadratic Programs. In: Le Thi H., Le H., Pham Dinh T. (eds) Optimization of Complex Systems: Theory, Models, Algorithms and Applications. WCGO 2019. Advances in Intelligent Systems and Computing,Springer, Cham, vol 991, pp. 299-309 (2020).

Communications in national / International conferences

[1] Samir Sara, Le Thi Hoai An. A Collaborative Approach Based on DCA and VNS for Solving Mixed Binary Linear Programs. Presentation in the 29th European Conference on Operational Research, Valencia, Spain, July 8-11, 2018.

Bibliography

K. S. Al-Sultan. A tabu search approach to the clustering problem. *Pattern Recognition*, 28(9):1443–1451, 1995.

E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience and Sons, 2005.

S. Atakan and S. Sen. A progressive hedging based branch-and-bound algorithm for mixed-integer stochastic programs. *Computational Management Science*, 15(3):501–540, 2018.

E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58(1):295–324, 1993.

N. Belacel, M. Čuperlović Culf, M. Laflamme, and R. Ouellette. Fuzzy J-Means and VNS methods for clustering genes from microarray data. *Bioinformatics*, 20(11): 1690–1701, 2004.

M. T. Belghiti, H. A. Le Thi, and T. Pham Dinh. Clustering via DC programming and DCA. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 499–507. Hermes Sciences, Publishing, 2004.

M. Bercachi. A new hybrid method between VNS and SEA to improve results on the 0-1 multidimensional knapsack problem. 2010.

C. Blum and A. Roli. *Hybrid Metaheuristics: An Introduction*, pages 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

M. J. Brusco. A repetitive branch-and-bound procedure for minimum within-cluster sums of squares partitioning. *Psychometrika*, 71(2):347–363, 2006.

A. V. Cabot and A. P. Hurter. An approach to zero-one integer programming. *Oper. Res.*, 16(6):1206–1211, 1968.

F. Charot. Architectures parallèles spécialisées pour le traitement d'image. Research Report RR-1978, INRIA, 1993. URL https://hal.inria.fr/inria-00074694.

B. Chu. Composite quasi-maximum likelihood estimation of dynamic panels with group-specific heterogeneity and spatially dependent errors. 2017.

K.-S. Chuang, H.-L. Tzeng, S. Chen, J. Wu, and T.-J. Chen. Fuzzy c-means clustering with spatial information for image segmentation. *Computerized Medical Imaging and Graphics*, 30(1):9–15, 2006.

C. Clark. *Brainstorming: The dynamic new way to create successful ideas.* Garden City, 1958.

H. Crowder, E. L. Johnson, and M. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

V.-D. Cung, S. L. Martins, C. C. Ribeiro, and C. Roucairol. *Strategies for the Parallel Implementation of Metaheuristics*, pages 263–308. Springer US, Boston, MA, 2002.

G. Da Silva, M. Le Hoai, H. A. Le Thi, V. Lefieux, and B. Tran. Customer clustering of french transmission system operator (RTE) based on their electricity consumption. In H. A. Le Thi, M. Le Hoai, and T. Pham Dinh, editors, *Optimization of Complex Systems: Theory, Models, Algorithms and Applications*, volume 991, pages 893–905, Cham, 2020. Springer International Publishing.

S. Das, A. Abraham, and A. Konar. *Metaheuristic Pattern Clustering – An Overview*, pages 1–62. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

S. Dejam, M. Sadeghzadeh, and S. J. Mirabedini. Combining cuckoo and tabu algorithms for solving quadratic assignment problems. volume 2, pages 1–8, 2012.

S. Delage-Santacreu. Introduction au calcul parallèle avec la bibliothèque MPI (message passing interface). 11 2008.

M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477, 1999.

E. Duman and I. Elikucuk. Solving credit card fraud detection problem by the new metaheuristics migrating birds optimization. pages 62–71, 2013.

E. Duman, M. Uysal, and A. F. Alkaya. Migrating birds optimization: A new metaheuristic approach and its application to the quadratic assignment problem. In C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. I. Esparcia-Alcázar, J. J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. N. Yannakakis, editors, *Applications of Evolutionary Computation*, pages 254–263, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

E. Duman, M. Uysal, and A. F. Alkaya. Migrating birds optimization: A new metaheuristic approach and its performance on quadratic assignment problem. *Information Sciences*, 217:65–77, 2012.

R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

O. A. Faickney. *Your Creative Power: How to Use Imagination to brighten life, to get ahead, ch. XXXIII, How To Organize a Squad To Create Ideas*, pages 265–274. Charles Scribner's Sons, New York, 1948.

K. Fleszar and K. S. Hindi. Solving the resource-constrained project scheduling problem by a variable neighbourhood search. *European Journal of Operational Research*, 155(2):402–413, 2004. Financial Risk in Open Economies.

M. Flynn. *Flynn's Taxonomy*, pages 689–697. Springer US, Boston, MA, 2011.

G. Franco-Sepúlveda, J. C. D. Rio-Cuervo, and M. A. Pachón-Hernández. State of the art about metaheuristics and artificial neural networks applied to open pit mining. *Resources Policy*, 60:125–133, 2019.

L. M. Gambardella, É. D. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50(2):167–176, 1999.

F. Glover. A note on linear programming and integerfeasibility. *Operations Research*, 16(6):1212, 1968. ISSN 0030364X.

F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

F. Glover and M. Laguna. *Tabu Search*, pages 2093–2229. Springer US, Boston, MA, 1998.

S. Hanafi, J. Lazić, and N. Mladenović. Variable neighbourhood pump heuristic for 0-1 mixed integer programming feasibility. *Electronic Notes in Discrete Mathematics*, 36:759–766, 2010. ISCO 2010-International Symposium on Combinatorial Optimization.

J. Handl and J. Knowles. An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, 11(1):56–76, 2007.

P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

P. Hansen, N. Mladenović, and D. Urošević. Variable neighborhood search and local branching. 33:3034–3045, 10 2006.

J. A. Hartigan and M. A. Wong. *Applied Statistics*, 28(1):100–108, 1979.

A. Hatamlou. Black hole: A new heuristic optimization approach for data clustering. *Information Sciences*, 222:175–184, 2013. Including Special Section on New Trends in Ambient Intelligence and Bio-inspired Systems.

C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming*, 82(3):291–315, 1998.

V. T. Ho, H. A. Le Thi, and B. D. Chien. Online DC optimization for online binary linear classification. In N. T. Nguyen, B. Trawiński, H. Fujita, and T.-P. Hong, editors, *Intelligent Information and Database Systems*, pages 661–670, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 1992.

K. Holmberg, M. Rönnqvist, and D. Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113(3):544–559, 1999.

A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR).

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

K. P. Jaiprakash and S. J. Nanda. Elephant herding algorithm for clustering. In J. Kalita, V. E. Balas, S. Borah, and R. Pradhan, editors, *Recent Developments in Machine Learning and Data Analytics*, volume 740, pages 317–325, Singapore, 2019. Springer Singapore.

N. Jardine and C. van Rijsbergen. The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval*, 7(5):217–240, 1971.

R. E. Jensen. A dynamic programming algorithm for cluster analysis. *Operations Research*, 17(6):1034–1057, 1969.

A. José-García and W. Gómez-Flores. Automatic clustering using nature-inspired metaheuristics: A survey. *Applied Soft Computing*, 41:192–213, 2016.

B. A. Julstrom. Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 607–614, New York, NY, USA, 2005. ACM.

R. K. Ahuja, J. Orlin, S. Pallottino, M. Scaparra, and M. Scutellà. A multi-exchange heuristic for the single-source capacitated facility location problem. 50:749–760, 2003.

T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

K. Kumbhar. Brainstorming technique: Innovative quality management tool for library. 06 2018.

I. LASRI. Bases de la parallélisation par passage de message : MPI (message passing interface). Technical report, Centre de Calcul de l'université de Bourgogne, 2016.

J. Lazić. *New Variants of Variable Neighbourhood Search for 0-1 Mixed Integer Programming and Clustering*. PhD thesis, School of Information Systems, Computing and Mathematics Brunel University, 2010.

J. Lazić, S. Hanafi, N. Mladenović, and D. Urošević. Variable neighbourhood decomposition search for 0-1 mixed integer programs. 37:1055–1067, 06 2010.

M. Le Hoai, H. A. Le Thi, and T. Pham Dinh. Hierarchical clustering based on mathematical optimization. In W.-K. Ng, M. Kitsuregawa, J. Li, and K. Chang, editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–173, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

M. Le Hoai, T. Ta Minh, H. A. Le Thi, and T. Pham Dinh. DC programming and DCA for clustering using weighted dissimilarity measures. In *Proceedings of the 5th NIPS Workshop on Optimization for Machine Learning*, page 4 pages, Lake Tahoe, Nevada, USA, 2012.

M. Le Hoai, H. Le Thi, T. Pham Dinh, and V. N. Huynh. Block clustering based on difference of convex functions (DC) programming and DC algorithms. *Neural Computation*, 25(10):2776–2807, 2013a.

M. Le Hoai, B. T. N. Thi, M. T. Ta, and H. A. Le Thi. Image segmentation via feature weighted fuzzy clustering by a DCA based algorithm. In N. T. Nguyen, T. van Do, and H. A. le Thi, editors, *Advanced Computational Methods for Knowledge Engineering*, volume 479, pages 53–63, Heidelberg, 2013b. Springer International Publishing.

H. A. Le Thi. http://www.lita.univ-lorraine.fr/~lethi/index.php/dca.html, 2005. (homepage).

H. A. Le Thi. DC programming and DCA for supply chain and production management: state-of-the-art models and methods. *International Journal of Production Research*, 0(0):1–37, 2019a.

H. A. Le Thi. Collaborative DCA: an intelligent collective optimization scheme, and its application for clustering. *Journal of Intelligent & Fuzzy Systems*, pages 1–8, 08 2019b.

H. A. Le Thi and V. T. Ho. Online learning based on online DCA and application to online classification. *Neural Computation*, 32(4):759–793, 2020.

H. A. Le Thi and T. Pham Dinh. Solving a class of linearly constrained indefinite quadratic problems by D.C. algorithms. *Journal of Global Optimization*, 11(3):253–285, 1997.

H. A. Le Thi and T. Pham Dinh. A continuous approch for globally solving linearly constrained quadratic. *Optimization*, 50(1-2):93–120, 2001.

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*, 133(1):23–46, 2005.

H. A. Le Thi and T. Pham Dinh. A continuous approach for the concave cost supply problem via DC programming and DCA. *Discrete Applied Mathematics*, 156(3): 325–338, 2008.

H. A. Le Thi and T. Pham Dinh. Minimum sum-of-squares clustering by DC programming and DCA. In D.-S. Huang, K.-H. Jo, H.-H. Lee, H.-J. Kang, and V. Bevilacqua, editors, *Emerging Intelligent Computing Technology and Applications. With Aspects of Artificial Intelligence*, pages 327–340, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

H. A. Le Thi and T. Pham Dinh. DC programming and DCA: thirty years of developments. *Mathematical Programming*, 169(1):5–68, May 2018.

H. A. Le Thi, V. N. Huynh, and T. Pham Dinh. Stochastic difference-of-convex algorithms for solving nonconvex optimization problems visualization. https://arxiv.org/abs/1911.04334. a.

H. A. Le Thi, H. M. Le, D. N. Phan, and B. Tran. Stochastic DCA for minimizing a large sum of DC functions with application to multi-class logistic regression learning systems revised version in neural networks. b.

H. A. Le Thi, T. Pham Dinh, and H. P. Hau Luu. DC programming and DCA: from deterministic to stochastic approaches. *2nd ed.Springer*, c.

H. A. Le Thi, T. Pham Dinh, and M. Le Dung. Exact penalty in d.c. programming. *Vietnam Journal of Mathematics*, 27(2):169–178, 1999.

H. A. Le Thi, T. Belghiti, and T. Pham Dinh. Clustering via DC programming and DCA. In *Procceding des 13ème Rencontres de la Société Francophone de Classification*, pages 133–149, 2006.

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*, 37(4): 593–608, 2007a.

H. A. Le Thi, M. Le Hoai, and T. Pham Dinh. Fuzzy clustering based on nonconvex optimisation approaches using difference of convex (DC) functions algorithms. *Advances in Data Analysis and Classification*, 1(2):85–104, 2007b.

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*, 183 (3):1067–1085, 2007c.

H. A. Le Thi, N. Trong Phuc, and T. Pham Dinh. A continuous DC programming approach to the strategic supply chain design problem from qualified partner set. 183:1001–1012, 12 2007d.

H. A. Le Thi, M. Le Hoai, T. P. Nguyen, and T. Pham Dinh. Noisy image segmentation by a robust clustering algorithm based on DC programming and DCA. In P. Perner, editor, *Advances in Data Mining. Medical Applications, E-Commerce, Marketing, and Theoretical Aspects*, pages 72–86, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

H. A. Le Thi, M. Moeini, and T. Pham Dinh. DC programming approach for portfolio optimization under step increasing transaction costs. *Optimization*, 58(3):267–289, 2009.

H. A. Le Thi, T. Pham Dinh, and H. Van Ngai. Exact penalty and error bounds in DC programming. 52:509–535, 03 2011.

H. A. Le Thi, P. Damel, N. Peltre, and N. T. Phuc. The confrontation of two clustering methods in portfolio management: Ward's method versus DCA method. In T. van Do, H. A. L. Thi, and N. T. Nguyen, editors, *Advanced Computational Methods for Knowledge Engineering*, pages 87–98, Cham, 2014a. Springer International Publishing.

H. A. Le Thi, V. N. Huynh, and T. Pham Dinh. DC programming and DCA for general DC programs. In T. van Do, H. A. L. Thi, and N. T. Nguyen, editors, *Advanced Computational Methods for Knowledge Engineering*, pages 15–35, Cham, 2014b. Springer International Publishing.

H. A. Le Thi, M. Le Hoai, and T. Pham Dinh. New and efficient DCA based algorithms for minimum sum-of-squares clustering. *Pattern Recognition*, 47(1):388–401, 2014c.

H. A. Le Thi, D. M. Nguyen, and T. Pham Dinh. A DC programming approach for planning a multisensor multizone search for a target. *Computers & Operations Research*, 41:231–239, 2014d.

H. A. Le Thi, T. Pham Dinh, and T. Belghiti. DCA based algorithms for multiple sequence alignment (MSA). *Central European Journal of Operations Research*, 22 (3), 2014e.

H. A. Le Thi, Q. Tran Duc, and K.-H. Adjallah. A difference of convex functions algorithm for optimal scheduling and real-time assignment of preventive maintenance jobs on parallel processors. *Journal of Industrial and Management Optimization*, 2014f.

H. A. Le Thi, M. Le Hoai, and T. Pham Dinh. Feature selection in machine learning: an exact penalty approach using a difference of convex functions algorithm. *Machine Learning*, 101:163–186, 2015.

H. A. Le Thi, H. M. Le, D. N. Phan, and B. Tran. Stochastic DCA for the large-sum of non-convex functions problem and its application to group variable selection in classification visualization. http://proceedings.mlr.press/v70/thi17a/thi17a.pdf. pages 3394–3403. ICML, Proceedings of the 34th International Conference on Machine Learning, 2017.

H. A. Le Thi, H. Le, D. N. Phan, and B. Tran. A DCA-Like algorithm and its accelerated version with application in data visualization. https://arxiv.org/abs/1806.09620. 06 2018.

C. E. Lemke and K. Spielberg. Direct search algorithms for zero-one and mixed-integer programming. *Oper. Res.*, 15(5):892–914, 1967.

T. Liu, Z. Luo, H. Qin, and A. Lim. A branch-and-cut algorithm for the two-echelon capacitated vehicle routing problem with grouping constraints. *European Journal of Operational Research*, 266(2):487–497, 2018.

A. Lokketangen and F. Glover. Solving zero-one mixed integer programming problems using tabu search. *European Journal of Operational Research*, 106(2):624–658, 1998.

D. Lowe. Local feature view clustering for 3D object recognition. volume 1, pages I–682, 2001.

P. R. P. M Sasikumar, Dinesh Shikhare. *Introduction to Parallel Processing*. PHI Learning Private Limited, 2014.

M. Mahdavi and H. Abolhassani. Harmony k-means algorithm for document clustering. *Data Mining and Knowledge Discovery*, 18(3):370–391, 2009.

A. S. Mamaghani and M. R. Meybodi. Solving the quadratic assignment problem with the modified hybrid pso algorithm. In *2012 6th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–6, 2012.

H. Marchand, A. Martin, R. Weismantel, and L. Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1):397–446, 2002.

A. Martin. *General Mixed Integer Programming: Computational Issues for Branch-and-Cut Algorithms*, pages 1–25. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

U. Maulik and S. Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern Recognition*, 33(9):1455–1465, 2000.

S. Merendino and M. E. Celebi. A simulated annealing clustering algorithm based on center perturbation using gaussian mutation. *FLAIRS 2013 - Proceedings of the 26th International Florida Artificial Intelligence Research Society Conference*, pages 456–461, 2013.

O. d. Merle, P. Hansen, B. Jaumard, and N. Mladenović. An interior point algorithm for minimum sum-of-squares clustering. *SIAM J. Sci. Comput.*, 21(4):1485–1505, 1999.

P. Merz and B. Freisleben. Genetic algorithms for binary quadratic programming. 04 1999.

A. Misevicius. A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, 30:95–111, 2005.

A. Misevicius and E. Staneviciene. A new hybrid genetic algorithm for the grey pattern quadratic assignment problem. *Information Technology And Control*, 47, 2018.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

V. Nwana, K. Darby-Dowman, and G. Mitra. A co-operative parallel heuristic for mixed zero-one linear programming: Combining simulated annealing with branch and bound. *European Journal of Operational Research*, 164(1):12–23, 2005.

I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, Oct 1996.

P. M. Pardalos and Rodgers. Parallel branch and bound algorithms for quadratic zero–one programs on the hypercube architecture. *Annals of Operations Research*, 22(1):271–292, 1990.

S. Paterlini and T. Krink. High performance clustering with differential evolution. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, volume 2, pages 2004–2011, 2004.

T. Pham Dinh and S. El Bernoussi. Algorithms for solving a class of nonconvex optimization problems. methods of subgradients. In J.-B. Hiriart-Urruty, editor, *Fermat Days 85: Mathematics for Optimization*, volume 129 of *North-Holland Mathematics Studies*, pages 249–271. North-Holland, 1986.

T. Pham Dinh and H. A. Le Thi. Convex analysis approach to D.C. programming: Theory, algorithm and applications. *Acta Mathematica Vietnamica*, 22:289–355, 1997a.

T. Pham Dinh and H. A. Le Thi. Convex analysis approach D. C. programming: theory, algorithms and applications. *Acta mathematica vietnamica*, 22(1):289–355, 1997b. Dedicated to Hoang Tuy on the occasion of his seventieth birthday.

T. Pham Dinh and H. A. Le Thi. Recent advances in DC programming and DCA. In N. T. Nguyen and H. A. Le Thi, editors, *Transactions on Computational Intelligence XIII*, pages 1–37. Springer Berlin Heidelberg, 2014.

T. Pham Dinh, H. A. Le Thi, and F. Akoa. Combining DCA (DC algorithms) and interior point techniques for large-scale nonconvex quadratic programming. *Optimization Methods and Software*, 23(4):609–629, 2008.

T. Pham Dinh, N. Nguyen Canh, and H. A. Le Thi. An efficient combined DCA and B&B using DC/SDP relaxation for globally solving binary quadratic programs. *Journal of Global Optimization*, 48(4):595–632", 2010.

D. N. Phan, M. Le Hoai, and H. A. Le Thi. Accelerated difference of convex functions algorithm and its application to sparse binary logistic regression. pages 1369–1375. IJCAI, Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018.

J.-Y. Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63(3):337–370, Jun 1996.

Y. Qi and S. Sen. The ancestral benders' cutting plane algorithm with multi-term disjunctions for mixed-integer recourse decisions in stochastic programming. *Mathematical Programming*, 161(1/2):193–235, 2017.

N. Quang Thuan and H. A. Le Thi. Solving an inventory routing problem in supply chain by DC programming and DCA. pages 432–441, 04 2011.

M. Ramadas and A. Abraham. *Metaheuristics for Data Clustering and Image Segmentation*, volume 152. Intelligent Systems Reference Library, 2019.

Y. Rochat and É. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, 1995.

R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

B. Rostami, F. Errico, and A. Lodi. A convex reformulation and an outer approximation for a class of binary quadratic program. Technical report, 03 2018.

A. Salehipour, K. Sörensen, P. Goos, and O. Bräysy. Efficient GRASP+VND and GRASP+VND metaheuristics for the traveling repairman problem. *4OR*, 9(2):189–209, 2011.

G. Schryen. Parallel computational optimization in operations research: A new integrative framework, literature review and research directions. *European Journal of Operational Research*, 01 2020.

S. Z. Selim and K. Alsultan. A simulated annealing algorithm for the clustering problem. *Pattern Recognition*, 24(10):1003–1008, 1991.

P. Shelokar, V. Jayaraman, and B. Kulkarni. An ant colony approach for clustering. *Analytica Chimica Acta*, 509(2):187–195, 2004.

H. D. Sherali and J. Desai. A global optimization rlt-based approach for solving the hard clustering problem. *Journal of Global Optimization*, 32(2):281–306, Jun 2005.

J. E. Smith. *Genetic Algorithms*, pages 275–362. Springer US, Boston, MA, 2002.

M. Snir, W. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The complete reference*, volume 1. 01 1996.

R. Soto, B. Crawford, B. Almonacid, and F. Paredes. Efficient parallel sorting for migrating birds optimization when solving machine-part cell formation problems. *Sci. Program.*, 2016:39, 2016.

M. T. Ta, H. A. Le Thi, and L. Boudjeloud-Assala. Clustering data stream by a sub-window approach using DCA. In P. Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, volume 7376, pages 279–292, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

T. M. Thuy and H. A. Le Thi. An improvement of stability based method to clustering. In H. A. Le Thi, N. T. Nguyen, and T. V. Do, editors, *Advanced Computational Methods for Knowledge Engineering*, volume 358, pages 129–140, Cham, 2015. Springer International Publishing.

T. M. Thuy, H. A. Le Thi, and L. Boudjeloud-Assala. An efficient clustering method for massive dataset based on DC programming and DCA approach. In M. Lee, A. Hirose, Z.-G. Hou, and R. M. Kil, editors, *Neural Information Processing*, volume 8227, pages 538–545, Berlin, Heidelberg, 2013a. Springer Berlin Heidelberg.

T. M. Thuy, H. A. Le Thi, and L. Boudjeloud-Assala. Clustering data streams over sliding windows by DCA. In N. T. Nguyen, T. van Do, and H. A. le Thi, editors, *Advanced Computational Methods for Knowledge Engineering*, volume 479, pages 65–75, Heidelberg, 2013b. Springer International Publishing.

V. Tongur and E. Ülker. Migrating birds optimization for flow shop sequencing problem. *Journal of Computer and Communications*, 02:142–147, 01 2014.

Van-Dat Cung, T. Mautor, P. Michelon, and A. Tavares. A scatter search based approach for the quadratic assignment problem. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pages 16–169, 1997.

T. J. Van Roy and L. A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Oper. Res.*, 35(1):45–57, 1987.

X. T. Vo, H. A. Le Thi, and T. Pham Dinh. Robust optimization for clustering. In N. T. Nguyen, B. Trawiński, H. Fujita, and T.-P. Hong, editors, *Intelligent Information and Database Systems*, volume 9622, pages 671–680, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

S. Voß. Meta-heuristics: The state of the art. In A. Nareyek, editor, *Local Search for Planning and Scheduling*, pages 1–23, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

Y. Xia and J. Peng. A cutting algorithm for the minimum sum-of-squared error clustering. pages 150–160, 2005.

M. Yagouni and H. A. Le Thi. A collaborative metaheuristic optimization scheme: Methodological issues. In T. van Do, H. A. L. Thi, and N. T. Nguyen, editors, *Advanced Computational Methods for Knowledge Engineering*, pages 3–14, Cham, 2014. Springer International Publishing.

X. Yang and Suash Deb. Cuckoo search via lévy flights. In *2009 World Congress on Nature Biologically Inspired Computing (NaBIC)*, pages 210–214, 2009.

C. Zhang, Z. Lin, and Z. Lin. Variable neighborhood search with permutation distance for QAP. In R. Khosla, R. J. Howlett, and L. C. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, pages 81–88, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.